



SOLIDProof
Bring trust into your projects

**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**

MADE IN GERMANY

Bit Reserve

AUDIT

SECURITY ASSESSMENT

16. July, 2024

FOR



Bit Reserve



SolidProof.io



@solidproof_io



Introduction	3
Disclaimer	3
Project Overview	4
Summary	4
Social Medias	4
Audit Summary	5
File Overview	6
Imported packages	6
Components	7
Exposed Functions	7
Capabilities	8
Inheritance Graph	9
Audit Information	10
Vulnerability & Risk Level	10
Auditing Strategy and Techniques Applied	11
Methodology	11
Overall Security	12
Upgradeability	12
Ownership	13
Ownership Privileges	14
Minting tokens	14
Burning tokens	15
Blacklist addresses	16
Fees and Tax	17
Lock User Funds	18
Centralization Privileges	19
Audit Results	20



Introduction

SolidProof.io is a brand of the officially registered company FutureVisions Deutschland, based in Germany. We're mainly focused on Blockchain Security such as Smart Contract Audits and KYC verification for project teams.

Solidproof.io assess potential security issues in the smart contracts implementations, review for potential inconsistencies between the code base and the whitepaper/documentation, and provide suggestions for improvement.

Disclaimer

SolidProof.io 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. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'...)

SolidProof.io Audits 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 technology proprietors. SolidProof Audits 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.

SolidProof.io 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. SolidProof's position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of the security or functionality of the technology we agree to analyze.



Project Overview

Summary

Project Name	Bit Reserve
Website	https://www.bitreserve.xyz/
About the project	Bitreserve leverages the advantages of DeFi to build extraordinary returns for BTC holders. This initiative not only releases vast potential but also heralds a new era for Bitcoin as an income-generating asset.
Chain	Core DAO
Language	Solidity
Codebase	https://github.com/bit-reserve/bit-reserve-contract/tree/main
Forked Status	This project is 1:1 forked with Vector reserve, The contract links are mentioned below: https://github.com/vectorreserve/vector-contracts/tree/main/contracts
Commit	3f3e82d73e2caf3f46b21ba4b89f79818701e45c
Unit Tests	Not Provided

Social Medias

Telegram	https://t.me/BitReserveFi
Twitter	https://x.com/BitReserveFi
Facebook	N/A
Instagram	N/A
GitHub	N/A
Reddit	N/A
Medium	N/A
Discord	N/A
YouTube	N/A
TikTok	N/A
LinkedIn	N/A



Audit Summary

Version	Delivery Date	Change Log
v1.0	17. June 2024	<ul style="list-style-type: none">· Layout Project· Automated/ Manual-Security Testing· Summary
v1.2	16. July 2024	<ul style="list-style-type: none">· Reaudit

Note – The following audit report presents a comprehensive security analysis of the smart contract utilized in the project that includes outside manipulation of the contract's functions in a malicious way. This analysis did not include functional testing (or unit testing) of the contract/s logic. We cannot guarantee 100% logical correctness of the contract as we did not functionally test it. This includes internal calculations in the formulae used in the contract.



File Overview

The Team provided us with the files that should be tested in the security assessment. This audit covered the following files listed below with an SHA-1 Hash.

N/A

Please note: Files with a different hash value than in this table have been modified after the security check, either intentionally or unintentionally. A different hash value may (but need not) be an indication of a changed state or potential vulnerability that was not the subject of this scan.

Imported packages.

Used code from other Frameworks/Smart Contracts.

Dependency / Import Path	Count
@layerzerolabs/lz-evm-oapp-v2/contracts/oft/OFT.sol	1
@openzeppelin/contracts/access/Ownable.sol	5
@openzeppelin/contracts/token/ERC20/ERC20.sol	5
@openzeppelin/contracts/token/ERC20/IERC20.sol	1
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	5
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	3

Note for Investors: We only audited contracts mentioned in the scope above. All contracts related to the project apart from that are not a part of the audit, and we cannot comment on its security and are not responsible for it in any way. This contract is a forked contract of Vector reserve so the issues that were present in their contract will remain the same and we recommended the user do their own research.



External/Public functions

External/public functions are functions that can be called from outside of a contract, i.e., they can be accessed by other contracts or external accounts on the blockchain. These functions are specified using the function declaration's external or public visibility modifier.

State variables

State variables are variables that are stored on the blockchain as part of the contract's state. They are declared at the contract level and can be accessed and modified by any function within the contract. State variables can be needed within the visibility modifier, such as public, private, or internal, which determines the access level of the variable.

Components

 Contracts	 Libraries	 Interfaces	 Abstract
8	8	14	0

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

 Public	 Payable			
186	9			
External	Internal	Private	Pure	View
157	189	5	27	66

StateVariables

Total	 Public
87	70



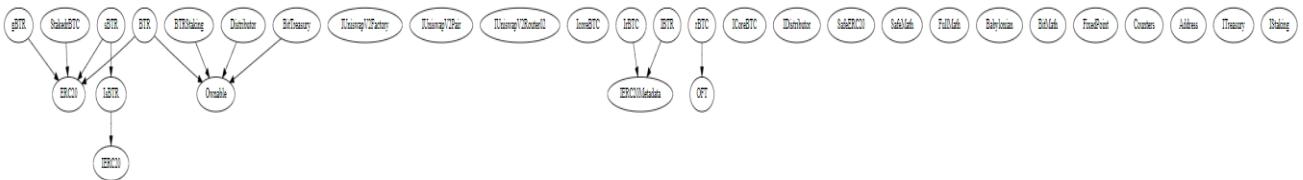
Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts	
0.8.19 0.8.20 ≥0.5.0 ≥0.7.5 ^0.7.5	-----	Yes	yes (6 asm blocks)	-----	
Transfers ETH	Low-Level Calls	Delegate Call	Uses Hash Functions	ECRecover	New/Create/Create2
yes		yes			



Inheritance Graph

An inheritance graph is a graphical representation of the inheritance hierarchy among contracts. In object-oriented programming, inheritance is a mechanism that allows one class (or contract, in the case of Solidity) to inherit properties and methods from another class. It shows the relationships between different contracts and how they are related to each other through inheritance.



Audit Information

Vulnerability & Risk Level

Risk represents the probability that a certain source threat will exploit the vulnerability and the impact of that event on the organization or system. The risk level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 - 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 - 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 - 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 - 1.9	A vulnerability that has informational character but is not affecting any of the code.	An observation that does not determine a level of risk



Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to check the repository for security-related issues, code quality, and compliance with specifications and best practices. To this end, our team of experienced pen-testers and smart contract developers reviewed the code line by line and documented any issues discovered.

We check every file manually. We use automated tools only so that they help us achieve faster and better results.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
 - a. Reviewing the specifications, sources, and instructions provided to SolidProof to ensure we understand the size, scope, and functionality of the smart contract.
 - b. Manual review of the code, i.e., reading the source code line by line to identify potential vulnerabilities.
 - c. Comparison to the specification, i.e., verifying that the code does what is described in the specifications, sources, and instructions provided to SolidProof.
2. Testing and automated analysis that includes the following:
 - a. Test coverage analysis determines whether test cases cover code and how much code is executed when those test cases are executed.
 - b. Symbolic execution, which is analysing a program to determine what inputs cause each part of a program to execute.
3. Review best practices, i.e., review smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on best practices, recommendations, and research from industry and academia.
4. Concrete, itemized and actionable recommendations to help you secure your smart contracts.



Overall Security Upgradeability

Contract is not an upgradable	 Deployer cannot update the contract with new functionalities.
Description	The contract is not an upgradeable contract. The Deployer is not able to change or add any functionalities to the contract after deploying.
Comment	N/A





Ownership

Contract ownership is not renounced.

 The ownership is not renounced.

Description

The owner has not renounced the ownership that means that the owner retains control over the contract's operations, including the ability to execute functions that may impact the contract's users or stakeholders. This can lead to several potential issues, including:

- Centralizations
- The owner has significant control over contract's operations.

Comment

N/A

Note – *The contract cannot be considered as renounced till it is not deployed or having some functionality that can change the state of the contract.*

Ownership Privileges

These functions can be dangerous. Please note that abuse can lead to financial loss. We have a guide where you can learn more about these Functions.

Minting tokens

Minting tokens refer to the process of creating new tokens in a cryptocurrency or blockchain network. This process is typically performed by the project's owner or designated authority, who has the ability to add new tokens to the network's total supply.

Contract owner can mint new tokens.

The owner can mint new tokens.

Description	The owner is able to mint new tokens once the contract is deployed.
Comment	The approved minter in the contract can mint new tokens which is not recommended as there must be a max token limit so that the owner or deployer of the contract can not mint tokens more than that particular amount.

File/Line(s): L144-151

Codebase: BitTreasury.sol

```
ftrace | funcSig
function mint(address _to, uint256 _amount) external {
    require(
        approvedMinter[msg.sender],
        "msg.sender is not approved minter"
    );
    require(excessReserves() >= _amount, "Too low excess reserves");
    IBTR(BTR).mint(_to, _amount);
}
```

File/Line(s): L599-602

Codebase: BTR.sol

```
ftrace | funcSig
function mint(address account_1, uint256 amount_1) external {
    require(msg.sender == treasury, "Not Treasury");
    _mint(account_1, amount_1);
}
```



Burning tokens

Burning tokens is the process of permanently destroying a certain number of tokens, reducing the total supply of a cryptocurrency or token. This is usually done to increase the value of the remaining tokens, as the reduced supply can create scarcity and potentially drive up demand.

Contract owner cannot burn tokens	 The owner cannot burn tokens.
-----------------------------------	---

Description	The owner is not able burn tokens without any allowances.
Comment	N/A



Blacklist addresses

Blacklisting addresses in smart contracts is the process of adding a certain address to a blacklist, effectively preventing them from accessing or participating in certain functionalities or transactions within the contract. This can be useful in preventing fraudulent or malicious activities, such as hacking attempts or money laundering.

Contract owner cannot blacklist addresses.

 **The owner cannot blacklist wallets.**

Description	The owner cannot blacklist wallets from transferring of tokens.
Comment	N/A



Fees and Tax

In some smart contracts, the owner or creator of the contract can set fees for certain actions or operations within the contract. These fees can be used to cover the cost of running the contract, such as paying for gas fees or compensating the contract's owner for their time and effort in developing and maintaining the contract.

Contract owner can set fees more than 25%.



The owner cannot set fees more than 25%.

Description	The owner cannot set fees of more than 25%.
Comment	N/A



Lock User Funds

In a smart contract, locking refers to the process of restricting access to certain tokens or assets for a specified period of time. When token or assets are locked in a smart contract, they cannot be transferred or used until the lock-up period has expired or certain conditions have been met.

Contract owner cannot lock function.



The owner cannot lock function.

Description	The owner cannot lock the contract.
Comment	N/A



Centralization Privileges

Centralization can arise when one or more parties have privileged access or control over the contract's functionality, data, or decision-making. This can occur, for example, if the contract is controlled by a single entity or if certain participants have special permissions or abilities that others do not.

In the project, there are authorities that have access to the following functions:

File	Privileges
BitBTC.sol	<ul style="list-style-type: none">➤ The owner can add/remove tokens from the approved tokens list.➤ The owner can update the amount of BTCperLST tokens for the particular token from the approved token list.➤ The owner can update the restaked LST router address.➤ The owner can enable redemption.➤ The owner can add/remove approved managers from the contract.➤ The owner can claim tokens excluding staked tokens from the contract.➤ The owner can enable deposit functionality in the contract only once.➤ The owner can set the redeem fees to not more than 51%.➤ The owner can update the redeem fee receiver wallet to any arbitrary wallet, including zero.
BTR.sol	<ul style="list-style-type: none">➤ The owner can add/remove the automated market maker pair address.➤ The treasury wallet address can mint an unlimited amount of tokens after the initial deployment.➤ The owner can initialize the token only once.➤ The owner can update the distributor in the contract.➤ The owner can enable trading only once.➤ The owner can update the minimum swap threshold amount between 0.001% to 0.50% of the total supply.➤ The owner can enable/disable swapping.➤ The owner can update the buy and sell fees to not more than 3%.



sBTR.sol
BitTreasury.sol
RewardDistributor.sol
BTRStaking.sol

- The owner can exclude wallets from fees.
- The owner can update the team wallet address.
- The owner can claim stuck tokens and ETH from the contract.
- The initializer can set the index only once.
- The initializer can initialize the staking contract only once.
- The staking contract can rebase and increase the supply of the token.
- The owner can update the redemption percentage to not more than 100%.
- The owner can add redeemable tokens to the contract.
- The owner can add/remove approved minters in the contract.
- The owner can add/remove approved senders into the contract.
- The owner can update the rBTC contract address.
- The approved minters can mint new tokens to not more than the excess reserve in the contract.
- The approved minters can transfer tokens from the contract.
- The BTR staking contract can distribute tokens.
- The owner can update the BTR reward rate to any arbitrary amount less than the rate denominator, including zero.
- The owner can update the sr BTC Reward per EPOCH to any arbitrary amount including zero.
- The owner can withdraw tokens and ETH from the contract.
- The owner can update the distributor contract address.

Recommendations

To avoid potential hacking risks, it is advisable for the client to manage the private key of the privileged account with care. Additionally, we recommend enhancing the security practices of centralized privileges or roles in the protocol through a decentralized mechanism or smart-contract-based accounts, such as multi-signature wallets.



Here are some suggestions of what the client can do:

- Consider using multi-signature wallets: Multi-signature wallets require multiple parties to sign off on a transaction before it can be executed, providing an extra layer of security e.g. Gnosis Safe
- Use of a timelock at least with a latency of e.g. 48-72 hours for awareness of privileged operations
- Introduce a DAO/Governance/Voting module to increase transparency and user involvement
- Consider Renouncing the ownership so that the owner cannot modify any state variables of the contract anymore. Make sure to set up everything before renouncing.



Audit Result

Critical Issues

No critical issues

High Issues

#1 | Missing zero check.

File	Severity	Location	Status
BitTreasury.sol	High	L85-90	Fixed

Description – The owner can set any arbitrary value in the redemption percentage redeemable value as this can cause the loss of funds for the user if it is set to zero. The user will not be able to receive any redeemed value into his wallet. It is recommended to add the zero ‘require’ check so that the user will get the redeemed value into his wallet.

#2 | Missing ‘isContract’ check.

File	Severity	Location	Status
BitTreasury.sol	High	L131-134	Fixed
BTR.sol	High	L619-623	Fixed
BTRStaking.sol	High	L132-134	Fixed

Description – The owner can set any arbitrary address in the place of contract address which is not recommended as if the address is set to an arbitrary address the functionality of the contract will fail. It is recommended to add that the address should be a contract address to avoid these circumstances in the contract.



Medium Issue

#1 | Liquidity is added to an externally owned account.

File	Severity	Location	Status
BTR.sol	Medium	L479-491	ACK

Description – The contract's liquidity is automatically added to the 'Treasury' address, which is not recommended because, in an extreme scenario, this can be used to drain liquidity from the contract.

#2 | Transfer of tokens without enabling trade.

File	Severity	Location	Status
BTR.sol	Medium	L379-457	ACK

Description – The trading needs to be enabled by the owner in order for regular users to transfer tokens. On the contrary, the owner can authorize addresses manually and those addresses will be able to trade tokens. This functionality can be exploited in the following way, For example, there is a presale and the wallets used for the presale can be authorized by the owner. All the tokens obtained can be consolidated into a final wallet address and facilitate trading and selling of the acquired tokens, the last wallet address can be authorized.

#3 | Missing 'require' check (Potential honeypot)

File	Severity	Location	Status
BTR.sol	Medium	L699-703	ACK

Description – The owner can set any arbitrary amount to the marketing wallet, LP wallet, and Dev wallet address, including zero address as this can lead to a potential honeypot if the owner has set the address to a contract address that cannot receive ETH. It is recommended that the address cannot be set to a contract address that cannot receive ETH to avoid these circumstances.



Low Issue

#1 | Missing events arithmetic.

File	Severity	Location	Status
BitBTC.sol	Low	L264-267, L269-271	ACK
BTR.sol	Low	L626-630, L633-646, L650-652, L655-671, L674-690	ACK

Description – Emit all the critical parameter changes.

#2 | Missing zero or dead address check.

File	Severity	Location	Status
BitBTC.sol	Low	L232-235, L238-241, L269-271	Fixed

Description – It is recommended that the address cannot be set to zero or dead address.

Informational Issue

#1 | NatSpec Documentation missing.

File	Severity	Location	Status
All	Informational	--	ACK

Description – If you started to comment on your code, also comment on all other functions, variables, etc.

Legend for the Issue Status

Attribute or Symbol	Meaning
Open	The issue is not fixed by the project team.
Fixed	The issue is fixed by the project team.
Acknowledged(ACK)	The issue has been acknowledged or declared as part of business logic.



**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**

MADE IN GERMANY