Bounce

Security Assessment

September 30th, 2020

By :

Sheraz Arshad

Email: sheraz.arshad@certik.org

PK: `5a7e6590f454960c9fc1ab386e85b62ff0bdab4fae4ffe56959598e6eedc67e7`

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	Bounce
Description	Bounce Smart Contracts implement protocol of staking with orders matching.
Platform	Ethereum; Solidity
Codebase	Bounce.sol, BounseSealedBid.sol, BounceStakeSimple.sol

Audit Summary

Delivery Date	Sep. 30, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	1
Timeline	Aug. 27, 2020 - Sep. 30 2020

Vulnerability Summary

Total Issues	49
Total Critical	0
Total Major	3
Total Minor	0
Total Informational	46

Findings

ID	Title	Type	Severity
BSB-01	Incorrect conditions in require statement	Incorrect code	Informational
BSB-02	Loss of Funds	Incorrect Code Implementation	Informational
BSB-03	Redundant Duplicate Mappings	Language Specific	Informational
BSB-04	Redundant Duplicate Mappings	Language Specific	Informational
BSB-05	Variable Type's Alias Usage	Language Specific	Informational
BSB-06	Confusing Variable Names	Language Specific	Informational
BSB-07	Redudant Code	Optimization	Informational
BSB-08	Return Value Not Checked	Language Specific	Informational
BSB-09	Inefficient Code	Optimization	Informational
BSB-10	Redundant Variable Initialization	Optimization	Informational

ID	Title	Type	Severity
BSB-11	Inefficient Comparison with Zero	Optimization	Informational
BSB-12	Redundant Code	Optimization	Informational
BSB-13	Incomplete Modifier Name	Language Specific	Informational
BSB-14	Incorrect Usage of `ether` Global Variable	Language Specific	Informational
BSB-15	Unlocked Compiler Version	Language Specific	Informational
BSB-16	Non-standard Code Layout	Language Specific	Informational
BSB-17	Change to `constant`	Optimization	Informational
BSB-18	Redundant `return` statement	Language Specific	Informational
BSB-19	Change to `constant`	Optimization	Informational
BSB-20	Ineffectual Variale Declaration Location	Language Specific	Informational
BBB-01	Uninitialized Variable	Incorrect code	Major
BBB-02	Redundant Duplicate Mappings	Optimization	Informational
BBB-03	Storage Layout Optimization	Optimization	Informational
BBB-04	Redudant Function Declaration	Optimization	Informational
BBB-05	Redundant Array Declaration	Optimization	Informational
BBB-06	Unreachable Code Block	Incorrect code	Informational
BBB-07	Unreachable Code Block	Incorrect code	Informational
BBB-08	Unsafe Addition and Subtraction	Unsafe Operation	Informational
BBB-09	Inefficient Variable Type	Optimization	Informational
BBB-10	Redundant Variable Initialization	Optimization	Informational
BBB-11	Inefficient Comparison with Zero	Optimization	Informational

ID	Title	Type	Severity
BBB-12	Incorrect Usage of `ether` Global Variable	Language Specific	Informational
BBB-13	Unlocked Compiler Version	Language Specific	Informational
BBB-14	Non-standard Code Layout	Language Specific	Informational
BBB-15	Change to `constant`	Optimization	Informational
BSS-01	Incorrect Value Assignment	Incorrect code	Major
BSS-02	Incorrect Value Assignment	Incorrect code	Major
BSS-03	Redundant Duplicate Mappings	Optimization	Informational
BSS-04	Redundant Duplicate Mappings	Optimization	Informational
BSS-05	Return Value not Checked	Language Specific	Informational
BSS-06	Inefficient Code	Optimization	Informational
BSS-07	Return Value not Checked	Language Specific	Informational
BSS-08	Inefficient Storage Access	Optimization	Informational
BSS-09	Redundant Variable Assignment	Optimization	Informational
BSS-10	Inefficient Comparison with Zero	Optimization	Informational
BSS-11	Inefficient Variable Type	Optimization	Informational
BSS-12	Unlocked Compiler Version	Language Specific	Informational
BSS-13	Non-standard Code Layout	Language Specific	Informational
BSS-14	Change to `constant`	Optimization	Informational

BSB-01: Incorrect conditions in require statement

Type	Severity	Location
Incorrect Code	Informational	BounceSealedBid.sol: L411, L416

Description:

Judging from the error messages and the modifiers' names, it seems like the conditions in require statments are incorrect.

Recommendation:

The condition in require statements should be swaped in both modifiers to satisfy the naming of modifiers and error messages of require statements.

We advise following changes for the code.

```
modifier isPoolClosed(uint index) {
    require(closeAtP[index] > now, "this pool is not closed");
    _;
}

modifier isPoolNotClosed(uint index) {
    require(closeAtP[index] <= now, "this pool is closed");
    _;
}
```

Alleviation:

No alleviations.

BSB-02: Incorrect Code Implementation

Type	Severity	Location
Loss of Funds	Informational	BounceSealedBid.sol

Description:

If there is a larger tokens amount filled by bidders than is avalable in the pool then the lower priced bids that are not part of tokens amount filling for pool, will have their Ether stuck in the contract. Neither bidders will be able to claim these Ether nor does the contract has any function to withdraw, rendering them stuck in the contract.

Recommendation:

We advise that the code be re-structured by adding a functionality where the bidders are able to withdraw Ether if their bids were not part of tokens filling of the pool. If returning Ether back to bidders is not an option then a function should be introduced, which a designated address can call to withdraw Ether from the contract.

Alleviation:

The case was a situational and no alleviations were applied.

BSB-03: Redundant Duplicate Mappings

Type	Severity	Location
Language Specific	Informational	BounceSealedBid.sol: L426 - L50

Description:

There are mapping declarations from `L26-L50` that have the same key type of `uint` reperesenting a unique Pool Id. Having different mappings for each value type reults in increase of lookup gas costs. Moreover, the names of all mappings are not elaborative and should be self-explanatory.

Recommendation:

All the `mapping` declarations from line `L26-L50` can be grouped into a single `mapping` declaration that points to a `struct` containing the variable types of all mappings in a single structure. We advise this pattern is followed to reduce the lookup cost of the values as well as the gas cost of interacting with them.

Names of the variables should be changed to reflect their purpose f.e `amountTotal0P` should reflect in its name that it represents a token amount and similarly `amountMin1P` should reflect that it represents ETH amount.

we can have the following names for the variables.

```
creatorP -> creator
nameP -> name
token0P -> tokenAddress
amountTotal0P -> totalTokenAmountToSell
amountMin1P -> minETHAmountForSwap
amountFilled0P -> tokenAmountFilled
amountFilled1P -> ETHAmountFilled
passwordP -> password
maxEthPerWalletP -> maxEthPerWallet
closeAtP -> closeAt
```

```
creatorClaimedP -> claimed
bidderListP -> bidderList
bidderPositionListP -> bidderPositionList
bidderListHeaderP -> bidderListHeader
bidCountP -> bidCount
minEthPerWalletP -> minEthPerWallet
```

Create a new struct `Pool` which has members comprised of value types of the mappings from line `L20-L38` and define a single mapping of `pools` with `uint64` key type and `Pool` value type.

```
struct Pool {
    address payable;
    string name;
    address tokenAddress;
    uint256 totalTokenAmountToSell;
    uint256 minETHAmountForSwap;
    uint256 tokenAmountFilled;
    uint256 ETHAmountFilled;
    uint256 closeAt;
    bool claimed;
    address[] bidderList;
    uint[] bidderPositionList;
    uint bidderListHeader;
    uint minEthPerWalletP;
}

mapping(uint => Pool) public pools;
```

Alleviation:

Alleviations were applied as advised.

BSB-04: Redundant Duplicate Mappings

Type	Severity	Location
Language Specific	Informational	BounceSealedBid.sol: L55-L65

Description:

There are mapping declerations from `L57-L63` that has `address` type as key representing a `bidder`. Similar to the last issue, it results in increased gas costs for looksups. Moreover, the names of all mappings are not elaborative and should be self-explanatory.

Recommendation:

All the `mapping` declarations from line `L55-L65` can be represented by two structs, one for the `Participant` and the other one for the `Bid`. A single top-level `mapping` declaration that points to a `struct` of type `Participant` will be used to replace all mappings from `L55-65`.

Create a new struct `Bid` which has members comprised of value types of the mappings from line `L57-L63` and define another struct `Participant` which has the `Bid` struct type as its member and also contain the rest of mappings from `L55-L65` that are not part of the `Bid` struct.

The names of the struct members have been changed in the fix to have more verbosity.

```
struct Bid {
    mapping(uint => uint) tokenAmount;
    mapping(uint => uint) ETHAmount;
    mapping(uint => uint) price;
    mapping(uint => bool) claimed;
}

struct Participant {
    Bid bids;
    uint[] myBidP;
}

mapping(address => Participant) public participants;
```

Alleviation:

Alleviations were applied as advised.

BSB-05: Variable Type's Alias Usage

Type	Severity	Location
Language Specific	Informational	BounceSealedBid.sol

Description:

The contract is using `uint` to declare 256-bit unsigned integers. Although, `uint` is an alias for `uint256` and both represent the same underlying integer allocation. It is advisable that for clean coding practices the complete form `uint256` should be used instead of the alias `uint`.

Recommendation:

We recommend to use `uint256` instead of the alias `uint` to comply with the standard practice of declaring 256-bit unsigned integers.

Alleviation:

No alleviations.

BSB-06: Confusing Variable Names

Type	Severity	Location
Language Specific	Informational	BounceSealedBid.sol: L74-L93

Description:

The events have `amount0` and `amount1` variables whose names do not describe what they are supposed to represent.

Recommendation:

The names should be changed to self-explainatory names as is disccussed earlier in the report.

Alleviation:

No alleviations.

BSB-07: Redudant Code

Type	Severity	Location
Optimization	Informational	BounceSealedBid.sol: L95 & L99

Description:

The `initialize` function internally calls `initial_V1_5_0` and only has an additonal `initializer` modifier. The two functions can be merged into one and redudancy can be avoided.

Recommendation:

We recommend to merge the functions and have `initial_V1_5_0` function's body and `initialize` function's modifier both implemented in a single function.

Alleviation:

The exhibit is longer applicable as the concerned code was removed.

BSB-08: Return Value Not Checked

Type	Severity	Location
Language Specific	Informational	BounceSealedBid.sol: L137-L141

Description:

The success state of `transferFrom` and `approve` is not checked by asserting against their return values. Additionaly, the structue of `create` functon is vulnerable to re-entrancy attacks from malicious users of ERC-20 tokens. While this issue will not lead to compromising of pool creation, it will lead to gas exhaustion and should generally be avoided.

Reference the Check Effects Interactions pattern:
https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html

Recommendation:

We advise to check the success state of `transferFrom` and `approve` functions by asserting against a `bool` value they return and restructuring the function body to comply with `checks-effects-interactions` pattern.

We advise following changes for the code.

```
...
IERC20  _token0 = IERC20(token0);
// transfer amount of token0 to this contract
require(
    _token0.transferFrom(
        creator,
        address(this),
        amountTotal0
    )
);
// reset allowance to 0
require(
    _token0.approve(
        address(this),
        0
    )
);
```

```
emit Created(creator, name, token0, amountTotal0, amountMin1, closeAt);
```

Alleviation:

Alleviations were applied as advised.

BSB-09: Inefficient Code

Type	Severity	Location
Optimization	Informational	BounceSealedBid.sol: L191

Description:

The `minPrice` is calculated based on `amountMin1P` and `amountTotal0P` for a particular bool and both of these values do not seem to change throughout the lifecylce of a pool. During each execution of the `bid` function, the deterministic `minPrice` value is recalculated every time resulting in unecessary gas overheads.

Recommendation:

We advise to introduce `minPrice` as member of `Pool` struct and then directly use the value in `bid` function instead of re-calculating the value during each exection.

Alleviation:

No alleviations.

BSB-10: Redundant Variable Initialization

Type	Severity	Location
Optimization	Informational	BounceSealedBid.sol: L321, L322, L351, L352 & L391

Description:

The aforementioned lines assign the value `0` to the `uint256` contract variable.

Recommendation:

As Solidity assigns a default value to all declared variables without an assignment and the default value of a `uint256` is 0, this assignment is redundant.

Alleviation:

Alleviations were applied as advised.

BSB-11: Inefficient Comparison with Zero

Type	Severity	Location
Optimization	Informational	BounceSealedBid.sol: L234, L238, L242, 264, L268, L326, L422, L435 & L437

Description:

The aforementioned lines perform inefficient comparison with zero because when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendation:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

Alleviations were applied as advised.

BSB-11: Redundant Code

Type	Severity	Location
Optimization	Informational	BounceSealedBid.sol: L280

Description:

`if-else` blocks are redundant.

Recommendation:

We advice to remove the `if-else` blocks and simply return the predicate.

We advise following changes for the code.

```
function isCreator(address target, uint64 index) private view returns (bool) {
    return creatorP[index] == target;
}
```

Alleviation:

No alleviations.

BSB-13: Incorrect conditions in require statement

Type	Severity	Location
Incomplete Modifier Name	Informational	BounceSealedBid.sol: L404

Description:

The modifier `checkBotHolder` has a check for `password` in addition to the check for bot holder and the password check is not indicated in the name of the modifier.

Recommendation:

We advise to either change the name of modifier to reflect the password check that it conducts or othewise introduce a separate modifier for password check.

Alleviation:

The exhibit intends to aid code readability as such not mandatory. No alleviations were applied.

BSB-14: Incorrect Usage of ether Global Variable

Type	Severity	Location
Language Specific	Informational	BounceSealedBid.sol: L406

Description:

The `ether` global variable is used as a decimals multiplier for a standard ERC20 token.

Recommendation:

We recommend that actual decimals multiplier for the ERC-20 token be used instead of the `ether` global variable.

Alleviation:

The exhibit is longer applicable as the concerned code was removed.

BSB-15: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	BounceSealedBid.sol: L3

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.

We advise following changes for the code.

```
pragma solidity 0.6.0;
```

Alleviation:

No alleviations.

BSB-16: Non-standard Code Layout

Type	Severity	Location
Language Specific	Informational	BounceSealedBid.sol

Description:

The structure of the codebase does not conform to the official Solidity style guide of v0.6.0.

Recommendation:

An indicative excerpt of the style guide is that functions should be grouped according to their visibility and ordered:

`constructor`

`receive function` (if exists)

`fallback function` (if exists)

`external`

`public`

`internal`

`private`

Additionally, the internal layout of a contract should be as follows:

`Type declarations`

`State variables`

`Events`

`Functions`

Alleviation:

No alleviations.

BSB-17: Change to constant

Type	Severity	Location
Optimization	Informational	BounceSealedBid.sol: L101-L105

Description:

The variables initialized from `L104-L106` are never assigned again in the contract and hence can be declared as constants to save gas costs.

Recommendation:

We recommend to declare the variables on aforementioned lines as constants which will be cheaper to use as constant variables do not occupy storage slot and are stored in the code of the deployed contract.

Alleviation:

Alleviations were applied as advised.

BSB-18: Redundant return statement

Type	Severity	Location
Language Specific	Informational	BounceSealedBid.sol: L387

Description:

On `L387` the variable `r`'s default value is returned which is always zero. A well crafted function should never return any of its named return variables.

Recommendation:

We advise either to return integer literal `0` or wrap the code after the first `if` statement in `else` block, so a default value of `r` is returned.

Alleviation:

No alleviations.

BSB-19: Change to constant

Type	Severity	Location
Optimization	Informational	BounceSealedBid.sol: L292, L303, L341 & L377

Description:

The aforementioned lines use `type(uint).max`. As it is a constant value representing a maximum value a `uint256` can hold, it can be initialized as constant in the contract to enhance the readability of the code.

Recommendation:

We recommend to declare a constant for `type(uint256).max` and it be used in the code for readability.

We advise following changes for the code.

```
uint256 constant MAX_UINT256 = type(uint256).max;
```

Alleviation:

Alleviations were applied as advised.

BSB-20: Ineffectual Variale Declaration Location

Type	Severity	Location
Language Specific	Informational	BounceSealedBid.sol: L295

Description:

The `L295` declares uint256 `curPosition` but it is only used in a child block's scope.

Recommendation:

We recommend to move the declaration of `curPosition` to `L301` outside the `while` loop.

We advise following changes for the code.

```
uint256 curPosition;
while (true) {
...
```

Alleviation:

No alleviations.

BBB-01: Uninitialized Variable

Type	Severity	Location
Incorrect code	Major	Bounce.sol: L394

Description:

The storage initialDateIndex is never initialized in the contract and its value remains 0 throughout contract's lifecycle.
Because of it, the predicate of if clause on L394 always evaluates to false, which results in totalBonus function returning 0 everytime it executes.

Recommendation:

We advise to assign initialDateIndex storage variable with a correct value.

Alleviation:

Alleviations were applied as advised.

BBB-02: Redundant Duplicate Mappings

Type	Severity	Location
Optimization	Informational	Bounce.sol: L22-L30

Description:

There are array deleclarations from L22-L30 that represent a pool.
Each of these arrays represents a property of pool. This approach can result in increased gas cost when we have to access storage repeatedly for accessing properties of pool.

Recommendation:

Our recommendation is to create a struct for pool and declare each of these arrays as properties of the Pool struct. An array of Pool would replace all of these individual array declarations.

We advise following changes for the code.

```
struct Pool {
    address payable[] creatorFP;
    string[] nameFP;
    address[] token0FP;
    address[] token1FP;
    uint256[] amountTotal0FP;
    uint256[] amountTotal1FP;
    uint256[] amountSwap0FP;
    uint256[] amountSwap1FP;
    uint256[] closeAtFP;

}

Pool[] publc pools;
```

Alleviation:

Alleviations were applied as advised.

BBB-03: Storage Layout Optimization

Type	Severity	Location
Optimization	Informational	Bounce.sol: L35

Description:

`initialDateIndex` is a `uint32` and it can placed in storage of the contract alongside another less-than-32-byte variable where both of the variables could be packed inside a single storage slot.

Recommendation:

We advise to move `initialDateIndex` at the top of storage layout, so `bonusToken` and `initialDateIndex` could be packed inside a single 32-byte slot.

We advise following changes for the code.

```
// bonus storage
uint32 public initialDateIndex;
address public bonusToken;
```

Alleviation:

Alleviations were applied as advised.

BBB-04: Redudant Function Declaration

Type	Severity	Location
Optimization	Informational	Bounce.sol: L90, L93

Description:

The body of function `initialize` is empty and the function `initialV1_5_0` contains the body for contract initialzation.

Recommendation:

Our recommendation is to merge the both functions with a single function having `initializer` modifier from initialize function and the body from `initialV1_5_0` function.

Alleviation:

This exhibit is longer applicable as the concerned code was removed.

BBB-05: Redundant Array Declaration

Type	Severity	Location
Optimization	Informational	Bounce.sol: L25

Description:

The array of address `token1FP` represents ETH addresses and its indexes are always assigned with `address(0)` which is also a default value of the `address` type. This makes the existence of array `token1FP` redundant.

Recommendation:

We advise to remove this array from storage and instead rely on any other pool's property's non-zero value to confirm the existance of pool.

Alleviation:

No alleviations.

BBB-06: Unreachable Code Block

Type	Severity	Location
Incorrect code	Informational	Bounce.sol: L279

Description:

On line 279, `token1FP[index]` will always be `address(0)` resulting in `else` block unreachable.

Recommendation:

We advise to move the `require` statement from `if` clause and place it at the start of the function. The `else` block is unreachable and `if` block is ineffectual, so the whole `if-else` block can be removed.

Alleviation:

No alleviations.

BBB-07: Unreachable Code Block

Type	Severity	Location
Incorrect code	Informational	Bounce.sol: L308

Description:

On line 308, `token1FP[index]` will always be `address(0)` resulting in `else` block unreachable.

Recommendation:

We advise to move the `sender.transfer(excessAmount1)` statement outisde the `if` clause and remove the whole `if-else` block.

Alleviation:

No alleviations.

BBB-08: Unsafe Addition and Subtraction

Type	Severity	Location
Unsafe Operation	Informational	Bounce.sol: L359, L414

Description:

The lines `L359` and `L414` perform unsafe addition and subtraction, respectively.

Recommendation:

Although, the probability of overflow in both of the cases is very low but we still advise that the SafeMath library is utilized regardless to ensure consistency in the project's codebase and account for all types of edge cases.

We advise following changes for the code.

```
// L359
uint256 amount = amountTotal0FP[index].sub(amountSwap0FP[index]);
// L414
return currentDateIndex().sub(1);
```

Alleviation:

Alleviations were applied as advised.

BBB-09: Inefficient Variable Type

Type	Severity	Location
Optimization	Informational	Bounce.sol: L409, L413

Description:

The `uint32` type is being used to represent number days.

Recommendation:

Although, the value can fit within `uint32` without any issues but as the EVM is geared towards 32-byte data types, it costs more gas to interact with and utilize a `uint32` variable than a `uint256`. As such, we advise that this is instead set to a functionally identical `uint256`.

Alleviation:

Alleviations were applied as advised.

BBB-10: Redundant Variable Initialization

Type	Severity	Location
Optimization	Informational	Bounce.sol: L258

Description:

The aforementioned line assigns the value `0` to the `uint256` contract variable.

Recommendation:

As Solidity assigns a default value to all declared variables without an assignment and the default value of a `uint256` is 0, this assignment is redundant.

Alleviation:

Alleviations were applied as advised.

BBB-11: Inefficient Comparison with Zero

Type	Severity	Location
Optimization	Informational	Bounce.sol: L163, L172, L174, L297, L306, L325, L328, L363, L375

Description:

The aforementioned lines perform inefficient comparison with zero because when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendation:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

Alleviations were applied as advised.

BBB-12: Incorrect Usage of ether Global Variable

Type	Severity	Location
Language Specific	Informational	Bounce.sol: L419

Description:

The `ether` global variable is used as a decimals multiplier for a standard ERC20 token.

Recommendation:

We recommend that actual decimals multiplier for the ERC-20 token be used instead of the `ether` global variable.

Alleviation:

This exhibit is not longer applicable as the concerned code was removed.

BBB-13: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Bounce.sol: L3

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.

We advise following changes for the code.

```
pragma solidity 0.6.0;
```

Alleviation:

No alleviations.

BBB-14: Non-standard Code Layout

Type	Severity	Location
Language Specific	Informational	Bounce.sol

Description:

The structure of the codebase does not conform to the official Solidity style guide of v0.6.0.

Recommendation:

An indicative excerpt of the style guide is that functions should be grouped according to their visibility and ordered:

`constructor`

`receive function` (if exists)

`fallback function` (if exists)

`external`

`public`

`internal`

`private`

Additionally, the internal layout of a contract should be as follows:

`Type declarations`

`State variables`

`Events`

`Functions`

Alleviation:

No alleviations.

BBB-15: Change to `constant`

Type	Severity	Location
Optimization	Informational	Bounce.sol: L95-L99

Description:

The variables initialized from `L95-L99` are never assigned again in the contract and hence can be declared as constants to save gas costs.

Recommendation:

We recommend to declare the variables on aforementioned lines as constants which will be cheaper to use as constant variables do not occupy storage slot and are stored in the code of the deployed contract.

Alleviation:

Alleviations were applied as advised.

BSS-01: Incorrect Value Assignment

Type	Severity	Location
Incorrect code	Major	BounceStakeSimple.sol: L88

Description:

`dailyStake[curDateIndex]` seems to be assigned an incorrect value of `totalState` instead of being incremented by `amount`.

Recommendation:

We advise to increment the `dailyStake[curDateIndex]` by `amount`.

We advise following changes for the code.

```
dailyStake[curDateIndex] = dailyStake[curDateIndex].add(amount);
```

Alleviation:

This exhibit is no longer applicable as the concerned code was removed.

BSS-02: Incorrect Value Assignment

Type	Severity	Location
Incorrect code	Major	BounceStakeSimple.sol: L119

Description:

`dailyStake[curDateIndex]` seems to be assig`dailyStake[curDateIndex]` seems to be assigned an incorrect value of `totalState` instead of being decremented by `amount`.
ned an incorrect value of `totalState` instead of being incremented by `amount`.

Recommendation:

We advise to decrement the `dailyStake[curDateIndex]` by `amount`.

We advise following changes for the code.

```
dailyStake[curDateIndex] = dailyStake[curDateIndex].sub(amount);
```

Alleviation:

This exhibit is no longer applicable as the concerned code was removed.

BSS-03: Redundant Duplicate Mappings

Type	Severity	Location
Optimization	Informational	BounceStakeSimple.sol: L28-L32

Description:

There are mapping deleclarations from `L28-L32` that represent stakes, rewards available and rewads claimed.
As all of these mappings have a common key i.e. a number representing a day, these mappings can be replaced by a single mapping with a struct value type.

Recommendation:

Our recommendation is to create a struct and then replace the all three mappings with a single mapping with the struct as value type. The struct would have stakes, rewards available and rewads claimed as its members.

Alleviation:

Alleviations were applied as advised.

BSS-04: Redundant Duplicate Mappings

Type	Severity	Location
Optimization	Informational	BounceStakeSimple.sol: L35-L46

Description:

There are mapping deleclarations from `L35-L46` that represent a user participant of the contract.
As all of these mappings have a common key i.e. an address representing a user, these mappings can be replaced by a single mapping with a struct value type.

Recommendation:

Our recommendation is to create a struct of `User` and then replace the all three mappings with a single mapping with the struct as value type. The struct would have members comprising of values types of all of the mappings involved.

We advise following changes for the code.

```
struct User {
    uint256 myTotalStake;
    mapping(uint32 => uint256) myDailyStake;
    mapping(uint32 => bool)  myRewardClaimed;
    mapping(uint32 => uint256) myUnStake;
    mapping(uint32 => uint32) myUnStakeEndAt;
    uint32[] myUnStakes;
}

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

Alleviation:

Alleviations were applied as advised.

BSS-05: Return Value not Checked

Type	Severity	Location
Language Specific	Informational	BounceStakeSimple.sol: L74-L76

Description:

The success state of `transferFrom` and `approve` is not checked by asserting against their return values. Additionaly, the structue of `staking` functon is vulnerable to re-entrancy attacks from malicious users of ERC-20 tokens. While this issue will not lead to compromising of staking , it will lead to gas exhaustion and should generally be avoided.

Reference the Check Effects Interactions pattern:
https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html

Recommendation:

We advise to check the success state of `transferFrom` and `approve` functions by asserting against a `bool` value they return and restructuring the function body to comply with `checks-effects-interactions` pattern.

We advise following changes for the code.

```
...
IERC20  _token0 = IERC20(token0);
// transfer amount of token0 to this contract
require(
     _stakeToken.transferFrom(sender, address(this), amount)
);
// reset allowance to 0
require(
     _stakeToken.approve(address(this), 0)
);
```

Alleviation:

Alleviations were applied as advised.

BSS-06: Inefficient Code

Type	Severity	Location
Optimization	Informational	BounceStakeSimple.sol: L217

Description:

The predicate of `if` clause `index < array.length - 1` can be converted to `index != array.length - 1` which will consume slightly less gas.

Recommendation:

We advise to change the predicate `index < array.length - 1` to `index != array.length - 1`

Alleviation:

Alleviations were applied as advised.

BSS-07: Return Value not Checked

Type	Severity	Location
Language Specific	Informational	BounceStakeSimple.sol: L148

Description:

The success state of `transfer` call is not checked by asserting against its return value.

Recommendation:

We advise to check the success state of `transfer` call.

We advise following changes for the code.

```
require(
     IERC20(StakeToken).transfer(sender, amount)
);
```

Alleviation:

Alleviations were applied as advised.

BSS-08: Inefficient Storage Access

Type	Severity	Location
Optimization	Informational	BounceStakeSimple.sol: L144, L214

Description:

We are reading from storage on each iteration of the array in function`removeArray` and it significantly increases the gas cost of the operation.

Recommendation:

We advice to change the data location of parameter `array` of function `removeArray` from `storage` to `memory`. This change will result in receiving the copy of array in `memory` which will be cheaper to perform operations on.

We advise following changes for the code.

```
// change the function signature to point array to memory
function removeArray(uint32[] memory array, uint32 index) private returns
(uint32[] memory);
```

Alleviation:

Alleviations were applied as advised.

BSS-09: Redundant Variable Assignment

Type	Severity	Location
Optimization	Informational	BounceStakeSimple.sol: L105, L129-L131, L142, L191-L192, L203-L204

Description:

The aforementioned line assigns the value `0` to the `uint256` contract variable.

Recommendation:

As Solidity assigns a default value to all declared variables without an assignment and the default value of a `uint256` is 0, this assignment is redundant.

Alleviation:

Alleviations were applied as advised.

BSS-10: Inefficient Comparison with Zero

Type	Severity	Location
Optimization	Informational	BounceStakeSimple.sol: L61, L70, L96, L106, L147, L168, L174

Description:

The aforementioned lines perform inefficient comparison with zero because when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendation:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

Alleviations were applied as advised.

BSS-11: Inefficient Variable Type

Type	Severity	Location
Optimization	Informational	BounceStakeSimple.sol: L44

Description:

The value type of the mapping `myUnStakeEndAt`, which is `uint32`, is not packed by the EVM. And as EVM works with 32-byte values, it costs more gas to utilize `uint32` types.

Recommendation:

We advice to change the value type of mapping `myUnStakeEndAt` from `uint32` to its functionally identical `uint256`.

Alleviation:

Alleviations were applied as advised.

BSS-12: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	BounceStakeSimple.sol: L3

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.

We advise following changes for the code.

```
pragma solidity 0.6.0;
```

Alleviation:

No alleviations.

BSS-13: Non-standard Code Layout

Type	Severity	Location
Language Specific	Informational	BounceStakeSimple.sol

Description:

The structure of the codebase does not conform to the official Solidity style guide of v0.6.0.

Recommendation:

An indicative excerpt of the style guide is that functions should be grouped according to their visibility and ordered:

`constructor`
`receive function` (if exists)
`fallback function` (if exists)
`external`
`public`
`internal`
`private`

Additionally, the internal layout of a contract should be as follows:

`Type declarations`
`State variables`
`Events`
`Functions`

Alleviation:

No alleviations.

BSS-14: Change to constant

Type	Severity	Location
Incorrect code	Informational	BounceStakeSimple.sol: L88

Description:

The variables initialized from `L54-L56` are never assigned again in the contract and hence can be declared as constants to save gas costs.

Recommendation:

We recommend to declare the variables on aforementioned lines as constants which will be cheaper to use as constant variables do not occupy storage slot and are stored in the code of the deployed contract.

Alleviation:

This exhibit is no longer applicable as the concerned code was marked as deprecated.