Saddle

Security Assessment

October 29th, 2020

For :
Saddle

By :
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

Angelos Apostolidis @ CertiK
angelos.apostolidis@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	Saddle
Description	The codebase contains a standard burnable ERC-20 token implementation, an ERC-20 token wrapper for the ones that can be staked and withdrawn, a pausable contract, a liquidity pool implementation, a StableSwap implementation and the respective mathematical utility contracts.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. a188f2b2b82f44b12e78e599b11b561530f01c0f

Audit Summary

Delivery Date	October 29th, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	October 3rd, 2020 - October 29th, 2020

Vulnerability Summary

Total Issues	27
Total Critical	0
Total Major	0
Total Medium	0
Total Minor	5
Total Informational	22

Executive Summary

The report represents the results of our engagement with the Thesis on their smart contracts behind Saddle.

Our findings mainly refer to optimizations and Solidity coding standards. Hence, the issues identified pose no threat to the safety of the contract deployement.

Files In Scope

ID	Contract	Location
ALL	Allowlist.sol	contracts/Allowlist.sol
CER	CERC20.sol	contracts/CERC20.sol
LPT	LPToken.sol	contracts/LPToken.sol
MUS	MathUtils.sol	contracts/MathUtils.sol
OPE	OwnerPausable.sol	contracts/OwnerPausable.sol
SWA	Swap.sol	contracts/Swap.sol
SUS	SwapUtils.sol	contracts/SwapUtils.sol
STW	StakeableTokenWrapper.sol	contracts/StakeableTokenWrapper.sol

Findings

ID	Title	Type	Severity	Resolved
ALL-01	Unlocked Compiler Version	Language Specific	Informational	✓
ALL-02	Inexistent Input Sanitization	Volatile Code	Minor	!✓
ALL-03	Visibility Specifiers Missing	Language Specific	Informational	✓
CER-01	Unlocked Compiler Version	Language Specific	Informational	✓
CER-02	Check Against Zero Address	Volatile Code	Informational	✓
LPT-01	Unlocked Compiler Version	Language Specific	Informational	✓
LPT-02	Inexistent Input Sanitization	Volatile Code	Minor	✓
MUS-01	Unlocked Compiler Version	Language Specific	Informational	✓
MUS-02	Function Optimization	Gas Optimization	Informational	✓
OPE-01	Unlocked Compiler Version	Language Specific	Informational	✓
OPE-02	Visibility Specifiers Missing	Language Specific	Informational	✓
STW-01	Unlocked Compiler Version	Language Specific	Informational	✓
STW-02	State Variables Optimization	Gas Optimization	Informational	✓
STW-03	Introduction of an `immutable` Variable	Language Specific	Informational	!✓
STW-04	User-Defined Getters	Gas Optimization	Informational	✓
STW-05	Inexistent Input Sanitization	Volatile Code	Minor	✓
SWA-01	Unlocked Compiler Version	Language Specific	Informational	✓
SWA-02	Omitted Parameter Description	Coding Style	Informational	✓

ID	Title	Type	Severity	Resolved
SUS-01	Unlocked Compiler Version	Language Specific	Informational	✓
SUS-02	`struct` Optimization	Gas Optimization	Informational	✓
SUS-03	Visibility Specifiers Missing	Language Specific	Informational	✓
SUS-04	Inexistent Input Sanitization	Volatile Code	Minor	!
SUS-05	`internal` Over `external` Functions	Gas Optimization	Informational	!
SUS-06	Function Optimization	Gas Optimization	Informational	✓
SUS-07	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓
SUS-08	Redundant Variable Initialization	Gas Optimization	Informational	✓
SUS-09	Inconsistent Mathematical Logic	Mathematical Operations	Minor	!

ALL-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Allowlist.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at.

Alleviation:

The Saddle development team opted to consider our references and locked the compiler to version `0.5.17`.

ALL-02: Inexistent Input Sanitization

Type	Severity	Location
Volatile Code	Minor	Allowlist.sol L57-L65, L67-L75

Description:

The linked functions can be manipulated by a malicious owner, as the input is not sanitized in both cases.

Recommendation:

We advise the team to add `require` statements in both functions to restrict the input values.

Alleviation:

The Saddle development team has acknowledged this exhibit but decided to not apply any remediation.

ALL-03: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	Informational	Allowlist.sol L16, L17, L18

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

The Saddle development team opted to consider our references and added specific visibility for the linked variable(s).

CER-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	CERC20.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at.

Alleviation:

The Saddle development team opted to consider our references and locked the compiler to version `0.5.17`.

CER-02: Check Against Zero Address

Type	Severity	Location
Volatile Code	Informational	CERC20.sol L13

Description:

The `getUnderlyingBalances` function should check whether the parameter `account` is the zero address.

Recommendation:

We advise the team to add the following `require` statement at the beginning of the function:

```
require(account != address(0), "Error Message");
```

Alleviation:

The Saddle development team opted to consider our references and added our recommended `require` statement.

LPT-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	LPToken.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at.

Alleviation:

The Saddle development team opted to consider our references and locked the compiler to version `0.5.17`.

LPT-02: Inexistent Input Sanitization

Type	Severity	Location
Volatile Code	Minor	LPToken.sol L12

Description:

The `mint` function should check whether the parameter `amount` is the zero.

Recommendation:

We advise the team to add the following `require` statements at the beginning of the function:

```
require(amount != 0, "Error Message");
```

Alleviation:

The Saddle development team opted to consider our references and added our recommended `require` statement.

MUS-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	MathUtils.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at.

Alleviation:

The Saddle development team opted to consider our references and locked the compiler to version `0.5.17`.

MUS-02: Function Optimization

Type	Severity	Location
Gas Optimization	Informational	MathUtils.sol L8-L19

Description:

The function `within1` can be further optimized.

Recommendation:

We advise the team to change the function `within1` to:

```
function within1(uint a, uint b) external pure returns (bool) {
    if (a > b) {
        return (a.sub(b) <= 1);
    }
    return (b.sub(a) <= 1);
}
```

Alleviation:

The Saddle development team opted to consider our references and optimized the `within1` function according to our recommendation.

OPE-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	OwnerPausable.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at.

Alleviation:

The Saddle development team opted to consider our references and locked the compiler to version `0.5.17`.

OPE-02: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	Informational	OwnerPausable.sol L16

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

The Saddle development team opted to consider our references and added specific visibility for the linked variable(s).

STW-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	StakeableTokenWrapper.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at.

Alleviation:

The Saddle development team opted to consider our references and locked the compiler to version `0.5.17`.

STW-02: State Variables Optimization

Type	Severity	Location
Gas Optimization	Informational	StakeableTokenWrapper.sol L18-L21

Description:

The state variable layout can be optimized to strive for tight packing.

Recommendation:

We advise the team to change the state variable layout to the following:

```
uint256 private _totalSupply;
IERC20 public stakedToken;
mapping(address => uint256) private _balances;
```

Alleviation:

The Saddle development team opted to consider our references and changed the layout of the state variables.

STW-03: Introduction of an `immutable` Variable

Type	Severity	Location
Language Specific	Informational	StakeableTokenWrapper.sol L18

Description:

The `stakedToken` state variable is arbitrarily assigned a value in the constructor of the contract and is never updated.

Recommendation:

We advise the team to change the mutability of the linked variable to `immutable`.

Alleviation:

The Saddle development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase, as the compiler version is locked at `0.5.17`, making this exhibit unavailable.

STW-04: User-Defined Getters

Type	Severity	Location
Gas Optimization	Informational	StakeableTokenWrapper.sol L20

Description:

The linked variables contain user-defined getter functions that are equivalent to their name barring for an underscore (`_`) prefix / suffix.

Recommendation:

We advise that the linked variables are instead declared as `public` and that they are renamed to their respective getter's name as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation:

The Saddle development team opted to consider our references, changed the linked variable's visibility to `public` and removed its manual getter function.

STW-05: Inexistent Input Sanitization

Type	Severity	Location
Volatile Code	Minor	StakeableTokenWrapper.sol L38

Description:

The `stake` function should check whether the parameter `amount` is the zero.

Recommendation:

We advise the team to add the following `require` statements at the beginning of the function:

```
require(amount != 0, "Error Message");
```

Alleviation:

The Saddle development team opted to consider our references and added our recommended `require` statement.

SWA-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Swap.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at.

Alleviation:

The Saddle development team opted to consider our references and locked the compiler to version `0.5.17`.

SWA-02: Omitted Parameter Description

Type	Severity	Location
Coding Style	Informational	Swap.sol L231, L243, L268, L280, L293

Description:

The description of the variable `deadline` is repeatedly omitted.

Recommendation:

We advise the team to add a description for the linked variable in the linked functions.

Alleviation:

The Saddle development team opted to consider our references and added descriptive documentation for the linked parameter.

SUS-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	SwapUtils.sol L1

Description:

An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers.
This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the full project can be compiled at.

Alleviation:

The Saddle development team opted to consider our references and locked the compiler to version `0.5.17`.

SUS-02: `struct` Optimization

Type	Severity	Location
Gas Optimization	Informational	SwapUtils.sol L32-L59

Description:

The `Swap` struct is not tightly packed. Every `struct` withholds the member information in 256-bit blocks. So, its members' data types should be as optimized as possible to reserve as little space possible.

Recommendation:

We advise the team to change the layout of the struct to:

```
struct Swap {
    uint256 A;
    uint256 swapFee;
    uint256 adminFee;
    uint256 defaultWithdrawFee;
    LPToken lpToken;
    IERC20[] pooledTokens;
    uint256[] tokenPrecisionMultipliers;
    uint256[] balances;
    mapping(address => uint256) depositTimestamp;
    mapping(address => uint256) withdrawFeeMultiplier;
}
```

Alleviation:

The Saddle development team opted to consider our references and optimized the `Swap` struct according to our recommendation.

SUS-03: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	Informational	SwapUtils.sol L62, L66, L69, L75, L80

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

The Saddle development team opted to consider our references and added specific visibility for the linked variable(s).

SUS-04: Inexistent Input Sanitization

Type	Severity	Location
Volatile Code	Minor	SwapUtils.sol L114, L403, L480, L503, L783, L798, L808, L817

Description:

The linked functions do not proper sanitization for their parameters against both the upper and lower bounds.

Recommendation:

We advise the team to revise the linked functions and add proper sanitization.

Alleviation:

The Saddle development team has acknowledged this exhibit but decided to not apply any remediation.

SUS-05: `internal` Over `external` Functions

Type	Severity	Location
Gas Optimization	Informational	SwapUtils.sol L103, L312, L438, L481, L503, L529, L553, L652, L693, L783, L798, L808, L817

Description:

The linked `external` functions are used by the `Swap.sol` contract to do the internal procedures, while it exposes the respective `public` functions.

Recommendation:

We advise the team to change the visibility specifier of the linked functions to `internal`.

Alleviation:

The Saddle development team has acknowledged this exhibit but decided to not apply any remediation, as per the following comment.

SUS-06: Function Optimization

Type	Severity	Location
Gas Optimization	Informational	SwapUtils.sol L463

Description:

The `calculateCurrentWithdrawFee` function can be further optimized.

Recommendation:

We advise the team to change the `calculateCurrentWithdrawFee` function to:

```
function calculateCurrentWithdrawFee(Swap storage self, address user)
public view returns (uint256) {
    uint256 endTime = self.depositTimestamp[user].add(4 weeks);

    if (endTime > block.timestamp) {
    uint256 timeLeftover = endTime - block.timestamp;
    return self.defaultWithdrawFee
    .mul(self.withdrawFeeMultiplier[user])
    .mul(timeLeftover)
    .div(4 weeks)
    .div(FEE_DENOMINATOR);
    }
}
```

Alleviation:

The Saddle development team opted to consider our references and optimized the `calculateCurrentWithdrawFee` function according to our recommendation.

SUS-07: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	SwapUtils.sol L563, L570, L582, L607, L787

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The Saddle development team opted to consider our references and changed to inequality operations.

SUS-08: Redundant Variable Initialization

Type	Severity	Location
Gas Optimization	Informational	SwapUtils.sol L186, L202, L226, L234, L361, L365, L381, L562, L597, L699, 700

Description:

When declaring variables without an initial value, they are assigned the specific data type's default value. Hence, the initialization of `uint256` to zero is redundant.

Recommendation:

We advise the team to remove the redundant assignments to the linked variables.

Alleviation:

The Saddle development team opted to consider our references and removed the redundant initializations.

SUS-09: Inconsistent Mathematical Logic

Type	Severity	Location
Mathematical Operations	Minor	SwapUtils.sol L703-L705

Description:

The logic to calculate the token amount seems to differ from the usual formula (see L703-L705).

Recommendation:

We advise the team to revise the token amount calculation procedure.

Alleviation:

The Saddle development team has acknowledged this exhibit and commented about the calculation applied in this block.

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.