BLOCKSEC

Security Audit
Report for BR contract

Date: March 07, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Bedrock-Technology
Target	BR contract

Version History

Version	Date	Description
1.0	March 07, 2025	First release

Signature

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the code repository of BR contract[1] of Bedrock-Technology. Specifically, BR contract is derived from `ERC20Burnable` and includes a freeze mechanism to curb the flow of illicit funds.

Please note that the scope of this audit is limited to the following files:

- BR.sol

The auditing process is iterative. Specifically, we will audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following. Our audit report is responsible for the only initial version (i.e., `Version 1`), as well as new codes (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
BR contract	`Version 1`	`5ad1e7679c26f1b0883a6ed5b77251eb0e360c15`
	`Version 2`	`5c190f61848a747e7894b0697867cb32b62393c8`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

[1] https://github.com/Bedrock-Technology/BR/tree/main

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Permission management
* Business logic
* Token operation
* Emergency mechanism
* Oracle security
* Whitelist and blacklist
* Economic impact
* Batch transfer

1.3.3 NFT Security

* Duplicated item
* Verification of the token receiver
* Off-chain metadata security

1.3.4 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology and Common Weakness Enumeration. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we did not find potential security issues. Besides, we have **three** recommendations and **two** notes.

- Recommendation: 3
- Note: 2

ID	Severity	Description	Category	Status
1	-	Add proper initializations in the `constructor()` of the contract `BR`	Recommendation	Fixed
2	-	Check `users.length` in the `freezeUsers()` and `unfreezeUsers()` functions	Recommendation	Fixed
3	-	Add validation checks for `_msgSender()` in the function `batchTransfer()`	Recommendation	Confirmed
4	-	Potential centralization risk	Note	-
5	-	Burning feature of `BR` tokens	Note	-

The details are provided in the following sections.

2.1 Recommendations

2.1.1 Add proper initializations in the `constructor()` of the contract `BR`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The role `FREEZER_ROLE` and the variable `freezeToRecipient` play important roles to curb the flow of illicit funds. It is recommended to initialize the two variables in the `BR` contract's `constructor()`.

```
13    constructor(address defaultAdmin, address minter) ERC20("Bedrock", "BR") {
14        require(defaultAdmin != address(0), "SYS001");
15        require(minter != address(0), "SYS001");
16        _grantRole(DEFAULT_ADMIN_ROLE, defaultAdmin);
17        _grantRole(MINTER_ROLE, minter);
18    }
```

Listing 2.1: BR.sol

Suggestion Initialize the `freezeToRecipient` and the `FREEZER_ROLE` in the `constructor()`.

2.1.2 Check `users.length` in the `freezeUsers()` and `unfreezeUsers()` functions

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `freezeUsers()` and `unfreezeUsers()` functions can emit the `UsersFrozen` or `UsersUnfrozen` event when the length of the input `users` is 0. This may lead to unexpected behavior in event processing. It is recommended to check the `users.length`.

```
70    function freezeUsers(address[] memory users) public onlyRole(FREEZER_ROLE) {
71        for (uint256 i = 0; i < users.length; ++i) {
72            frozenUsers[users[i]] = true;
73        }
74        emit UsersFrozen(users);
75    }
```

Listing 2.2: BR.sol

```
81    function unfreezeUsers(address[] memory users) public onlyRole(FREEZER_ROLE) {
82        for (uint256 i = 0; i < users.length; ++i) {
83            frozenUsers[users[i]] = false;
84        }
85        emit UsersUnfrozen(users);
86    }
```

Listing 2.3: BR.sol

Suggestion Add checks on `users.length`.

2.1.3 Add validation checks for `_msgSender()` in the function `batchTransfer()`

Status Confirmed

Introduced by `Version 1`

Description In the `BR` contract, the `batchTransfer()` function calls the `_transfer()` function within a loop. If the `_msgSender()` is frozen, the `_transfer()` function may revert due to an incorrect `recipient` (i.e., not `freezeToRecipient`). Therefore, it is recommended to add a validation check for the `_msgSender()` and unify the recipient before the loop for gas optimization.

```
102   function batchTransfer(address[] memory recipients, uint256[] memory amounts) external {
103       require(recipients.length > 0, "USR001");
104       require(recipients.length == amounts.length, "USR002");
105
106       for (uint256 i = 0; i < recipients.length; ++i) {
107           _transfer(_msgSender(), recipients[i], amounts[i]);
108       }
109   }
```

Listing 2.4: BR.sol

```
35    function _transfer(address sender, address recipient, uint256 amount) internal override {
36        if (frozenUsers[sender]) {
37            require(recipient == freezeToRecipient, "USR016");
38        }
39        super._transfer(sender, recipient, amount);
40    }
```

Listing 2.5: BR.sol

Suggestion Add validation checks for `_msgSender()` in the function `batchTransfer()`.

2.2 Notes

2.2.1 Potential centralization risk

Introduced by Version 1

Description There are a few roles (e.g., MINTER_ROLE and FREEZER_ROLE) in the contract that can conduct privileged operations (i.e., mint(), freezeUsers() and unfreezeUsers()), which introduces potential centralization risks. If the private key of the privileged account is lost or maliciously exploited, it could potentially lead to significant losses for users and the protocol.

2.2.2 Burning feature of BR tokens

Introduced by Version 1

Description The BR contract inherits the burning feature from the ERC20Burnable contract without modifications, allowing all token owners (including frozen users) to burn their tokens. This may pose potential risks if future developments involve calculations based on the totalSupply of the token.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS