#HashEx
BLOCKCHAIN SECURITY

Brighthub

smart contracts final audit report

February 2025

hashex.org

contact@hashex.org

Contents

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).

2. Overview

HashEx was commissioned by the Brighthub team to perform an audit of their smart contract. The audit was conducted between 18/02/2025 and 20/02/2025.

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 at 0x6E4042ded23e78C725aA3004a8F87e4c5F29d6BA in the BNB Smart Chain.

2.1 Summary

Project name	Brighthub
URL	https://bhf.lol
Platform	Binance Smart Chain
Language	Solidity
Centralization level	High
Centralization risk	High

2.2 Contracts

Name	Address
RechargeManager	0x6E4042ded23e78C725aA3004a8F87e4c5F29d6BA

3. Project centralization risks

C99CR3c Owner privileges

The contract owner can add new recipients and transfer ownership to another account.

4. Found issues

High	1 (11%)
Low	5 (56%)
Info	3 (33%)

C99. RechargeManager

ID	Severity	Title	Status
C99Iec	High	Possible frontrun	Open
C99If2	Low	No mechanism to remove an invalid recipient	Open
C99If4	Low	Missing zero address check in transferOwnership function	Open
C99Ied	Low	Lack of events for state changes	Open
C99If0	Low	Lack of slippage protection in swaps	Open
C99If1	Low	Gas optimizations	Open
C99Iee	Info	Non descriptive error message	Open
C99If3	Info	Return values not checked in token operations	Open
C99Ief	Info	Unnecessary deadline parameter	Open

5. Contracts

C99. RechargeManager

Overview

The RechargeManager contract executes a two-step token swap process. It maintains a round-robin list of recipients, which can be updated by the contract owner. In the main function, `doRecharge()`, a user sends USDT tokens to the contract. These tokens are then swapped for USDC, and the received USDC is subsequently swapped back into USDT and sent to the next recipient in the list.

Issues

C99Iec Possible frontrun

High Open

A user who sends funds to the contract using the `doRecharge()` function expects that the funds will be sent to the next recipient in line. However, if another user front-runs the transaction, the initial transaction will send the funds not to the next recipient, but to the one after that. In some cases—such as when the expected recipient is the first in the list—the contract owner may add a new address which will receive the sent funds.

Recommendation

Pass the next recipient index as a function parameter so that if the transaction is front-run, it will fail.

C99If2 No mechanism to remove an invalid recipient

Low Open

The contract provides a function to add recipients but does not include a mechanism to remove or correct an invalid recipient that may have been added by mistake. Without the ability to remove or update recipients, an incorrect or malicious address could remain in the recipient list indefinitely, potentially diverting funds from their intended destination.

C99lf4 Missing zero address check in transferOwnership function

Low Open

The `transferOwnership()` function does not verify that the new owner address is not the zero address (`address(0)`). Allowing the zero address to be set as the owner may result in the contract being effectively locked, as no one would have the proper authority to manage the contract afterward.

```
    function transferOwnership(address owner) onlyOwner external {
        _owner = owner;
    }
```

C99led Lack of events for state changes

Low Open

The contract does not emit events for several important state changes, such as when a new recipient is added or when ownership is transferred. This absence of event logging makes it difficult for external observers and developers to track these changes, complicating debugging and monitoring efforts.

C99lf0 Lack of slippage protection in swaps

Low Open

The swap operations within the `doRecharge` function do not enforce any slippage protection, as the `amountOutMinimum` parameter is set to zero for both swap calls. This absence of a minimum return threshold exposes the contract to adverse price movements and front-running attacks, potentially resulting in significantly less favorable swap outcomes than expected.

```
    function doRecharge(uint256 _amount) external {
        ...
        IPancakeV3Router.ExactInputSingleParams memory params2 =
IPancakeV3Router.ExactInputSingleParams({
            tokenIn: _usdc,
            tokenOut: _usdt,
```

```
            fee: 100,
            recipient: _recipients[_recipientIndex],
            deadline: block.timestamp + 300,
            amountIn: usdcAmount,
            amountOutMinimum: 0,
            sqrtPriceLimitX96: 0
        });
        ...
    }
```

Recommendation

Add minimum out parameter to the `doRecharge()` function.

C99lf1 Gas optimizations

Low Open

- Multiple reads from storage in the `doRecharge()` function. State variables `_usdt`, `_usdc`, `_v3Router`,`_recipients.length`, `_recipientIndex` can be memoised.

- State variables `_usdt`, `_usdc`, `_v3Router` can be declared as immutable.

- Token approvals for USDT and USDC for the router can be made only once.

C99lee Non descriptive error message

Info Open

In the `doRecharge()` function, the contract uses a vague error message ("sss") in the require statement that checks if the `recipients` array is non-empty. This non-descriptive message does not clearly indicate the issue to the user or developer, making debugging more difficult.

```
    function doRecharge(uint256 _amount) external {
        ...
        require(_recipients.length > 0, "sss");
        ...
    }
```

C99lf3 Return values not checked in token operations

Info Open

The contract makes external calls to token functions such as `transferFrom()` and `approve()` without checking their return values. This practice may lead to silent failures if these functions return false or do not behave as expected (as is the case with some tokens in certain networks, like USDT).

```
    function doRecharge(uint256 _amount) external {
        IBEP20(_usdt).transferFrom(msg.sender, address(this), _amount);
        IBEP20(_usdt).approve(_v3Router, type(uint256).max);
        ...
        IBEP20(_usdc).approve(_v3Router, usdcAmount);
        ...
    }
```

C99lef Unnecessary deadline parameter

Info Open

The contract sets the deadline parameter in the swap function calls to `block.timestamp + 300`. Since `block.timestamp` always reflects the current time at the moment a transaction is mined, adding a fixed 300 seconds does not provide meaningful protection against delayed transactions. This approach might give the false impression of a time-sensitive safeguard. The deadline parameter can be set off-chain and passed as a function parameter.

```
    function doRecharge(uint256 _amount) external {
        ...
        IPancakeV3Router.ExactInputSingleParams memory params2 =
IPancakeV3Router.ExactInputSingleParams({
            tokenIn: _usdc,
            tokenOut: _usdt,
            fee: 100,
            recipient: _recipients[_recipientIndex],
            deadline: block.timestamp + 300,
            amountIn: usdcAmount,
            amountOutMinimum: 0,
            sqrtPriceLimitX96: 0
        });
```

```
        ...
    }
```

6. Conclusion

1 high, 5 low severity issues were found during the audit. No 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

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY