



PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Blackrock Fund

02 March 2024



paladinsec.co



info@paladinsec.co

Table of Contents

Table of Contents	2
Disclaimer	3
1 Overview	4
1.1 Summary	4
1.2 Contracts Assessed	5
1.3 Findings Summary	6
1.3.1 General Issues	7
1.3.2 401k and lpstaking	7
1.3.3 btc	8
1.3.4 etf	8
1.3.5 aBTC	8
1.3.6 breth	9
2 Findings	10
2.1 General Issues	10
2.1.1 Issues & Recommendations	11
2.2 401k and lpstaking	14
2.2.1 Privileged Functions	14
2.2.2 Issues & Recommendations	15
2.3 btc	27
2.3.1 Privileged Functions	27
2.3.2 Issues & Recommendations	28
2.4 etf	31
2.4.1 Privileged Functions	31
2.4.2 Issues & Recommendations	32
2.5 aBTC	37
2.5.1 Privileged Functions	37
2.5.2 Issues & Recommendations	38
2.6 breth	40
2.6.1 Privileged Functions	40
2.6.2 Issues & Recommendations	41

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocacy for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Blackrock Fund on the Ethereum network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

NOTE: Throughout the report, BTC refers to the Blackrock Fund token and not Bitcoin.

1.1 Summary

Project Name	Blackrock Fund
URL	https://blackrockfund.finance
Platform	Ethereum
Language	Solidity
Preliminary Contracts	https://github.com/BlackrockFund/btc-contracts/tree/ba65f611beef7ff61b66fda303646b183b009b06 for 401k, lpstaking, BTC, etf, aBTC https://github.com/BlackrockFund/btc-contracts/tree/e0db5eb3a50c0eaa8cd1dcb5b4db4e3116991ecf for breth
Resolution 1	https://github.com/BlackrockFund/btc-contracts/commit/c8760704bccc1ae0a7413f50033637d21075ac96

1.2 Contracts Assessed

Name	Contract	Live Code Match
401k*	0x1FDe0d2F44539789256D94D1784a86bF77D66DD0	UNMATCHED
lpstaking*	0x1e4A10d18698E4450E13b4E8EF361a5841850611	UNMATCHED
btc	0xbD6323A83b613F668687014E8A5852079494fB68	✓ MATCH
etf	0xef9E6123Eff4Dae9dB1E6d585884b45b369a8aD1	✓ MATCH
aBTC	0xa709aaD0691Fc67279577566640ae1D6515c1b81	✓ MATCH
breth	Undeployed	N/A

The 401k and lpstaking contracts listed are the ones that have been deployed before the audit. Please follow the team for announcements on when the audited contracts with the updated code are deployed.

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● Governance	4	-	-	4
● High	5	2	-	3
● Medium	9	1	-	8
● Low	11	-	-	11
● Informational	8	-	-	8
Total	37	3	-	34

Classification of Issues

Severity	Description
● Governance	Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example.
● High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
● Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
● Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
● Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 General Issues

ID	Severity	Summary	Status
01	GOV	Centralised risk	ACKNOWLEDGED
02	INFO	The usage of SafeMath is obsolete	ACKNOWLEDGED
03	INFO	Lack of events	ACKNOWLEDGED
04	INFO	Typographical issues	ACKNOWLEDGED

1.3.2 401k and lpstaking

ID	Severity	Summary	Status
05	GOV	Governance can call several functions that can cause funds to be stuck in the contract	ACKNOWLEDGED
06	HIGH	Lack of emergencyWithdraw	✓ RESOLVED
07	MEDIUM	Users who deposit after having accrued rewards are debasing their own BTC	ACKNOWLEDGED
08	MEDIUM	Withdrawal fee is applied on all of the previous deposits even if the last deposit is a small amount	ACKNOWLEDGED
09	MEDIUM	BTC and USDC rewards are claimed together	ACKNOWLEDGED
10	MEDIUM	It is disadvantageous for users to stake LPs compared to staking BTC	ACKNOWLEDGED
11	MEDIUM	withdraw() can be backrun by an adversary	ACKNOWLEDGED
12	MEDIUM	The contract works only with USDC with 6 decimals	ACKNOWLEDGED
13	MEDIUM	BTC rewards collected are always less than what is calculated	ACKNOWLEDGED
14	MEDIUM	Insufficient check of totalAllocPoint	✓ RESOLVED
15	LOW	The contract does not work with tokens that have a fee on transfer	ACKNOWLEDGED
16	LOW	maUSDCPerSecond is declared incorrectly to be 1e20 instead of 1e18	ACKNOWLEDGED
17	LOW	The 401k contract seems unnecessary	ACKNOWLEDGED
18	LOW	Rounding vulnerability of tokens with very large supply can cause users to receive zero rewards	ACKNOWLEDGED
19	INFO	Typographical issues	ACKNOWLEDGED

1.3.3 btc

ID	Severity	Summary	Status
20	GOV	Governance can mint/rebase freely	ACKNOWLEDGED
21	HIGH	Rebase mechanism is very fragile	ACKNOWLEDGED
22	HIGH	DOMAIN_SEPARATOR is not implemented in BTC.sol permit()	ACKNOWLEDGED
23	INFO	Typographical issues and gas optimizations	ACKNOWLEDGED

1.3.4 etf

ID	Severity	Summary	Status
24	GOV	The bond price can be set to any value	ACKNOWLEDGED
25	HIGH	Vesting applies to the full bonded amount	ACKNOWLEDGED
26	MEDIUM	The bond cap can be circumvented	ACKNOWLEDGED
27	LOW	Sybil attack is still possible	ACKNOWLEDGED
28	INFO	Gas optimizations	ACKNOWLEDGED

1.3.5 aBTC

ID	Severity	Summary	Status
29	LOW	Governance also pays fees for redeeming the collected fees	ACKNOWLEDGED
30	LOW	Fees are rounded down	ACKNOWLEDGED
31	LOW	Small precision on fees calculation	ACKNOWLEDGED
32	INFO	Gas optimizations	ACKNOWLEDGED

1.3.6 breth

ID	Severity	Summary	Status
33	HIGH	Protocol can be insolvent during a LIDO slashing event	✓ RESOLVED
34	LOW	Fees are rounded down	ACKNOWLEDGED
35	LOW	stETH could be lost when redeeming with a contract	ACKNOWLEDGED
36	LOW	Future airdrops can be stuck in the contract	ACKNOWLEDGED
37	INFO	Gas optimizations	ACKNOWLEDGED

2 Findings

2.1 General Issues

The issues listed in this section apply to the protocol as a whole. Please read through them carefully and take care to apply the fixes across the relevant contracts.



2.1.1 Issues & Recommendations

Issue #01	Centralised risk
Severity	 GOVERNANCE
Description	Throughout the codebase, we have noticed several use cases where a malicious owner or erroneous actions done by the owner can block or lose user funds. Most of the issues have been already reported in their corresponding sections but to mention an example: Within btc, the MINTER_ROLE can simply rebase/mint tokens freely without any restrictions which can increase/decrease the total supply or move the scaling factor as desired. This can cause a loss of funds for users in various scenarios.
Recommendation	Consider having the owners of the contracts held under a KYC-ed multi-signature wallet. We also highly recommend a Timelock contract to be put in place with at least 24 hours of waiting time before a governance action can be applied. This can strengthen the security and can alert users in case an erroneous action is intended to be executed.
Resolution	 ACKNOWLEDGED

Issue #02 The usage of SafeMath is obsolete	
Severity	 INFORMATIONAL
Description	Throughout the codebase, SafeMath is used even if the Solidity version is > 0.8.0 which protects against underflows and overflows.
Recommendation	Consider removing the SafeMath usage to make the code more readable and save gas.
Resolution	 ACKNOWLEDGED

Issue #03 Lack of events	
Severity	 INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. As an example, the following functions do not emit events (etf.sol): - claim - bond - setMAXtx - recoverTreasuryTokens - setvestingTime There are too many functions throughout all contracts without events so we chose to only point out a few of them. Every function that modifies the state of a contract should emit an event.
Recommendation	Consider emitting events for the functions that modify the state of the contract.
Resolution	 ACKNOWLEDGED

Issue #04	Typographical issues
Severity	INFORMATIONAL
Description	Several variables and functions do not follow conventional naming conventions, here are some examples (non-exhaustive): <u>etf.sol</u> setceil should be setCeil setbondPrice should be setBondPrice. Consider going through the contract to update the names of variables and functions. — Several imports are not used. For example, btc.sol contains a btc contract that is not used. — The contracts are not formatted properly which makes them pretty hard to understand.
Recommendation	Consider fixing the typographical issues.
Resolution	ACKNOWLEDGED



2.2 401k and lpstaking

401k and lpstaking are an implementation of the Masterchef contract by SushiSwap. The version done by the Blackrock Fund team distributes BTC and USDC based on a pool's allocation power and amount of rewards per second. 401k allows users to stake BTC and receive rewards in both BTC and USDC. lpstaking allows users to stake Uniswap LP tokens and receive both BTC and USDC as rewards. A key difference here is that BTC is burned when being supplied to 401k, whereas the LP tokens remain in the contract when supplied to lpstaking.

The Governance can always pause withdrawals, causing user funds to be stuck in the contract.

2.2.1 Privileged Functions

- `supplyRewards`
- `closeWithdraw`
- `setBTCPerSecond`
- `setUSDCPerSecond`
- `add`
- `set`
- `updateRewards`
- `transferOwnership`
- `renounceOwnership`

2.2.2 Issues & Recommendations

Issue #05	Governance can call several functions that can cause funds to be stuck in the contract
Severity	GOVERNANCE
Description	The current implementation of the Masterchef contains several mechanisms that can end up bricking the contract. 1. withdrawable can be set by the owner to true which blocks withdrawals . 2. The BTC/USDC addresses can be set to any address via updateRewards — these can be set to incorrect addresses. 3. Adding too many pools will eventually lead to massUpdatePools reverting which will block the whole functionality. 4. The set function does not check if the pool (pid) exists which can dilute the rewards.
Recommendation	Recommendations are as follows: 1. Consider if withdrawable is necessary — from our point of view, this should not be used but if it is, consider keeping the contract under a KYC-ed multi-signature wallet. 2. We see no point to change the BTC/USDC addresses. A new Masterchef can be deployed with new addresses if needed. 3. Consider adding an upper limit for pools to avoid reverting with an out of gas error. 4. Consider checking if the pid exists before updating emissions.
Resolution	ACKNOWLEDGED

Issue #06	Lack of emergencyWithdraw
Severity	 HIGH SEVERITY
Description	Every Masterchef contract has an emergencyWithdraw function which allows users to exit the Masterchef and forfeit their rewards in case the contract malfunctions. This function is not present in the contract which can lead to funds being stuck in the contract.
Recommendation	Consider implementing an emergencyWithdraw function. Example on Sushiswap: https://github.com/sushiswap/masterchef/blob/master/contracts/MasterChef.sol#L273
Resolution	 RESOLVED



Issue #07**Users who deposit after having accrued rewards are debasing their own BTC****Severity** MEDIUM SEVERITY**Description**

In a regular Masterchef contract, whenever tokens are withdrawn or deposited, rewards are first collected. In this case, BTC rewards are given via `claimtokenRebase()`. `claimtokenRebase()` mints rewards for users then debases based on the amount of rewards earned.

However, the issue here is that the debasing happens before a user's BTC tokens are transferred. As a result, users are erroneously punished for staking, despite it being an encouraged action of the protocol based on the game theory flywheel.

Note that in the withdrawal function, this is handled correctly. Rebase happens before BTC is transferred to users. This way, users' BTC that are staked are safe from the debase mechanism.

Recommendation

Consider swapping the `safeTransferFrom` logic with reward emission logic so that users can deposit their BTC before the debasing happens due to rewards collected in the 401k contract.

Resolution ACKNOWLEDGED

Team comments: "This is so that people can compound using 1 transaction by submitting a deposit tx with parameters amount up to their (balance + rewards). The debase effect the rewards of that user have on their \$BTC is negligible in a big context as a whole."

Issue #08

Withdrawal fee is applied on all of the previous deposits even if the last deposit is a small amount

Severity

 MEDIUM SEVERITY

Description

The protocol applies a withdrawal fee on early withdrawal, but the implementation is too simple. It simply checks for the `lastDepositTime` and applies the fee on any amount withdrawn.

To illustrate this vulnerability, we will use a simple example.

1. Alice deposits 10000 BTC at $t = 100$.
2. 1 year has passed, Alice's 10000 BTC deposit should no longer have any withdrawal fee.
3. Alice decides to make a small deposit of 10 BTC.
4. One day later, Alice withdraws all of her BTC - 10010 BTC.
5. Alice is aware of the penalty of early withdrawal, but expects that this penalty is applied only on 10 BTC and not 10000 BTC which has been staked for a year already.
6. Alice withdraws with only 9009 BTC.

Recommendation

Consider tracking the total amount of deposits where two days have not passed and take the fee based on this amount instead of the full staked amount.

Note that there could be increased complexity for implementing this. One possible implementation is to track every deposit with their respective timestamp, and use binary search to find the latest timestamp from two days.

Resolution

 ACKNOWLEDGED

Team comments: "Our mechanism dictates that any withdraw/deposit action will reset the clock for all \$BTC staked."

If this has been documented to users, the issue can be Medium instead of High severity.

Severity

 MEDIUM SEVERITY

Description

We will be explaining two cases where the rewards can be completely lost if one of the BTC or USDC rewards mechanisms fails.

USDC can blacklist addresses. When an address is blacklisted, USDC cannot be transferred to and from the address.

In both the `401k.sol` and `lpstaking.sol` contracts, USDC is sent to the users when rewards have been accrued. This means that if a user stakes their BTC/LP token, accrues some USDC rewards, and then tries to withdraw, it will always revert if their address has been blacklisted. Their BTC/LP tokens will then be stuck in the contract forever.

For BTC, `claimtokenRebase` is called, and similar to USDC, if that fails, both USDC and BTC rewards will be lost.

Recommendation

First, the rewards distribution should be split into two `if` statements — each reward should be distributed for each token if it exists. Currently, the rewards distribution is triggered if one of them is greater than 0.

```
if(pending > 0 || USDCpending >0) {  
    BTC.claimtokenRebase(msg.sender, pending);  
    USDC.transfer(msg.sender, USDCpending);  
}
```

For USDC, there should be a check to see if `msg.sender` is blacklisted, and if so, skip USDC rewards.

For BTC, there is not an easy solution to circumvent this other than implementing an emergency withdrawal function to let the user get their staked tokens. We will add this into a separate issue.

Resolution

 ACKNOWLEDGED

Issue #10**It is disadvantageous for users to stake LPs compared to staking BTC****Severity** MEDIUM SEVERITY**Description**

The protocol operates based on the 3,3 game theory. Its purpose is to encourage users to stake instead of simply holding on to their BTC.

Users can stake via 2 ways:

1. Stake directly using BTC
2. Contribute to the Uniswap BTC-ETH liquidity pool and stake the LP token.

Ideally, we want to ensure that both ways of staking has equal EV since they are both positive actions and encouraged by the protocol.

However, BTC in the Uniswap liquidity pool can debase. Users who stake their LP are rewarded less assuming that the emission for both staking pools are the same. In fact, it is possible for users to claim back less tokens in total when they convert their LP token back to BTC and ETH due to the debasing mechanism. Users will hold LP tokens owning a percentage of a pool that grows smaller overtime.

Currently on the live contracts, staking in BTC is more rewarding for users than staking in LP (BTCPerSecond is higher).

Recommendation

Consider rewarding the LP staking pool with more BTCPerSecond compared to the BTC pool to compensate for the inherent loss due to debasing.

Resolution ACKNOWLEDGED

Team comments: "While BTCPerSecond in LP staking pool is lower currently, the amount of stakes in the LP pool is lower, hence APR is still higher for users who stake via the LP pool. Protocol can adjust emitted rewards based on APR."

Issue #11**withdraw() can be backrun by an adversary****Severity** MEDIUM SEVERITY**Description**

Whales who have staked a large amount of BTC are vulnerable to this. An adversary can accrue a large amount of BTC rewards without triggering a withdrawal or deposit, wait for a victim who makes a large withdrawal from the staking contract, and then backruns him by collecting the BTC reward.

This way, due to the large BTC reward, the debasing effects would be higher, and the victim who has just made a large withdrawal would have his BTC rebased down before he can sell.

Note that this attack is profitable for the adversary, as debasing of another user's BTC is beneficial for themselves (the adversary's BTC is likely still in the staking contract which is protected from debasing).

Recommendation

The implementation design of the protocol and nature of the EVM results in this vulnerability. The only way this attack can be prevented is for the user to bundle their large withdrawal transaction with a sale transaction.

Resolution ACKNOWLEDGED

Issue #12**The contract works only with USDC with 6 decimals****Severity** MEDIUM SEVERITY**Description**

The current implementation of USDC rewards supports only 6 decimals as the pending rewards are divided by $1e12$ twice to support the 6 decimals of USDC.

This contract will not work on BSC or if a token that does not have 6 decimals is used in a new Masterchef instance.

Recommendation

Consider not using any non-6 decimals tokens as rewards instead of USDC. If this protocol is deployed on BSC, consider not using USDC.

Another possibility will be to use a proper denomination when setting the rewards per seconds for USDC which should account for the token's decimals and not set with the assumption that the token has 18 decimals when in fact it does not.

Resolution ACKNOWLEDGED

The team will not be using USDC for any potential deployment on BSC and will not use any other non-6 decimal tokens.



Severity

 MEDIUM SEVERITY

Description

Users can use the function `pendingBTC()` to view the amount of pending BTC in the staking contracts. This calculates the amount of BTC rewards accrued so far in the contract.

However, this is not an accurate representation of how much reward users will actually receive. Whenever tokens are claimed with `_claimtokensWRebase()`, `_mint()` occurs before `_rebase()`. The amount of BTC that users receive will always be rebased downwards, leading to a lower than expected amount of BTC received.

Recommendation

Within `_claimtokensWRebase()`, consider rebasing downwards before tokens are minted so that the rewards received is exactly what is calculated.

Resolution

 ACKNOWLEDGED

Issue #14 Insufficient check of totalAllocPoint	
Severity	 MEDIUM SEVERITY
Location	<u>Line 206 & 219</u> <pre>if (block.timestamp > pool.lastRewardTime && lpSupply != 0) {</pre> <u>Line 242</u> <pre>if (lpSupply == 0) {</pre>
Description	Within updatePool, pendingUSDC and pendingBTC, a check for totalAllocPoint is missing when we validate the amount of rewards to be distributed per share. The check should be done alongside with lpSupply != 0 or lpSupply == 0. If this variable is ever set to 0, the whole functionality of the masterchef will break and user funds will be stuck in the contract.
Recommendation	Consider adding the check that totalAllocPoint != 0 for pendingUSDC and totalAllocPoint == 0 for updatePool.
Resolution	 RESOLVED The recommended checks have been added.

Issue #15 The contract does not work with tokens that have a fee on transfer	
Severity	 LOW SEVERITY
Description	The contract does not work with tokens that have a fee on transfer as the balances are not checked before and after transfers which will render an incorrect amount of stakes present in the contract
Recommendation	Consider implementing a before/after balance check and update the amounts accordingly or consider not using these tokens.
Resolution	 ACKNOWLEDGED

Issue #16	maUSDCPerSecond is declared incorrectly to be 1e20 instead of 1e18
Severity	 LOW SEVERITY
Description	Based on the documentation, max BTC per second should not be higher than 1 per second, and this value should be 1e18. It is currently set at 1e20, which is 100 BTC per second. Furthermore, this variable is incorrectly named for USDC when it should be for BTC. Another variable should be present for USDC that should be 1e6 or another maximum variable for USDC.
Recommendation	Consider changing <code>maUSDCPerSecond = 1e20</code> to <code>maBTCPerSecond = 1e18</code> . Also consider adding a second variable <code>maUSDCPerSecond</code> with a proper maximum value.
Resolution	 ACKNOWLEDGED

Issue #17	The 401k contract seems unnecessary
Severity	 LOW SEVERITY
Description	The sole purpose of the 401k contract is supposed to let users stake BTC and get BTC as a reward. For this use-case, the Masterchef contract seems inappropriate and instead the Sushiswap staking contract should be used.
Recommendation	Consider replacing the whole functionality with Sushiswap's staking rewards implementation. An example can be seen: https://github.com/lidofinance/staking-rewards-sushi/blob/main/contracts/StakingRewards.sol Of course this should be updated to use a more recent Solidity version.
Resolution	 ACKNOWLEDGED

Issue #18	Rounding vulnerability of tokens with very large supply can cause users to receive zero rewards
Severity	 LOW SEVERITY
Description	The Masterchef accepts any token as pools to be rewarded. updatePool uses lpSupply to calculate how much reward per share should be distributed. If this variable is a large value then the rewards will be rounded down to 0.
Recommendation	We advise using a higher precision than 1e12 to avoid this. Based on previous implementations, a 1e18 precision should be sufficient.
Resolution	 ACKNOWLEDGED

Issue #19	Typographical issues
Severity	 INFORMATIONAL
Description	The increase/decrease allocation functions are not used. — The supplyRewards function for USDC is missing, which renders USDCdistributed redundant. — The EmergencyWithdraw event is not used. If the emergencyWithdraw function is implemented then this event should be used.
Recommendation	Consider fixing the typographical errors.
Resolution	 ACKNOWLEDGED

2.3 btc

btc is the primary token of BlackrockFund. It is an ERC20 rebase token with permit functionality. The rebase mechanism occurs only when MINTER_ROLE calls `claimtokenRebase` or REBASER_ROLE calls `rebase`.

MINTER_ROLE can mint the underlying supply or mint BTC freely without restrictions as long as the scaling factor is kept under the maximum scaling factor.

The rebase mechanism is not strictly enforced on any supply change and can go out of hand as several actors can change the supply without restrictions.

2.3.1 Privileged Functions

- `setLP` [Owner]
- `mint` [MINTER_ROLE]
- `mintUnderlying` [MINTER_ROLE]
- `rebase` [REBASER_ROLE]
- `claimtokenRebase` [MINTER_ROLE]
- `updateRebaseRatio` [OWNER]
- `updateTreasury` [OWNER]
- `grantRole` [ROLE_ADMIN]
- `revokeRole` [ROLE_ADMIN]

2.3.2 Issues & Recommendations

Issue #20	Governance can mint/rebase freely
Severity	 GOVERNANCE
Description	The rebase and mint functions can be called by addresses with the MINTER_ROLE. These functions can mint and rebase freely any amount of BTC which can cause the rebasing token total supply and scaling factor to decrease or increase.
Recommendation	Unfortunately we do not see a way for BTC at the current design to combat this. Consider carefully managing the ownership of this contract.
Resolution	 ACKNOWLEDGED

Issue #21	Rebase mechanism is very fragile
Severity	 HIGH SEVERITY
Description	Usually, a rebase token has one sole purpose — to adjust the supply using a special rebase mechanism to maintain price stability. The current design of BTC allows the supply to change without a rebase via several methods, which makes the rebase mechanism very fragile. This can affect the users as they might not be aware of the rebase effects. Paladin did manage to point out a few cases that it can go wrong if an action was performed incorrectly.
Recommendation	We advise the team to reassess the rebase mechanism to making it more restrictive. Usually a rebase token is more restrictive on how the supply changes — more concretely, the supply does not change without actually rebasing. This way, the users are aware of the true value that the token has.
Resolution	 ACKNOWLEDGED

Issue #22	DOMAIN_SEPARATOR is not implemented in BTC.sol permit()
Severity	 HIGH SEVERITY
Description	DOMAIN_SEPARATOR is used to protect against signature replayability. While the current permit() implementation has a nonce variable which prevents the same signature from being used twice on the same contract, it fails to protect users from signature replayability across chains or different contracts. For instance, if a user gives approval using this flawed signature on testnet, the same signature can be stolen and used on mainnet.
Recommendation	Implement DOMAIN_SEPARATOR correctly, which importantly includes chain.id and address(this) in the message digest as stipulated by EIP-712.
Resolution	 ACKNOWLEDGED



Severity

 INFORMATIONAL

Description

`_notEntered` is not used.

—

`uint256(int256(-1))` can be `type(uint256).max`.

—

`_msgSender` and `msg.sender` are used in several instances. Consider using just one format as currently the contract does not support EIP2771.

—

Within the `_claimtokensWRebase` internal function, the `MINTER_ROLE` is checked again. This function is called by `claimtokenRebase` which already has this check.

—

The revert in `updateRebaseRatio` does not have a revert message.

—

The `rebase` public function should use the `_rebase` internal function.

Recommendation Consider fixing the above issues.

Resolution

 ACKNOWLEDGED

2.4 etf

etf implements a bond-style mechanism where users can buy BTC at a discounted price with the BondToken. The contract implements a vesting period, where every bond bought has a vesting period defined by `vestingTime`. Currently, this period is set at 5 days with a maximum of 30 days.

The price per bond is defined by the `bondPrice` variable and can be set to any amount. The bond period can be opened by the team at times they want. They plan to use this at random times to increase treasury assets.

Every wallet can buy a maximum amount of bonds by paying a `maxperTX` amount, which is currently set at 300e18.

Another restriction is to have these periods of buying bonds denoted by the `bondCap` variable which will limit the amounts of bonds bought. Once this limit is reached, the period stops.

2.4.1 Privileged Functions

- `setceil`
- `setbondPrice`
- `setvestingTime`
- `setOpenbond`
- `setMAXtx`
- `recoverTreasuryTokens`
- `transferOwnership`
- `renounceOwnership`

2.4.2 Issues & Recommendations

Issue #24	The bond price can be set to any value
Severity	GOVERNANCE
Description	bondPrice has no restrictions within the setbondPrice function which allows the governance to set a bond price higher than the limit of 1000, which basically causes the user to buy the bonds at a higher price than BTC token, giving them less BTC. This is contrary to the scope of the contract.
Recommendation	Consider adding a maximum cap that is lower than 1000 to avoid accidental price setting which will cause users to lose money when buying bonds. Furthermore, this contract should either be put under a timelock or multi-signature wallet if the timelock cannot be used due to the nature of the strategy used to enable bond buying BTC.
Resolution	ACKNOWLEDGED

Issue #25**Vesting applies to the full bonded amount****Severity** HIGH SEVERITY**Description**

Whenever a user buys or claims a bond, a variable is updated with `block.timestamp` for that specific user. This variable is used to calculate the vesting period for the bonds bought by a user.

If a user buys bonds in period 1 and does not sell but also participates in period 2, the amount that should be already vested from period 1 is also locked and vested in period 2. The same can happen if the user claims in period 1 while they try to buy in period 2.

Recommendation

Consider storing the user's `UserInfo` per epoch, which means each bond-buying period should be defined as the epoch and the user's vesting schedule should be per epoch not per total amount of bonds bought in any period.

This approach will also not push the vesting time for previous periods when a user buys a bond in subsequent periods.

Resolution ACKNOWLEDGED

Team comments: "This is working as intended. We designed it so that everything is revested should users forget to claim their bond and bond again."



Severity



Description

The bondCap restriction is meant to restrict a bond-buying period to a certain amount of bonds that can be bought for that price.

Users who bought bonds in a previous periods can claim their tokens in subsequent periods which will reduce the total supply.

The bondCap restriction takes into account the total supply to calculate how many tokens can be bought.

Example:

- Period 1 starts with 20 bonds opened to be bought at 10%.
- Period 1 finishes, all bonds bought.
- Period 2 starts with 30 bonds at 20% discount.
- All the users claim their previous tokens in this period, reducing the supply with 20, this will expand the bond cap to 50 bonds.

Recommendation

Consider using another variable to track how many bonds can be bought per period. That variable should be reset when a new period starts, and should account for every amount that is bought during that period, ultimately using the bondCap to check against it.

Resolution



Team comments: "We don't really have a fixed period to buy bonds. Bonds are always purchasable as long as there's available bonds. We only add the cap to prevent too many bonds being bought/held at the same time => It is designed to free up when old bonds claim."

Issue #27 Sybil attack is still possible	
Severity	 LOW SEVERITY
Description	The maxperTX variable is supposed to prevent a wallet from buying too many tokens on a single transaction but this mechanism cannot fully protect against Sybil attacks. A wallet can submit multiple transactions with the maximum cap per each transaction. Furthermore, even if a cap is set per wallet, a user split their purchase over multiple wallets.
Recommendation	This issue cannot be circumvented easily — usually a whitelist phase should be performed beforehand with a Sybil-resistance approach to prevent whales from capturing the whole bond-buying period.
Resolution	 ACKNOWLEDGED



Severity

 INFORMATIONAL

Description

The constructor contains a `_mint` that mints 0 tokens.

—

The constructor sets `ceil` to 0, however it is 0 by default.

—

`setceil` should use the internal `_setceil` function.

—

The burn function can be made `internal` and should always burn from `address(this)` instead of using the external call performed like `this.burn(canclaim);`

—

`userInfo[msg.sender]` can be cached in several sections to save gas.

For example, it can be cached within `bond` and `claim` when it is read from storage and then set at the final with the updated values.

```
require(userInfo[msg.sender].totalbonded > 0, "no bond");
uint256 durationPass =
block.timestamp.sub(userInfo[msg.sender].finalInteractionBlock);
```

`userInfo[msg.sender]` can be cached into memory and be used throughout the function then be set at the end of the function to the storage variable.

—

`canclaimTokens` and `remainingVestingTime` can be grouped into one view function that returns how much a user can claim and the period left from the vesting.

Recommendation

Consider implementing the gas optimizations mentioned above.

Resolution

 ACKNOWLEDGED

2.5 aBTC

aBTC is an ERC20 token that acts as a BTC wrapper which does not debase. The functionality of this token is to be used in a pool as part of an LP token to earn fees from the pool.

It contains a mint/redeem fee which can be a maximum of 10%, which is sent to buybackWallet. Currently, the fee is set to 1% for minting and 2% for redeeming.

2.5.1 Privileged Functions

- `setMintFeePercent`
- `setRedeemFeePercent`
- `setBuybackWallet`
- `transferOwnership`
- `renounceOwnership`

2.5.2 Issues & Recommendations

Issue #29	Governance also pays fees for redeeming the collected fees
Severity	 LOW SEVERITY
Description	Whenever a redeem or mint is performed, a certain fee is taken from the amount in form of aBTC. If ever this fee should be transformed in the main currency, BTC, the governance has cannot avoid paying the redemption fee.
Recommendation	Consider if this is the desired behavior, if not, consider adding an <code>onlyOwner</code> function that has no fees on redeem.
Resolution	 ACKNOWLEDGED

Issue #30	Fees are rounded down
Severity	 LOW SEVERITY
Description	Whenever a fee is applied to a user's interaction, the rounding should always be done in favor of the protocol to prevent users from paying less fee than needed. The current implementation on <code>calculateFee</code> rounds down: <pre>return amount * feePercent / 100;</pre>
Recommendation	Consider rounding up for the fee.
Resolution	 ACKNOWLEDGED

Issue #31 Small precision on fees calculation	
Severity	LOW SEVERITY
Description	The fees are calculated with no precision, which means 100 points == 100%. We usually recommend using a higher precision for the fees calculation to be able to set decimal point fees, for example it is currently impossible to set a 0.5% fee.
Recommendation	Consider changing the precision of fees to 100_000 == 100%.
Resolution	ACKNOWLEDGED

Issue #32 Gas optimizations	
Severity	INFORMATIONAL
Description	btcToken should be immutable.
Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	ACKNOWLEDGED



2.6 breth

breth is a wrapper of ETH that sends the ETH to Lido. Users will be able to mint brETH via the mint function which receives ETH and submits it to Lido, transforming it into stETH held by the contract.

Users will be able to redeem brETH at any point in time as the receipt token which is stETH is transferred to the user.

There is a fee applied to mint/redeem which is set to 0.3% on mint and 0.7% on redeem. The maximum fee can be 10% and can be set by the owner.

All fees and accrued value of stETH inside the breth contract is calculated as protocol profit. The fees are sent to the buybackWallet address. The profits accrued from holding stETH in the breth contract are sent to the owner of the contract.

2.6.1 Privileged Functions

- `claimProtocolProfits`
- `setMintFeePercent`
- `setRedeemFeePercent`
- `setBuybackWallet`
- `transferOwnership`
- `renounceOwnership`

2.6.2 Issues & Recommendations

Issue #33	Protocol can be insolvent during a LIDO slashing event
Severity	 HIGH SEVERITY
Description	breth seems to assume that stETH can only rebase upwards. During a slashing event, stETH can rebase downwards. When that happens, 1 breth is still redeemable for 1 stETH even though the contract does not have enough stETH to compensate for all breth. This will lead to a bankrun situation where users who redeem last will be holding worthless breth since they can no longer redeem their tokens as protocol has run out of stETH. <u>PoC:</u> - Alice deposits 500 ETH to mint 500 breth, Bob deposits 500 ETH to mint 500 breth. The protocol deposits the 1000 ETH to mint 1000 stETH. - LIDO validators get slashed by 10%, and stETH rebases down to 900 stETH. - Alice sees this and quickly redeems all her breth for stETH, resulting in 0 losses (assuming no fees) for her. With only 400 stETH left in the contract, Bob can only redeem 400 breth, emptying the contract. Bob is now holding 100 worthless breth.
Recommendation	Within <code>redeem()</code>, check that the stETH balance of the contract is more than the total supply of minted breth. If the balance of stETH is lower, it means a slashing event has occurred and users should not be allowed to redeem until yield from future staking rewards has increased enough such that $\text{balance of stETH} > \text{breth}$. Alternatively, users can be given the option withdraw their breth but the amount should be proportionally reduced according to the ratio of stETH and breth. Another option from the protocol side is to use stETH from the treasury to cover the slashing losses.
Resolution	 RESOLVED
	redeem is only allowed if the protocol is in profit. However, this can prevent users from redeeming their breth during a slashing event.

Issue #34	Fees are rounded down
------------------	------------------------------

Severity LOW SEVERITY**Description**

Whenever a fee is applied towards a user's interaction, the rounding should always be done in favor of the protocol to prevent users from paying less fees than needed. The current implementation on `calculateFee` rounds down:

```
return amount * feePercent / 10000;
```

Recommendation

Consider rounding up for the fee.

Resolution ACKNOWLEDGED

Issue #35	stETH could be lost when redeeming with a contract
------------------	---

Severity LOW SEVERITY**Description**

Users mint breth with ETH, but when redeeming, they receive proceeds in stETH. If the receiving address is a contract, it may not be equipped to handle stETH since it requires implementation of redeeming from LIDO and converting it to native ETH.

Recommendation

Consider checking that the receiving address is not a smart contract. Alternatively, it should be highlighted explicitly that users will receive stETH when they redeem their breth and not ETH.

Resolution ACKNOWLEDGED

Issue #36 Future airdrops can be stuck in the contract	
Severity	LOW SEVERITY
Description	breth would be holding a large number of stETH as users deposit ETH to convert into stETH and they will be left in the contract. It is possible that future airdrops will give rewards/tokens to large holders of stETH, but such rewards cannot be claimed by the contract since there is no way to recover them.
Recommendation	Consider adding a recoverERC20 function that allows the protocol to transfer any ERC20 other than stETH.
Resolution	ACKNOWLEDGED

Issue #37 Gas optimizations	
Severity	INFORMATIONAL
Description	The receive function can be removed — if someone tries to send ETH to the contract, it will fail anyway if this function is not present. — stETH and lido variables should be immutable.
Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	ACKNOWLEDGED



PALADIN
BLOCKCHAIN SECURITY