mStable

Smart Contracts

Security Assessment

January 18th, 2021

By:
Sheraz Arshad @ CertiK
sheraz.arshad@certik.org

Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

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	mStable
Description	The audited contracts comprise Savings and Staking contracts. The Savings contract can be deposited with mAsset and imUSD are received which can be staked in the Staking contract to earn MTA. The mAssets deposited in the Savings contract earn interests and the redeemer of mAssets benefit from the increased exhchange rate from the deposited interest in the Savings contract.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. e6d2dfe2823e8b51a24c9a8a824b38ce996930de 2. 9284431fa5d1c0f315c415061ea47f9a9c677acc

Audit Summary

Delivery Date	Jan. 18, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Jan. 8, 2020 - Jan. 18, 2021

Vulnerability Summary

Total Issues	18
Total Critical	0
Total Major	1
Total Medium	4
Total Minor	1
Total Informational	12

Executive Summary

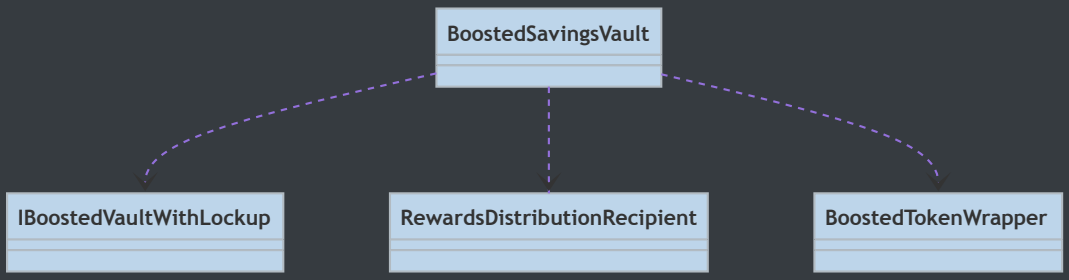
This report represents the results of CertiK's engagement with mStable on their implementation of the Savings smart contracts.

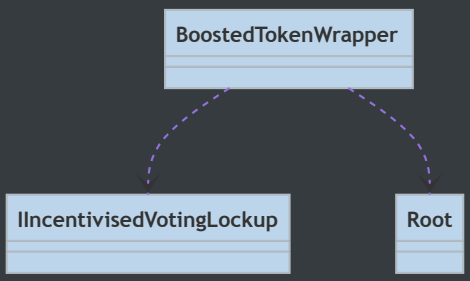
Our findings mainly refer to optimizations and a few logical issues. All the non-informational and most of the informational findings were remediated. The overall security of the contracts can be deemed as high and the identified issues pose no threat to the safety of the contracts.

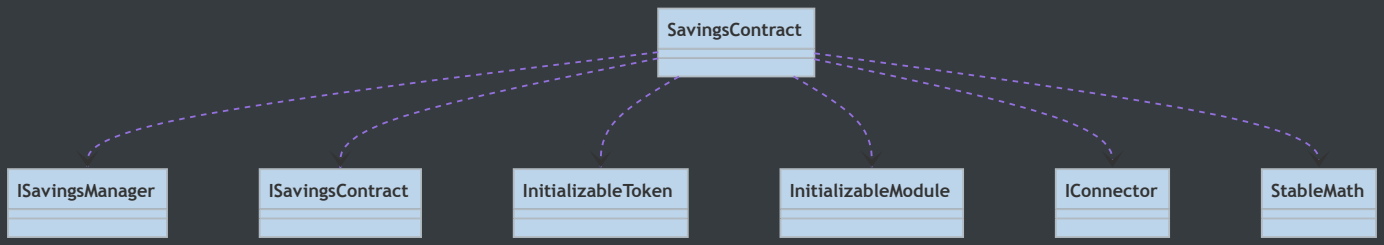
Files In Scope

ID	Contract	Location
BSV	BoostedSavingsVault.sol	contracts/savings/BoostedSavingsVault.sol
BTW	BoostedTokenWrapper.sol	contracts/savings/BoostedTokenWrapper.sol
ICR	IConnector.sol	contracts/savings/peripheral/IConnector.sol
SCT	SavingsContract.sol	contracts/savings/SavingsContract.sol

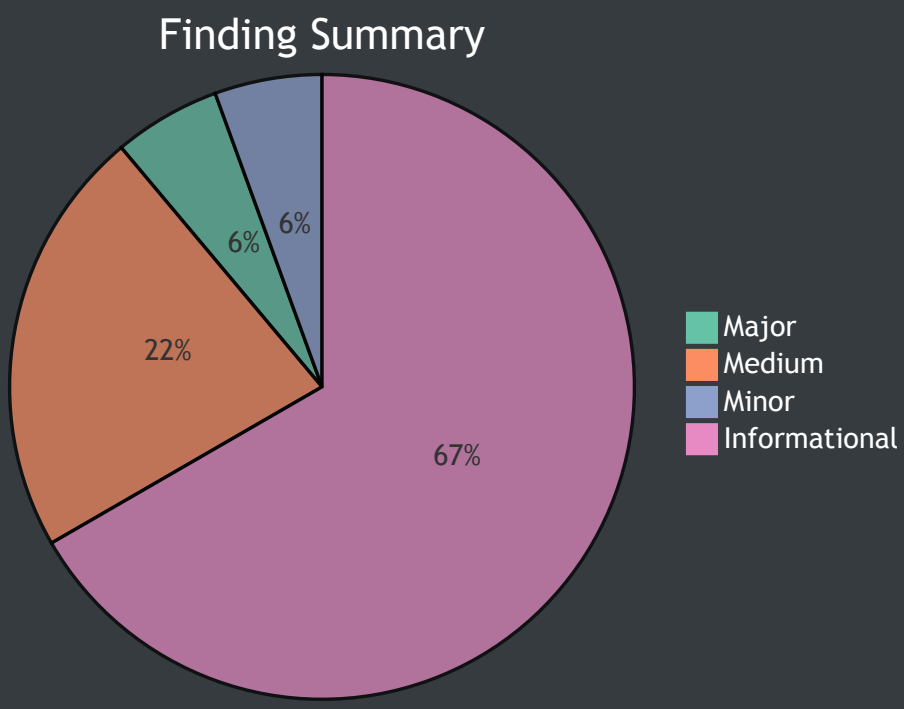
File Dependency Graph (BETA)







Findings



ID	Title	Type	Severity	Resolved
SCT-01	Lack of verification for the passed argument	Logical Issue	Medium	✓
SCT-02	Lack of verification for the passed argument	Logical Issue	Medium	✓
SCT-03	Documentation discrepancy	Inconsistency	Informational	✓
SCT-	Redundant Variable Initialization	Coding Style		✓

04			Informational	
SCT-05	Incorrect function argument is passed	Logical Issue	Major	✓
SCT-06	Possibility of faulty exchange rate calculation	Volatile Code	Medium	✓
SCT-07	Return Variable Utilization	Gas Optimization	Informational	✓
BTW-01	mappings data can be packed in a struct	Gas Optimization	Informational	✓
BTW-02	Comparison with a minimum value should be greater-than-or-equal	Logical Issue	Minor	✓
BTW-03	Redundant `require` statement	Gas Optimization	Informational	✓
BTW-04	Explicitly returning a local variable	Gas Optimization	Informational	✓
BSV-01	Redundant Variable Initialization	Coding Style	Informational	✓
BSV-02	Return Variable Utilization	Gas Optimization	Informational	✓
BSV-03	Explicitly returning a local variable	Gas Optimization	Informational	✓
BSV-04	Lack of verification for the passed argument	Logical Issue	Medium	✓
BSV-05	mappings data can be packed in a struct	Gas Optimization	Informational	✓
BSV-06	Inefficient struct layout	Gas Optimization	Informational	✓

BSV-07	Documentation discrepancy	Inconsistency	Informational	✓

SCT-01: Lack of verification for the passed argument

Type	Severity	Location
Logical Issue	Medium	SavingsContract.sol L206

Description:

The function `preDeposit` on the aforementioned line receives `_beneficiary` as its parameter. There is no check in place in the code flow to verify that the address is not zero which can result in unretrievable minted tokens to zero address if it is passed to the function.

Recommendation:

We recommend that a check is added in the code to assert that the `_beneficiary` is not a zero address.

```
require(
  _beneficiary != address(0),
  "_beneficiary cannot be a zero address"
);
```

Recommendation:

Alleviations were applied as advised in the commit `9284431fa5d1c0f315c415061ea47f9a9c677acc`.

SCT-02: Lack of verification for the passed argument

Type	Severity	Location
Logical Issue	Medium	SavingsContract.sol L238

Description:

The function `depositSavings` on the aforementioned line receives `_beneficiary` as its parameter. There is no check in the code flow to verify that the address is not zero which can result in unretrievable minted tokens to zero address if it is passed to the function.

Recommendation:

We advise that a check is added in the code to assert that the `_beneficiary` is not a zero address.

```
require(
  _beneficiary != address(0),
  "_beneficiary cannot be a zero address"
);
```

Recommendation:

Alleviations were applied as advised in the commit `9284431fa5d1c0f315c415061ea47f9a9c677acc`.

SCT-03: Documentation discrepancy

Type	Severity	Location
Inconsistency	Informational	SavingsContract.sol L692

Description:

The comment on the aforementioned line describes the behaviour of the function following it incorrectly. It suggests that the function converts underlying to credits yet it converts credits to underlying.

Recommendation:

We advise to rectify the comment to correctly describe the behaviour of the function.

```
   Converts credits amount into mAsset  based on exchange rate
```

Recommendation:

Alleviations were applied as advised in the commit `9284431fa5d1c0f315c415061ea47f9a9c677acc`.

SCT-04: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	SavingsContract.sol L353-L355

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint / int` : All `uint` and `int` variable types are initialized at `0`
- `address` : All `address` types are initialized to `address(0)`
- `byte` : All `byte` types are initialized to their `byte(0)` representation
- `bool` : All `bool` types are initialized to `false`
- `ContractType` : All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct` : All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Recommendation:

Alleviations were applied as advised in the commit `9284431fa5d1c0f315c415061ea47f9a9c677acc`.

SCT-05: Incorrect function argument is passed

Type	Severity	Location
Logical Issue	Major	SavingsContract.sol L549

Description:

The function call `_validateCollection(lastBalance_, connectorBalance.sub(lastBalance_), timeSinceLastPoke);` on the aforementioned line passes incorrect first argument of `lastBalance_` which can result in incorrect `extrapolatedAPY` calculated in the function's body as the function `_validateCollection` expects first parameter to be the latest balance.

Recommendation:

We recommend to correctly pass the first argument to the function call by providing the latest balance i.e. connectorBalance.

```
_validateCollection(connectorBalance, connectorBalance.sub(lastBalance_), timeSinceLastPoke);
```

Recommendation:

Alleviations were applied as advised in the commit `9284431fa5d1c0f315c415061ea47f9a9c677acc` .

SCT-06: Possibility of faulty exchange rate calculation

Type	Severity	Location
Volatile Code	Medium	SavingsContract.sol L575

Description:

The function call `_refreshExchangeRate(sum, _data.totalCredits, false);` on the aforementioned line receives `sum` as first argument which is the sum of underlying balance in the contract itself and the connector, prior to connector's balance is adjusted to the `ideal`. During the connector's balance adjustment, the slippage or any fee associated with the deposit can result in the actual total underlying balance being less than the calculated total underlying balance on `L553`. When the balance is deposited/withdrawn from the connector, the connector immediately deposits/withdraws balance in Curve protocol and due to slippage/fee, the actual balance can be less than the `ideal` balance that we assume the connector would contain.

Recommendation:

We recommend that an invariant is added after the connector's balance adjustment which asserts that the connector's balance should be greater-than-or-equal to `ideal`. The `require` condition will get the latest connector's balance and then compare it with `ideal` so that the slippage did not affect the actual balance to the point where it is less than `ideal`.

```
require(
  connector_.checkBalance() >= ideal,
  "connector's balance cannot be less than ideal"
);
```

Recommendation:

Alleviations were applied as advised in the commit `9284431fa5d1c0f315c415061ea47f9a9c677acc`.

SCT-07: Return Variable Utilization

Type	Severity	Location
Gas Optimization	Informational	SavingsContract.sol L683

Description:

The linked function declarations contain explicitly named `return` variables that are not utilized within the function's code block.

Recommendation:

We advise that the linked variables are either utilized or omitted from the declaration.

Recommendation:

Alleviations were applied as advised in the commit `9284431fa5d1c0f315c415061ea47f9a9c677acc`.

BTW-01: mappings data can be packed in a struct

Type	Severity	Location
Gas Optimization	Informational	BoostedTokenWrapper.sol L34-L35

Description:

The mappings on the aforementioned lines have the key type of `address` representing users' addresses. These mappings can be combined into a single mapping having address as key type and a struct representing value types of both mappings, as its value type. This will reduce the lookup gas cost associated with the individual mappings.

Recommendation:

We advise to replace the aforementioned mappings with a single mapping by utilizing a struct for the value types.

```
struct UserBalance {
  uint256 boostedBalance;
  uint256 rawBalance;
}
```

```
mapping(address => UserBalance) private userBalances;
```

Recommendation:

The mStable development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

BTW-02: Comparison with a minimum value should be greater-than-or-equal

Type	Severity	Location
Logical Issue	Minor	BoostedTokenWrapper.sol L141

Description:

The comparison with minimum value inside `if` statement on the aforementioned line is greater-than which should be replaced with the greater-than-or-equal comparison to increase the legibility of the code.

Recommendation:

We recommend to do the comparison of greater-than-or-equal with the minimum value on the aforementioned line.

```
if (rawBalance >= MIN_DEPOSIT) {...}
```

Recommendation:

Alleviations were applied as advised in the commit `9284431fa5d1c0f315c415061ea47f9a9c677acc`.

BTW-03: Redundant `require` statement

Type	Severity	Location
Gas Optimization	Informational	BoostedTokenWrapper.sol L163

Description:

The `require` statement on the aforementioned line compares the deposit value with the minimum deposit, as this comparison is already performed on `L141`, so the `require` statement will never evaluate to `false` and hence is redundant.

Recommendation:

We advise to remove the redundant `require` statement on the aforementioned line.

Recommendation:

Alleviations were applied as advised in the commit `9284431fa5d1c0f315c415061ea47f9a9c677acc`.

BTW-04: Explicitly returning a local variable

Type	Severity	Location
Gas Optimization	Informational	BoostedTokenWrapper.sol L158

Description:

The function on the aforementioned line explicitly returns a local variable which increases the overall cost of gas.

Recommendation:

Since named return variables can be declared in the signature of a function, consider refactoring to remove the local variable declaration and explicit return statement in order to reduce the overall cost of gas.

Recommendation:

Alleviations were applied as advised in the commit `9284431fa5d1c0f315c415061ea47f9a9c677acc`.

BSV-01: Redundant Variable Initialization

Type	Severity	Location
Coding Style	● Informational	BoostedSavingsVault.sol L45, L47, L49, L51

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int` : All `uint` and `int` variable types are initialized at `0`
- `address` : All `address` types are initialized to `address(0)`
- `byte` : All `byte` types are initialized to their `byte(0)` representation
- `bool` : All `bool` types are initialized to `false`
- `ContractType` : All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct` : All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Recommendation:

Alleviations were applied as advised in the commit `9284431fa5d1c0f315c415061ea47f9a9c677acc`.

BSV-02: Return Variable Utilization

Type	Severity	Location
Gas Optimization	Informational	BoostedSavingsVault.sol L374

Description:

The linked function declarations contain explicitly named `return` variables that are not utilized within the function's code block.

Recommendation:

We advise that the linked variables are either utilized or omitted from the declaration.

Recommendation:

The mStable development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

BSV-03: Explicitly returning a local variable

Type	Severity	Location
Gas Optimization	Informational	BoostedSavingsVault.sol L434, L365, L374

Description:

The functions on the aforementioned lines explicitly return local variable which increases the overall cost of gas.

Recommendation:

Since named return variables can be declared in the signature of a function, consider refactoring to remove the local variable declaration and explicit return statement in order to reduce the overall cost of gas.

Recommendation:

The mStable development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

BSV-04: Lack of verification for the passed argument

Type	Severity	Location
Logical Issue	Medium	BoostedSavingsVault.sol L172

Description:

The function `stake` on the aforementioned line receives `_beneficiary` as its parameter. There is no check in place in the code flow to verify that the address is not zero which can result in unretrievable minted tokens to zero address if it is passed to the function.

Recommendation:

We recommend that a check is added in the code to assert that the `_beneficiary` is not a zero address.

```
require(
  _beneficiary != address(0),
  "_beneficiary cannot be a zero address"
);
```

Recommendation:

Alleviations were applied as advised in the commit `9284431fa5d1c0f315c415061ea47f9a9c677acc`.

BSV-05: mappings data can be packed in a struct

Type	Severity	Location
Gas Optimization	Informational	BoostedSavingsVault.sol L52-L55

Description:

The mappings on the aforementioned lines have the key type of `address` representing users' addresses. These mappings can be combined into a single mapping having address as key type and a struct representing value types of both mappings, as its value type. This will reduce the lookup gas cost associated with the individual mappings.

Recommendation:

We advise to replace the aforementioned mappings with a single mapping by utilizing a struct for the value types.

```
struct User {
  UserData data;
  Reward[] rewards;
  uint64 claim;
}
```

```
mapping(address => User) public users;
```

Recommendation:

The mStable development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

BSV-06: Inefficient struct layout

Type	Severity	Location
Gas Optimization	Informational	BoostedSavingsVault.sol L57-L62

Description:

The struct on the aforementioned line would utilize two 32-byte slots in the storage where the second 32-byte slot would only be occupied half. As EVM operates on 32-byte data and to operate on the second slot it will cost extra gas to pack the data in 16-byte as the slot is half empty.

Recommendation:

We advise to convert the type of `rewards` to `uint256` and be laid out as a last property in the struct which will result in the struct occupying two complete 32-byte slots resulting in reduced gas cost compared to previous implementation without compromising any functionality.

```
struct UserData {
  uint128 rewardPerTokenPaid;
  uint64 lastAction;
  uint64 rewardCount;
  uint256 rewards;
}
```

Recommendation:

The mStable development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

BSV-07: Documentation discrepancy

Type	Severity	Location
Inconsistency	Informational	BoostedSavingsVault.sol L145

Description:

The comment on the aforementioned line describes the behaviour of the `modifier` following it incorrectly. It suggests that the `modifier` updates reward for a given address yet it updates the `boost` for a given address.

Recommendation:

We advise to rectify the comment to correctly describe the behaviour of the function.

```
/** @dev Updates the boost for a given address, after executing function
*/
```

Recommendation:

Alleviations were applied as advised in the commit `9284431fa5d1c0f315c415061ea47f9a9c677acc`.

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 in-storage 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.