

# BRCStarter Staking and Launchpad

smart contracts  
final audit report

---

January 2024



[hashex.org](https://hashex.org)



[contact@hashex.org](mailto:contact@hashex.org)

# Contents

1. Disclaimer	3
2. Overview	4
3. Project centralization risks	6
4. Found issues	7
5. Contracts	9
6. Conclusion	15
Appendix A. Issues' severity classification	16
Appendix B. Issue status description	17
Appendix C. List of examined issue types	18
Appendix D. Centralization risks classification	19

# 1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below - please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author ([hashex.org](https://hashex.org)).

## 2. Overview

HashEx was commissioned by the BRCStarter team to perform an audit of their smart contract. The audit was conducted between 22/12/2023 and 25/12/2023.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available in the @BRCStarter/Contract GitHub repository after the commit [dd1c2bc](#).

**Update.** The BRCStarter team has responded to this report. The updated code is located in the same repository after the commit [03d5523](#).

## 2.1 Summary

Project name	BRCStarter Staking and Launchpad
URL	<a href="https://github.com/BRCStarter/">https://github.com/BRCStarter/</a>
Platform	Binance Smart Chain
Language	Solidity
Centralization level	● High
Centralization risk	● High

## 2.2 Contracts

Name	Address
Staking	
LaunchPadTemplate	

### 3. Project centralization risks

The contracts are ownable, their governance functions can be maliciously or mistakenly used to make some public functions unusable.

#### C7dCR09 Owner can update tier parameters

---

The `setTiers()` function allows the owner to adjust tier list parameters in terms of required locking period and amount. Selecting unrealistic requirements may lead to all users being pushed to 0 tier. Also, user's tier may be changed by the owner at any moment, if it's used anywhere out of the scope of this audit.

#### C7eCR0a Users donate their funds

---

The LaunchPadTemplate contract only collects users' ERC20 tokens in exchange of nothing. Users can only trust the owner if they are promised anything beyond that.

## 4. Found issues











Medium	2 (18%)
Low	6 (55%)
Info	3 (27%)

### C7d. Staking

ID	Severity	Title	Status
C7dl65	Medium	ERC-4337 accounts are not supported	Resolved
C7dl63	Low	Lack of documentation	Acknowledged
C7dl64	Low	Gas optimizations	Partially fixed
C7dl62	Info	Typographical error	Partially fixed

### C7e. LaunchPadTemplate

ID	Severity	Title	Status
C7el6c	Medium	Wrong result of round2Allowance() function	Resolved
C7el68	Low	Gas optimizations	Partially fixed
C7el69	Low	Confusing error messages	Acknowledged

C7el6a	 Low	Lack of input validation	 Resolved
C7el6b	 Low	Transfers of an arbitrary ERC20	 Acknowledged
C7el66	 Info	Naming conventions violation	 Resolved
C7el67	 Info	Default visibility of state variables	 Resolved



## 5. Contracts

### C7d. Staking

#### Overview

Simple staking contract for a single ERC20 token. First stake must satisfy the minimum amount requirement, which can be adjusted by the contract owner, subsequent stakes may be performed with an arbitrary amount. Unstaking is available at any moment without additional conditions but can't be performed partially or selectively. The contract has public getter `getUserStakingData()`, calculating the user's staking tier and subjected amount, which is calculated with the following rules:

1. Time-sorted deposits are iterated from past to future to find deposit, that meets the requirements for both locking period and locked amount.
2. Each deposit is inspected for tier list from highest to lowest, if the tier requirements are met, the user tier index is updated (but only upward), and the sum value is increased by the amount of the found deposit.
3. Every deposit after the first found is checked with the simplified requirement of `depositAmount + previouslyFoundSum >= tierAmountRequirement`, locking period requirement is remaining the same as before.
4. Finally, the maximum found tier index and sum of eligible deposits are returned.

#### Issues

##### C7dl65 ERC-4337 accounts are not supported

Medium

Resolved

The [ERC-4337](#) accounts can't be used with Staking contract since it strictly requires `msg.sender == tx.origin`. Users with ERC-4337 wallets will not be able to interact with Staking.

```
function staking(uint256 _amount) external {
    require(tx.origin == msg.sender, "Contracts not allowed.");
    ...
}
```

```
}
```

## Recommendation

Consider removing `tx.origin` requirement.

### C7dl63 Lack of documentation

● Low

✓ Acknowledged

The smart contract lacks NatSpec documentation. NatSpec is a natural language specification format used in Solidity to write comments that describe the purpose and behavior of functions, contracts, and other code structures. Proper documentation improves code readability, aids in understanding the contract's intended functionality, and provides guidance for future developers or auditors.

In addition, ambiguous logic of the `getUserStakingData()` function needs to be confirmed in the documentation: two tier4 deposits (by timestamps) are not stacked if both of them below minimum in terms of amount (but their sum satisfies the requirements).

### C7dl64 Gas optimizations

● Low

🔧 Partially fixed

1. Multiple reads from storage in the `getUserStakingData()` function: `deposits[_user].length` is read as `for()` loop condition.
2. Excessive read and unnecessary actions in the `getUserStakingData()` function: `x > y || x+a > y`.

### C7dl62 Typographical error

● Info

🔧 Partially fixed

The terms "statut", "elligible", "meens" are used in the contract, which is presumably a typographical error. The correct terms should be "status", "eligible", "means". Misnaming variables can lead to confusion for developers, maintainers, and auditors, potentially obscuring the intent and functionality of the code.

## C7e. LaunchPadTemplate

### Overview

Simple 2-round sale, payments in form of fixed ERC20 token can be made according to the allowances, governed by the contract owner. The only output for the user is the `claimable()` and `claimableAmount()`, two separate getters for the same stored data.

### Issues

#### C7e16c Wrong result of round2Allowance() function ● Medium ✓ Resolved

Input parameter `_address` is ignored in the `round2Allowance()` function if the checked user has been already participated, i.e., function returns allowance for `msg.sender` instead of the one for querying account. This problem will specifically arise in chain explorer, displaying zero allowance for every participated user. However, user interaction both purchase functions will remain intact.

```
function round2Allowance(address _address) public view returns (uint256 result) {
    if (_hasParticipated[_address]) {
        result = _round2Allowance[msg.sender];
    } else {
        result = round1Allowance[_address] * ROUND2_MULTIPLIER;
    }
}
```

### Recommendation

Return allowance for `_address` user.

```
function round2Allowance(address _address) public view returns (uint256 result) {
    if (_hasParticipated[_address]) {
        result = _round2Allowance[_address];
    }
    ...
}
```

```
}
```

## C7el68 Gas optimizations

● Low

🔧 Partially fixed

1. Multiple reads from storage in the `_roundNumber()` function: `round1BeganAt` variable.
2. Duplicated code in the `buyRound1Stable()` function: `require(round1Allowance[msg.sender] >= _amount)`.
3. Multiple reads from storage in the `buyRound1Stable()` function: `round1Allowance[msg.sender]`, `claimable[msg.sender]`, `stableRaised`, `stableTarget` variables.
4. Multiple reads from storage in the `buyRound2Stable()` function: `_hasParticipated[msg.sender]`, `round2Allowance[msg.sender]`, `claimable[msg.sender]`, `stableRaised`, `stableTarget` variables.

## C7el69 Confusing error messages

● Low

✅ Acknowledged

The error messages in the `buyRound1Stable()` and `buyRound2Stable()` functions may confuse users, who may think that the target is already hit, and purchase is impossible.

```
require(stableRaised + _amount <= stableTarget, "Target already hit")
```

## C7el6a Lack of input validation

● Low

✅ Resolved

The `setRound1Timestamp()` function has no safety check against current timestamp while the same parameter is checked in the constructor section.

```
constructor(IERC20 _stable, uint256 _stableTarget, uint256 _round1BeganAt)
Ownable(msg.sender){
    require(_round1BeganAt >= block.timestamp, "Timestamp already passed");
    ...
    round1BeganAt = _round1BeganAt;
```

```
}

function setRound1Timestamp(uint256 _round1BeginAt) external onlyOwner {
    round1BeganAt = _round1BeginAt;
}
```

## C7e16b Transfers of an arbitrary ERC20

● Low

✔ Acknowledged

The `emergencyWithdrawToken()` function is used to transfer out any contract holdings of ERC-20 tokens. We recommend wrapping transfers of an arbitrary tokens into the [SafeERC20](#) library transfers in order to support slightly out-of-standard tokens as well.

```
function emergencyWithdrawToken(IERC20 _token)
    external
    onlyOwner
    returns (bool)
{
    return
        _token.transfer(
            msg.sender,
            _token.balanceOf(address(this))
        );
}
```

## C7e166 Naming conventions violation

● Info

✔ Resolved

The [Solidity Naming Conventions](#) suggest constants to be named in `UPPER_CASE_WITH_UNDERSCORES` style to be easily recognizable. The `round1Duration` constant in the LaunchPadTemplate contract violates these conventions while the `ROUND2_MULTIPLIER` is properly named.

```
uint256 public constant round1Duration = 3600;
uint256 public constant ROUND2_MULTIPLIER = 10;
```

## C7e167 Default visibility of state variables

● Info

✔ Resolved

The `_round2Allowance` and `_hasParticipated` state variables in the contract have been declared without an explicit visibility specifier. In Solidity, if no visibility is specified, the default is `internal`. This means that while these variables are not directly accessible from external calls, they can be accessed and potentially modified by derived contracts. Not specifying visibility explicitly can lead to confusion about the intended accessibility of these variables and may inadvertently expose them to unintended modifications in future contract iterations.

## 6. Conclusion

2 medium, 6 low severity issues were found during the audit. 2 medium, 1 low issues were resolved in the update. The reviewed contracts are highly dependent on the owner's account. See the centralization risks chapter.

This audit includes recommendations on code improvement and the prevention of potential attacks.

## Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.



## Appendix B. Issue status description

- ✔ **Resolved.** The issue has been completely fixed.
- 🔧 **Partially fixed.** Parts of the issue have been fixed but the issue is not completely resolved.
- 🕒 **Acknowledged.** The team has been notified of the issue, no action has been taken.
- ❓ **Open.** The issue remains unresolved.

## Appendix C. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

## Appendix D. Centralization risks classification

### Centralization level

- **High.** The project owners can manipulate user's funds, lock user's funds on their will (reversible or irreversible), or maliciously update contracts parameters or bytecode.
- **Medium.** The project owners can modify contract's parameters to break some functions of the project contract or contracts, but user's funds remain withdrawable.
- **Low.** The contract is trustless or its governance functions are safe against a malicious owner.

### Centralization risk

- **High.** Lost ownership over the project contract or contracts may result in user's losses. Contract's ownership belongs to EOA or EOAs, and their security model is unknown or out of scope.
- **Medium.** Contract's ownership is transferred to a contract with not industry-accepted parameters, or to a contract without an audit. Also includes EOA with a documented security model, which is out of scope.
- **Low.** Contract's ownership is transferred to a well-known or audited contract with industry-accepted parameters.

 [contact@hashex.org](mailto:contact@hashex.org)

 [@hashex\\_manager](https://t.me/hashex_manager)

 [blog.hashex.org](https://blog.hashex.org)

 [linkedin](https://www.linkedin.com/company/hashex)

 [github](https://github.com/hashex)

 [twitter](https://twitter.com/hashex)

**#HashEx**  
BLOCKCHAIN SECURITY