our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Terplayer

BeraBTC

SECURITY REVIEW

Date: 3 April 2025

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and has secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Vyper, Rust, Cairo, Move and Go.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Terplayer - beraBTC

beraBTC is an ERC20 token pegged 1:1 to Bitcoin, with a unique issuance mechanism that integrates Physically Settled Futures and overcollateralized Honey stablecoin.

It represents a decentralized financial solution for Bitcoin trading, offering unparalleled security, liquidity, and transparency through its integration with Berachain.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** - results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** - results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** - losses will be limited but bearable - and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** - almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** - still relatively likely, although only conditionally possible
- **Low** - requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted 1 day, with a total of 16 hours dedicated by 2 researchers from the Shieldify team.

Overall, the code is well-written. The audit report identified two Critical/High-severity issues, two Medium-severity issues, one Low-severity issue, and two Informational findings. The impacful vulnerabilities stem from fees loss due to missing treasury collection mechanism and flawed or incomplete validation in relation to the blacklisting functionality.

5.1 Protocol Summary

Project Name	**Terplayer-beraBTC**
Repository	bera-bitcoin
Type of Project	Upgradeable ERC20
Audit Timeline	1 day
Review Commit Hash	19d509911de64a062dfaeef33f14e26f08b23c10
Fixes Review Commit Hash	4e6ba26757bed822416a51de3feba96ad469c158

5.2 Scope

The following smart contracts were in the scope of the security review:

File	nSLOC
contracts/BeraBitcoin.sol	119
Total	**119**

6. Findings Summary

The following issues have been identified, sorted by their severity:

- **Critical/High** issues: **2**
- **Medium** issues: **2**
- **Low** issues: **1**

ID	Title	Severity	Status
[C-01]	Protocol Loses `mint()` / `redeem()` Fees Due to Missing Treasury Collection Mechanism	Critical	Fixed
[H-01]	Blacklisted User Can Transfer Tokens via Approved Addresses	High	Fixed
[M-01]	Users Can Transfer Tokens to Blacklisted Addresses Causing Fund Locking	Medium	Fixed
[M-02]	Blacklisted Addresses Can Still Mint and Redeem Tokens Despite Restrictions	Medium	Fixed
[L-01]	Incorrect Recipient Data Type in the `redeem()` Function	Low	Acknowledged

7. Findings

[C-01] Protocol Loses `mint()` / `redeem()` Fees Due to Missing Treasury Collection Mechanism

Severity

Critical Risk

Description

The contract implements minting and redemption fees in the `custodianMint()` and `redeem()` functions but fails to properly collect and store these fees for the protocol. During minting, the fee is simply deducted from the minted amount without being allocated to a treasury. Similarly, during redemption, the fee is subtracted from the redeemed value without being captured. While the fee calculations are correctly implemented mathematically, the contract lacks both a treasury address variable and the logic to transfer these fees to protocol-controlled addresses. This creates a situation where protocol revenue that should be accruing to the project treasury is effectively being burned or lost.

Location of Affected Code

File: src/BeraBitcoin.sol#L178

```
    address to,
    uint256 value
) public onlyRole(CUSTODIAN_ROLE) nonReentrant {
    if (mintFeeRate > 0) {
        value -= (value * mintFeeRate) / BASE_RATE;
    }
    _mint(to, value);
    emit CustodianMinted(to, value);
}
```

File: src/BeraBitcoin.sol#L203

```
    uint256 value,
    string memory recipient
) public nonReentrant {
    uint256 burned = value;
    _burn(msg.sender, burned);
    if (redeemFeeRate > 0) {
        value -= (burned * redeemFeeRate) / BASE_RATE;
    }
    emit Redeemed(msg.sender, burned, value, recipient);
}
```

Impact

The protocol loses all potential revenue from its fee structure as they are not being collected.

Recommendation

Add treasury address state variable and add fees to this variable which are generated through `custodianMint()` and `redeem()`

Team Response

Fixed.

[H-01] Blacklisted User Can Transfer Tokens via Approved Addresses

Severity

High Risk

Description

The contract implements a blacklist mechanism to prevent certain addresses from performing token transfers. However, the implementation contains a critical flaw in the `transferFrom()` function where it only checks if the caller (`msg.sender`) is blacklisted, not the actual token owner (from address). This allows a blacklisted user to transfer tokens if they had previously approved another address before being blacklisted. The approved address can then call `transferFrom()` to move the blacklisted user's tokens, since the modifier only verifies the caller's status. The vulnerability stems from incomplete blacklist checks in the transfer authorization logic.

Location of Affected Code

File: src/BeraBitcoin.sol#L242

```
function transferFrom(address from, address to, uint256 amount) public
    override notBlacklisted returns (bool) {
    return super.transferFrom(from, to, amount);
}
```

File: src/BeraBitcoin.sol#L107

```
modifier notBlacklisted() {
    if (isBlacklisted(msg.sender)) revert AccountIsBlacklisted();
    _;
}
```

Impact

This vulnerability severely compromises the effectiveness of the blacklist feature. Malicious actors who get blacklisted can still move their funds through pre-approved addresses, defeating the purpose of the security measure. In scenarios where tokens need to be frozen, this flaw allows continued token movement, potentially enabling money laundering or helping attackers escape with stolen funds.

Recommendation

The fix requires modifying the `transferFrom()` function to check both the caller (`msg.sender`) and the token owner (from address) against the blacklist. The `notBlacklisted()` modifier should be extended to verify both addresses in transfer operations.

Team Response

Fixed.

[M-01] Users Can Transfer Tokens to Blacklisted Addresses Causing Fund Locking

Severity

Medium Risk

Description

The contract's `transfer()` function in `BeraBitcoin.sol` fails to verify whether the recipient address (`to` parameter) is blacklisted before executing the transfer. While the function properly checks that the sender (`msg.sender`) is not blacklisted through the `notBlacklisted()` modifier, it omits this critical check for the receiving address.

This oversight exists despite the contract maintaining a comprehensive blacklisting system intended to restrict token movements for flagged addresses. The vulnerability stems from incomplete validation in the transfer authorization logic, specifically in the inherited ERC20 `transfer()` function implementation.

Location of Affected Code

File: src/BeraBitcoin.sol#L234

```
function transfer(address to, uint256 amount) public override
    notBlacklisted returns (bool) {
    return super.transfer(to, amount);
}
```

Impact

This vulnerability can lead to locking of tokens when users transfer tokens to blacklisted addresses. Once tokens are transferred to a blacklisted address, they become effectively frozen since the recipient cannot transfer them out.

Recommendation

The fix requires modifying the `transfer()` function to include recipient address validation against the blacklist. The function should revert if either the sender or recipient is blacklisted.

Team Response

Fixed.

[M-02] Blacklisted Addresses Can Still Mint and Redeem Tokens Despite Restrictions

Severity

Medium Risk

Description

The contract fails to implement blacklist checks in critical token operations including `custodianMint()`, `excessStakeMint()`, and `redeem()` unctions. While the contract maintains a blacklist mapping and implements checks for standard transfers via the `notBlacklisted()` modifier, these critical administrative functions lack similar protections.

The `custodianMint()` and `excessStakeMint()` functions allow privileged roles to mint tokens to any address without verifying if the recipient is blacklisted. Similarly, the redeem function permits any caller to burn tokens without checking if their address is blacklisted.

Location of Affected Code

File: src/BeraBitcoin.sol#L178

```
function custodianMint(
    address to,
    uint256 value
) public onlyRole(CUSTODIAN_ROLE) nonReentrant {
    if (mintFeeRate > 0) {
        value -= (value * mintFeeRate) / BASE_RATE;
    }
    _mint(to, value);
    emit CustodianMinted(to, value);
}
```

File: src/BeraBitcoin.sol#L192

```
/// @notice Mints beraBTC by the excess stake role.
/// @param account The address to mint the beraBTC to.
/// @param value The amount of beraBTC to mint.
function excessStakeMint(
    address account,
    uint256 value
) public onlyRole(EXCESS_STAKE_ROLE) nonReentrant {
    _mint(account, value);
    emit ExcessStakeMinted(account, value);
}
```

File: src/BeraBitcoin.sol#L203

```
function redeem(
    uint256 value,
    string memory recipient
) public nonReentrant {
    uint256 burned = value;
    _burn(msg.sender, burned);
    if (redeemFeeRate > 0) {
        value -= (burned * redeemFeeRate) / BASE_RATE;
    }
    emit Redeemed(msg.sender, burned, value, recipient);
}
```

Impact

This vulnerability severely compromises the effectiveness of the blacklisting mechanism. Malicious actors who have been blacklisted can still receive newly minted tokens through the custodian or excess stake minting functions, allowing them to maintain token holdings despite sanctions. Additionally, blacklisted users can continue redeeming tokens, potentially enabling them to extract value from the system even after being flagged.

Recommendation

Add blacklist check inside `custodianMint()`, `excessStakeMint()`, and `redeem()` functions to check the user is receiving/burning tokens is blacklisted or not.

Team Response

Fixed.

[L-01] Incorrect Recipient Data Type in the `redeem()` Function

Severity

Low Risk

Description

The `redeem()` function in the `BeraBitcoin` contract incorrectly defines the recipient parameter as a string type when it should logically be an address type. This design flaw exists despite the parameter name clearly indicating it should represent a recipient address. While currently, the recipient value is only passed to the `Redeemed` event emission, this can create a problem if anything is handled off-chain using the event data, which can create a loss to users.

Location of Affected Code

File: src/BeraBitcoin.sol#L203

```
function redeem(
    uint256 value,
// @audit it should be address instead of string
    string memory recipient
) public nonReentrant {
    uint256 burned = value;
    _burn(msg.sender, burned);
    if (redeemFeeRate > 0) {
        value -= (burned * redeemFeeRate) / BASE_RATE;
    }
    emit Redeemed(msg.sender, burned, value, recipient);
}
```

Impact

The immediate impact is limited since the parameter is only used in event logging but can create problems if some logic runs through the data of event emission.

Recommendation

The secure fix requires changing the recipient parameter type from string to address to properly reflect its intended use.

Team Response

Acknowledged.

shieldify

Thank you!