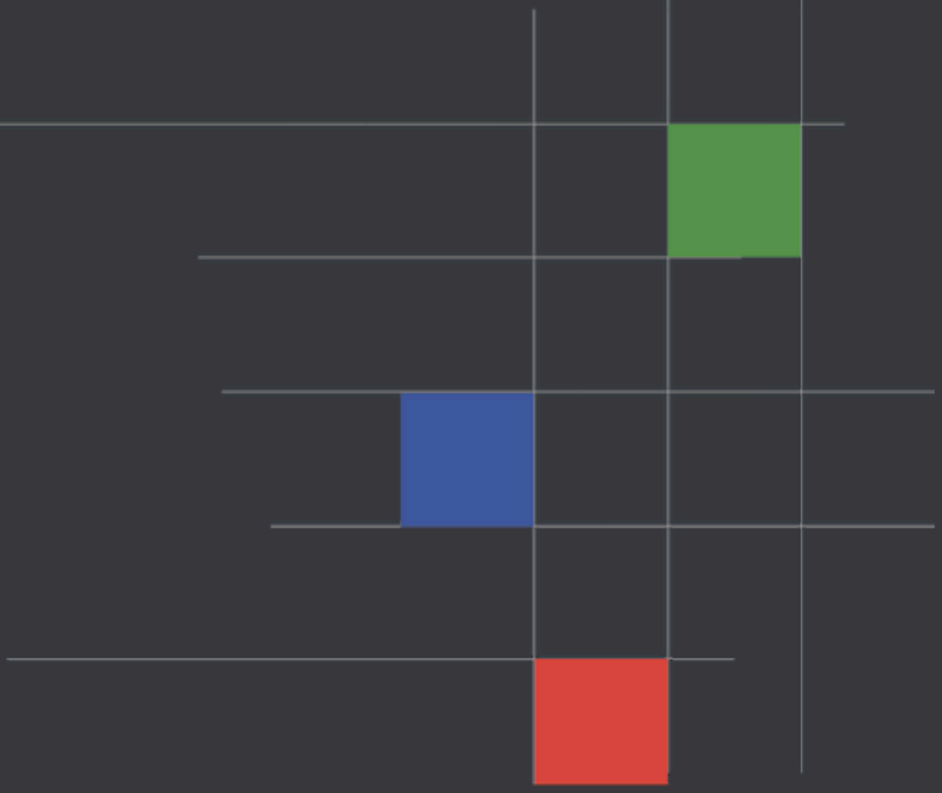
# Monster Slayer Finance

## Security Assessment

March 16, 2021

For :
Monster Slayer Finance



# Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

### What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.

# Overview

Project Summary

| | |
|---|---|
| **Project Name** | Monster Slayer Finance |
| **Description** | DeFi |
| **Platform** | Binance Smart Chain; Solidity |
| **Codebase** | Private Repository |
| **Commit** | Boardroom.sol:<br>37129bd78d3ed983ddcc73b43810b44d46d38d2147d3c7285faad33c11d781ab<br>ContractGuard.sol:<br>d8c38e84fe213116afd6514723fb177bbc02191d121793108bb4135ad917de6d<br>IBasisAsset.sol:<br>7edf6080ba6400119e5ac86c7552b64d5b7e877fcc9ebc3aa634b2af8b25c229<br>IERC20.sol:<br>397534ef6f97a7fa11089b8f05cf5383749067352d06c0e39b880fd08743489a<br>ITreasury.sol:<br>281dbb71439a816e0af2d13a72bb5894a2c00fdac873fef288dcd4b4d18de52a<br>Treasury.sol:<br>3bedf30164496d78d0dd23b5a0e3c3e87d1aaab748f3ac364eec10c04ff2d43f<br>Math.sol:<br>534e4353a4e96ae4097f81e1a620faa5464222db5bb9d9b567a92f58d31813fe<br>SafeERC20.sol:<br>de150078598d137b0e456bb0ef0eb220a748cadc7b648999b546be4187556bf2<br>SafeMath.sol:<br>a644e4558659f5d8d58f4dded8d284d5a48ea3a0f75ed34a6bb73237a4499665 |

## Audit Summary

| | |
|---|---|
| **Delivery Date** | Mar. 7th, 2021 |
| **Method of Audit** | Static Analysis, Manual Review |
| **Consultants Engaged** | 2 |
| **Timeline** | Mar. 1, 2021 - Mar. 7, 2021, Mar 16, 2021 |

## Vulnerability Summary

| | |
|---|---|
| **Total Issues** | 6 |
| **Total Critical** | 0 |
| **Total Major** | 0 |
| **Total Medium** | 0 |
| **Total Minor** | 2 |
| **Total Informational** | 4 |

# Executive Summary

This report has been prepared for **Monster Slayer Finance** smart contract to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

All of the functions in the protocol have proper access restriction and parameter sanitization where necessary. The equity was found to be calculated correctly for each of the accounts. Most of the findings are optimizational.

Additionally, to bridge the trust gap between administrator and users, administrator needs to express a sincere attitude with the consideration of the administrator team's anonymousness. The administrator has the responsibility to notify users with the following capability of the `operator` :

- `operator` can transfer any amount of `_token` assests to any addresses through `governanceRecoverUnsupported()` function in `Boardroom.sol` and `Treasury.sol` smart contract.
- `operator` can update `withdrawLockupEpochs` and `rewardLockupEpochs` through `setLockUp()` function in `Boardroom.sol` smart contract.
- `operator` can update `setMaxClaimAndWithdraw` and `maxWithdrawWhenDollarPriceLessThanOne` through `setMaxClaimAndWithdraw()` function in `Boardroom.sol` smart contract.
- `operator` can update the `boardroom` implementation address through `setBoardroom()` function in `Treasury.sol` smart contract.
- `operator` can update the `dollarOracle` implementation address through `setDollarOracle()` function in `Treasury.sol` smart contract.
- `operator` can update the value of `dollarPriceCeiling` through `setDollarPriceCeiling()` function in `Treasury.sol` smart contract.
- `operator` can update the value of `dollarPriceMaxPremium` through `setDollarPriceMaxPremium()` function in `Treasury.sol` smart contract.
- `operator` can update the value of `maxSupplyExpansionPercent` and `maxSupplyExpansionPercentInDebtPhase` through `setMaxSupplyExpansionPercents()` function in `Treasury.sol` smart contract.
- `operator` can update the value of `bondDepletionFloorPercent` through `setBondDepletionFloorPercent()` function in `Treasury.sol` smart contract.
- `operator` can update the value of `seigniorageExpansionFloorPercent` through `setSeigniorageExpansionFloorPercent()` function in `Treasury.sol` smart contract.
- `operator` can update the value of `maxSupplyContractionPercent` through `setMaxSupplyContractionPercent()` function in `Treasury.sol` smart contract.

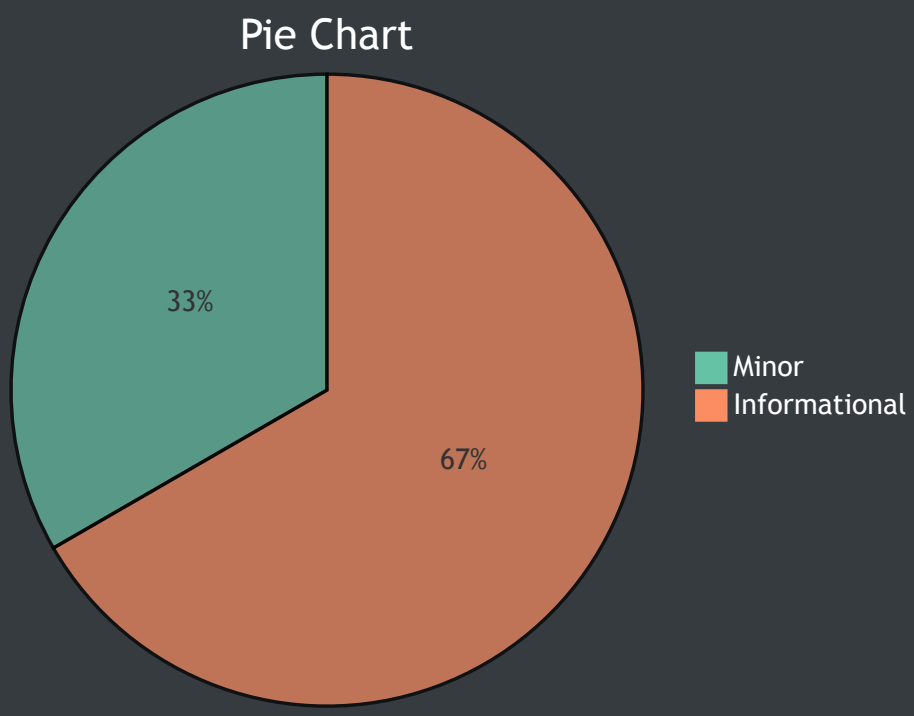
- `operator` can update the value of `maxDebtRatioPercent` through `setMaxDebtRatioPercent()` function in `Treasury.sol` smart contract.
- `operator` can update the value of `stablizationFund` through `setStablizationFund()` function in `Treasury.sol` smart contract.
- `operator` can update the value of `stablizationFundSharedPercent` and `stablizationFundSharedPercentInDebtPhase` through `setStablizationFundSharedPercent()` function in `Treasury.sol` smart contract.
- `operator` can migrate dollar, bond and share to arbitary `target` address through `migrate()` function in `Treasury.sol` smart contract.
- `operator` can update the `_operator` of the `boardroom` contract through `boardroomSetOperator()` function in `Treasury.sol` smart contract.
- `operator` can update the value of `_withdrawLockupEpochs` and `_rewardLockupEpochs` of the `boardroom` contract through `boardroomSetLockUp()` function in `Treasury.sol` smart contract.
- `operator` can update the value of `_maxRewardClaimWhenDollarPriceLessThanOne` and `_maxWithdrawWhenDollarPriceLessThanOne` of the `boardroom` contract through `boardroomSetMaxClaimAndWithdraw()` function in `Treasury.sol` smart contract.
- `operator` can update the allocateSeigniorage of the `boardroom` contract through `boardroomAllocateSeigniorage()` function in `Treasury.sol` smart contract.
- `operator` can transfer any amount of `_token` assests to any addresses through `governanceRecoverUnsupported()` function in `Boardroom.sol` through `boardroomGovernanceRecoverUnsupported()` function in `Treasury.sol` smart contract

As implementation of `cash`, `share` protocols are not in scope of the auditing, we advise client to deploy correct `cash`, `share` contracts and inform project's community to improve the trustworthiness of the project. Moreover, any dynamic runtime changes on the protocol should be notified to the community. We also advise client to adopt Multisig, Timelock and/or DAO in the project.

## File in Scope

| ID | Contract | SHA-256 Checksum |
|---|---|---|
| BRM | Boardroom.sol | 37129bd78d3ed983ddcc73b43810b44d46d38d2147d3c7285faad33c11d781ab |
| CTG | ContractGuard.sol | d8c38e84fe213116afd6514723fb177bbc02191d121793108bb4135ad917de6d |
| IBA | IBasisAsset.sol | 7edf6080ba6400119e5ac86c7552b64d5b7e877fcc9ebc3aa634b2af8b25c229 |
| IERC | IERC20.sol | 397534ef6f97a7fa11089b8f05cf5383749067352d06c0e39b880fd08743489a |
| ITR | ITreasury.sol | 281dbb71439a816e0af2d13a72bb5894a2c00fdac873fef288dcd4b4d18de52a |
| TSY | Treasury.sol | 3bedf30164496d78d0dd23b5a0e3c3e87d1aaab748f3ac364eec10c04ff2d43f |
| Math | Math.sol | 534e4353a4e96ae4097f81e1a620faa5464222db5bb9d9b567a92f58d31813fe |
| SERC | SafeERC20.sol | de150078598d137b0e456bb0ef0eb220a748cadc7b648999b546be4187556bf2 |
| SMATH | SafeMath.sol | a644e4558659f5d8d58f4dded8d284d5a48ea3a0f75ed34a6bb73237a4499665 |

## Findings




| ID | Title | Type | Severity | Resolved |
|---|---|---|---|---|
| BRM-01 | Missing Emit Event | Optimization | Informational | ! |
| BRM-02 | Lack of input validation | Volatile Code | Minor | ! |
| BRM-03 | Proper Usage of `public` and `external` Type | Gas Optimization | Informational | ! |
| TSY-01 | State Variable never Initialized before Usage | Optimization | Informational | ! |
| TSY-02 | Lack of input validation | Volatile Code | Minor | ! |
| TSY-03 | Missing Emit Event | Optimization | Informational | ! |

## BRM-01: Missing Emit Event

| Type | Severity | Location |
| --- | --- | --- |
| Optimization | Informational | Boardroom.sol L135, L139 |

### Description:

Functions that affect the status of sensitive variable should be able to emit event as notification to customers

Examples:

- `setOperator()`
- `setLockUp()`

### Recommendation:

Consider adding event for sensitive action, and emit it in the function like below.

```
event SetOperator(address indexed _operator, address indexed _caller);

function setOperator(address _operator) external onlyOperator {
    operator = _operator;
    emit SetOperator(_operator, msg.sender);
}
```

### Alleviation:

Client acknowledged the finding and decided to implement it in the future if they were to do a major update to the contract.

## BRM-02: Lack of input validation

| Type | Severity | Location |
| --- | --- | --- |
| Volatile Code | Minor | Boardroom.sol L135, L267 |

### Description:

The assigned value to `operator` should be verified as non zero value to prevent being mistakenly assigned as `address(0)` in the constructor of the contract. Violation of this may cause losing ownership of `operator` authorization.

The value of parameter `_to` in function `governanceRecoverUnsupported()` should also be verified as non zero value to prevent assets being sent to `address(0)` .

### Recommendation:

Check that the address is not zero by adding following check in the constructor of the contract.

```
require(_operator != address(0), "Operator's address must not be address(0)");
```

And add following check in function `governanceRecoverUnsupported()`

```
require(_to != address(0), "Asset should not be sent to address(0)");
```

### Alleviation:

Client acknowledged the finding and decided to implement it in the future if they were to do a major update to the contract.

## BRM-03: Proper Usage of "public" and "external" Type

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | Boardroom.sol L195 |

### Description:

"public" functions that are never called by the contract could be declared "external" . When the inputs are arrays "external" functions are more efficient than "public" functions.
Examples:

Functions `rewardPerShare()` in contract `Boardroom.sol` .

### Recommendation:

Consider using the "external" attribute for functions never called from the contract.

### Alleviation:

Client acknowledged the finding and decided to implement it in the future if they were to do a major update to the contract.

## TSY-01: State Variable never Initialized before Usage

| Type | Severity | Location |
| --- | --- | --- |
| Optimization | Informational | Treasury.sol L364 |

### Description:

Some variables do not initialize explicitly before usage.

- `uint256 price` at L158
- `uint256 _savedForBond` at L364

### Recommendation:

We advise client to consider initializing all the variables explicitly. If a variable is meant to be initialized to zero, explicitly set it to zero.

### Alleviation:

Client acknowledged the finding and decided to implement it in the future if they were to do a major update to the contract.

## TSY-02: Lack of input validation

| Type | Severity | Location |
| --- | --- | --- |
| Volatile Code | Minor | Treasury.sol L110, L192, L196, L200, L260, L397, L411, L427 |

### Description:

The assigned value to `dollar`, `bond`, `share`, `dollarOracle` and `stablizationFund` should be verified as non zero value to prevent being mistakenly assigned as `address(0)` in the constructor of the contract. Violation of this may cause losing ownership of `operator` authorization.

Similar sanity check should also be conducted on following parameters and functions:

- `_operator` in function `setOperator()`
- `_boardroom` in function `setBoardroom()`
- `_dollarOracle` in function `setDollarOracle()`
- `target` in function `migrate()`
- `_to` in function `governanceRecoverUnsupported()`
- `_operator` in function `boardroomSetOperator()`
- `_to` in function `boardroomGovernanceRecoverUnsupported()`

### Recommendation:

We advise client to check that the addresses are not zero by adding following check for all abovementioned parameters in the constructor of the contract as well as functions.

```
1  require(_operator != address(0), "Operator's address must not be address(0)");
```

### Alleviation:

Client acknowledged the finding and decided to implement it in the future if they were to do a major update to the contract.

## TSY-03: Missing Emit Event

| Type | Severity | Location |
| --- | --- | --- |
| Optimization | Informational | Treasury.sol L135, L139 |

### Description:

Functions that affect the status of sensitive variable should be able to emit event as notification to customers

Examples:

- `setOperator()`
- `setBoardroom()`
- `setDollarOracle()`
- `setDollarPriceCeiling()`
- `setDollarPriceMaxPremium()`
- `setMaxSupplyExpansionPercents()`
- `setBondDepletionFloorPercent()`
- `setSeigniorageExpansionFloorPercent()`
- `setMaxSupplyContractionPercent()`
- `setMaxDebtRatioPercent()`
- `setStablizationFund()`
- `migrate()`
- `boardroomSetOperator()`
- `boardroomSetLockUp()`
- `boardroomSetMaxClaimAndWithdraw()`
- `boardroomAllocateSeigniorage()`
- `boardroomGovernanceRecoverUnsupported()`

### Recommendation:

Consider adding event for sensitive action, and emit it in the function like below.

```
event SetOperator(address indexed _operator, address indexed _caller);

function setOperator(address _operator) external onlyOperator {
    operator = _operator;
    emit SetOperator(_operator, msg.sender);
}
```

Alleviation:

Client acknowledged the finding and decided to implement it in the future if they were to do a major update to the contract.

## Appendix

---

## Finding Categories

#### Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

#### Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

#### Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

#### Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

#### Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

#### Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

#### Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete` .

#### Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

**Inconsistency**

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

**Magic Numbers**

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

**Compiler Error**

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

**Dead Code**

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

---

## Icons explanation

 : Issue resolved

 : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

 : Issue partially resolved. Not all instances of an issue was resolved.