GA GUARDIAN

Bracket

LST Vault

Security Assessment

January 20th, 2025

Summary

Audit Firm Guardian

Prepared By Owen Thurm, Daniel Gelfand, Robert Rodriguez,

Wafflemakr, 0xCiphky

Client Firm Bracket

Final Report Date January 20, 2025

Audit Summary

Bracket engaged Guardian to review the security of their LST management system. From the 18th of December to the 23rd of December, a team of 5 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Issues Detected Throughout the engagement 4 High/Critical issues were uncovered and promptly remediated by the Bracket team.

Security Recommendation Given the number of High and Critical issues detected as well as additional code changes made after the main review, Guardian recommends that an independent security review of the protocol at a finalized frozen commit is conducted before deployment.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/bracket-1

Table of Contents

Project Overview

Project Summary

Project Name	Bracket
Language	Solidity
Codebase	https://github.com/bracket-fi/core-contracts
Commit(s)	727a7751260e155a88d539bc1af5a195a3c7da83

Audit Summary

Delivery Date	January 20, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	3	0	0	0	0	3
High	1	0	0	0	0	1
Medium	7	1	0	1	0	5
Low	17	0	0	4	0	13

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	🔴 Critical	🟠 High	🟡 Medium
Likelihood: *Medium*	🟠 High	🟡 Medium	🟢 Low
Likelihood: *Low*	🟡 Medium	🟢 Low	🟢 Low

Impact

High Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.

Medium A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.

Low Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.

Medium An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.

Low Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	Incorrect Amount Of Collateral Returned	Logical Error	Critical	Resolved
C-02	Inaccurate Share Calculation On Mint	Logical Error	Critical	Resolved
C-03	Incorrect Collateral Calculation During Burn	Logical Error	Critical	Resolved
H-01	Missing lastNavUpdate Update	Logical Error	High	Resolved
M-01	Incomplete Check In calculateMint Function	Validation	Medium	Resolved
M-02	Decreased NAV Frontrunning	Frontrunning	Medium	Resolved
M-03	Manager Avoids Negative Performance Fees	Unexpected behavior	Medium	Pending
M-04	Wrong Index For Whitelist/Blacklist	Logical Error	Medium	Resolved
M-05	Incorrect Withdrawable Assets	Logical Error	Medium	Resolved
M-06	totalValue DoS	DoS	Medium	Acknowledged
M-07	Manager Fees Withdrawn By Users	Logical Error	Medium	Resolved
L-01	Swap Can Have Equal In And Out Tokens	Validation	Low	Resolved
L-02	Misvaluation Due To stETH-ETH Peg Assumption	Oracles	Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-03	Fees Unavailable For Claim	Logical Error	Low	Acknowledged
L-04	Lack Of Slippage Control	Slipagge	Low	Resolved
L-05	Unused Params	Optimization	Low	Resolved
L-06	Increased Gas Cost For Multiple Collaterals	Configuration	Low	Acknowledged
L-07	Adding Collateral Without Oracle Support	Validation	Low	Resolved
L-08	Vault Does Not Validate Zero Amounts	Validation	Low	Resolved
L-09	Missing Admin Functions	Configuration	Low	Resolved
L-10	Unlicensed Smart Contracts	Best practices	Low	Resolved
L-11	balanceOf Includes Pending Amounts	Unexpected behavior	Low	Acknowledged
L-12	Vanity Nav Updated In Epoch 0	Validation	Low	Resolved
L-13	Lacking Pause Mechanism	Best practices	Low	Resolved
L-14	Unnecessary Address(0) Check	Optimization	Low	Resolved
L-15	Excessive Wait For Manager Withdrawals	Logical Error	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-16	Collateral DoS Risk	DoS	Low	Resolved
L-17	Lacking Slippage Check Allows Griefing	Griefing	Low	Resolved

C-01 | Incorrect Amount Of Collateral Returned

Category	Severity	Location	Status
Logical Error	● Critical	BrktETH.sol	Resolved

Description PoC

The existing `calculateBurn` logic incorrectly calculates how much of the `token` should be returned to the user for the given amount of brktETH value.

For example:
(1) User deposits 10 WSTETH and receives 10 brktETH. Assume this is the entire brktETH supply.
(2) ETH rate of WSTETH goes from 1 ETH to 1.1 ETH.
(3) User burns all 10 brktETH shares, but does not receive 10 WSTETH. Instead they get (1.1 ether * 11 ether / 1 ether) which is 12.1 WSTETH, more than the 10 WSTETH they put in.

This can be extremely detrimental to protocol as a user may extract more funds than appropriate, as well as prevent a depositor from withdrawing all their shares.

Furthermore, this issue will occur every single time `burn` is called. To accurately calculate how much `token` should be returned, instead the ETH value of the redeemed brktETH should be divided by the ETH value of 1 `token`.

Recommendation

Change the calculation to `return Math.mulDiv(value, 1 ether, oracle.getRate(token));`

Resolution

Bracket Team: The issue was resolved in commit 4efa7d7.

C-02 | Inaccurate Share Calculation On Mint

Category	Severity	Location	Status
Logical Error	● Critical	BrktETH.sol: 64	Resolved

Description PoC

The `mint` function in the BrktETH contract currently deposits tokens into the contract—updating the token's `totalDeposit`—before calculating the amount to mint.

As a result, the user's newly deposited tokens are included in the total value during the share calculation, causing the user to receive fewer shares than intended.

For example:
(1) Alice deposits 10 WSTETH and mints 10 brktETH
(2) Bob deposits 10 WSTETH directly after, and mints 5 brktETH (10 brktETH * 10 ETH / 20 ETH), although he should receive the same amount of brktETH as Alice due to 50-50% supply of the pool.

Consequently, this causes loss of assets for the depositor as they receive less shares than necessary, and will occur each time after the first mint.

Recommendation

Modify the mint function so that it calculates the amount of tokens to be minted before depositing them into the contract.

Resolution

Bracket Team: The issue was resolved in commit 4a2dc96.

C-03 | Incorrect Collateral Calculation During Burn

Category	Severity	Location	Status
Logical Error	● Critical	BrktETH.sol: 79	Resolved

Description PoC

The burn function currently burns the user's brktETH tokens before calculating the amount of collateral they are entitled to.

By doing so, the calculation for `colAmount` takes place after the user's share has already been removed from the supply, which may cause them to receive more collateral than they should or end up with nothing at all if they are the only remaining shareholder.

Recommendation

Modify the burn function to calculate the collateral amount before burning the user's brktETH.

Resolution

Bracket Team: The issue was resolved in commit 4a2dc96.

H-01 | Missing lastNavUpdate Update

Category	Severity	Location	Status
Logical Error	High	BracketVault.sol: 117	Resolved

Description

In the `updateNav` function there is no update to the `lastNavUpdate` variable, therefore the confirmation period validation will never apply past the initial 1 day period after the vault starts.

Recommendation

Update the `lastNavUpdate` variable in the `updateNav` function

Resolution

Bracket Team: The issue was resolved in commit cce9ce5.

M-01 | Incomplete Check In calculateMint Function

Category	Severity	Location	Status
Validation	Medium	BrktETH.sol: 242	Resolved

Description

The current `calculateMint` implementation assumes that if `totalSupply` is zero, there is no existing pool of assets and thus sets `brktAmount = value`:

```
function calculateMint(uint256 value)

     public view returns (uint256 brktAmount) {uint256 supply = totalSupply(); if (supply = 0) {brktAmount = Math.mulDiv(supply, value, getTotalValue());} else {brktAmount = value;}}
```

However, in a very specific edge case scenario, it could be possible that `totalSupply` is non-zero but `getTotalValue` is actually zero, for example, due to a sudden asset devaluation.

In that case, a division by zero would occur blocking any new deposits (once the issue where you deposit before calculating the minted `brktAmount` is corrected).

Recommendation

Update the `calculateMint` function as shown below:

```
function calculateMint(uint256 value)

     public view returns (uint256 brktAmount) {uint256 supply = totalSupply(); uint256 totalValue = getTotalValue(); if ((supply = 0) & (totalValue = 0)) {brktAmount = Math.mulDiv(supply, value, totalValue);} else {brktAmount = value;}}
```

Resolution

Bracket Team: The issue was resolved in commit a7ed83f.

M-02 | Decreased NAV Frontrunning

Category	Severity	Location	Status
Frontrunning	Medium	BracketVault.sol	Resolved

Description

If the `NAV_UPDATER` calls `updateNav` with a `newNav` that is lower than the previous one, a user can frontrun the update and trigger a withdrawal.

If the delay is only 1 epoch, then the user's shares will be valued at the previous epoch's NAV which is higher, withdrawing more assets and avoiding the loss.

Recommendation

Consider enforcing that the delay is greater than 1 epoch, otherwise clearly document this risk and using a private RPC.

Resolution

Bracket Team: The issue was resolved in commit 90315e6.

M-03 | Manager Avoids Negative Performance Fees

Category	Severity	Location	Status
Unexpected behavior	Medium	BracketVault.sol: 175	Pending

Description

The `claimManagerPerformanceFees` function may be called once every 90 days and each time the `accruedManagerPerformanceFees` are reset to zero.

This way a manager does not have to continue to pay down a large negative performance fee if it has accrued and has been being paid off for a long duration of time.

However if a manager submits an `updateNav` call with a negative performance fee directly before the 90 period is over they can almost immediately avoid the negative performance fee by calling the `claimManagerPerformanceFees` function again, whether on purpose or by accident.

Recommendation

Consider using a separate interval for clearing negative performance fees which resets every time a new negative performance fee is added.

Resolution

Bracket Team: Pending.

M-04 | Wrong Index For Whitelist/Blacklist

Category	Severity	Location	Status
Logical Error	Medium	BrktEth.sol	Resolved

Description

Within functions `whitelistCollateral` and `blacklistCollateral` the index of the token is retrieved to whitelist/blacklist the token respectively. The issue is that the `collateralsIndex` starts from 1, so the whitelist/blacklist will be set for the wrong token: `uint256 index = collateralsIndex[token];`

Recommendation

Use function `_getIndex` instead since it subtracts the returned index by 1.

Resolution

Bracket Team: The issue was resolved in commit 6853354.

M-05 | Incorrect Withdrawable Assets

Category	Severity	Location	Status
Logical Error	Medium	BracketVault.sol: 293	Resolved

Description PoC

The BracketVault.withdrawableAssets() should allow users to query the max amount of assets they can withdraw from the vault.

However, this function will incorrectly subtract pending deposit shares when the lastDeposit has occurred in a previous epoch.

The assets for a lastDeposit that occurred in a previous epoch should be represented as withdrawable though since they will be minted to the user in the withdraw function.

Recommendation

Modify the withdrawableAssets view function such that the pendingDepositShares are only removed from the withdrawable amount when the epoch of the lastDeposit is the same as the current epoch:

```
function withdrawableAssets(address account)

    external view returns (uint256) {uint256 lastNav = epoch = 0 ; 1e18 :
navs[epoch - 1]; Deposit memory _deposit = lastDeposit[user]; if
(_deposit.epoch == epoch) return convertToAssets(sharesOf(account) -
convertToShares(_deposit.assets, vanityNav), lastNav); else return
convertToAssets(sharesOf(account), lastNav);}
```

Resolution

Bracket Team: The issue was resolved in commit bf3f70b.

M-06 | totalValue DoS

Category	Severity	Location	Status
DoS	Medium	BracketOracle.sol: 46	Acknowledged

Description

The BrktETH contract relies on the `getTotalValue` function to determine the total value of all collateral denominated in ETH, which is critical for functions like minting and burning. The process involves fetching each collateral's value and summing the results.

However, if any one token's price retrieval fails, the entire `getTotalValue` function reverts—causing all dependent operations (e.g., minting, burning) to fail as well.

For instance, the EZETH token uses an external oracle that performs certain validations when fetching its price, which could revert if those validations fail. This creates a single point of failure that disrupts the contract's functionality whenever a single token's price feed encounters an issue.

Recommendation

Implement a fallback oracle for each token so that if the primary method reverts (e.g., due to external validation errors), the contract can still fetch the token's price.

This prevents the entire `getTotalValue` function—and consequently critical operations like minting or burning—from halting when a single price feed fails.

Resolution

Bracket Team: Acknowledged.

M-07 | Manager Fees Withdrawn By Users

Category	Severity	Location	Status
Logical Error	Medium	BracketVault.sol: 260	Resolved

Description

In the `_processDepositsWithdrawals` function the available balance for withdrawal by the users is based on the `_getManagerAvailableBalance`, which does not set aside the `accruedManagerPerformanceFees` (if positive), `accruedManagerTvlFees`, and `accruedBrktTvlFees`.

If a manager has not claimed these fees in a significant amount of time, but then submits an `updateNav` where they cannot fully cover all withdrawals for this epoch then the full manager balance or approval amount is used up.

The fee amounts will still be tracked in the `accruedBrktTvlFees`, `accruedManagerTvlFees`, and `accruedManagerPerformanceFees` variables, but the manager will not be able to claim these amounts as the brktEth and brktEth approval has been removed from the manager address.

Recommendation

Consider reducing the available amount by the `accruedManagerPerformanceFees` (if positive), `accruedManagerTvlFees`, and `accruedBrktTvlFees` in the `_processDepositsWithdrawals` function.

Resolution

Bracket Team: The issue was resolved in commit ffc20bb.

L-01 | Swap Can Have Equal In And Out Tokens

Category	Severity	Location	Status
Validation	Low	BrktETH.sol: 155	Resolved

Description

Function `swapCollateral` lacks validation that `tokenIn = tokenOut`. In such a case, there could be an early return not to waste gas on calculating the rebalance and transfers.

Recommendation

Early return if `tokenIn = tokenOut`

Resolution

Bracket Team: The issue was resolved in commit c78455d.

L-02 | Misvaluation Due To stETH-ETH Peg Assumption

Category	Severity	Location	Status
Oracles	Low	BracketOracle.sol: 58	Acknowledged

Description

The `_getWstethRate` function calls the `stEthPerToken()` function in the WSTETH contract, which returns the amount of stETH per wstETH.

This value is then treated as if stETH were pegged 1:1 to ETH. However, stETH can and has previously depegged from ETH (Ref), making this assumption unreliable.

Recommendation

Use a reliable price feed, such as Chainlink's stETH-ETH feed (Link), to accurately determine the ETH value instead of assuming a one-to-one peg.

Resolution

Bracket Team: Acknowledged.

L-03 | Fees Unavailable For Claim

Category	Severity	Location	Status
Logical Error	Low	BracketVault.sol	Acknowledged

Description

Over time fees are accrued within the BracketVault: accruedManagerPerformanceFees, accruedManagerTvlFees, and accruedBrktTvlFees.

However, there is no guarantee that there is sufficient brktETH balance for the fees to be claimed because user withdrawals may decrease the manager's balance.

For example:
1. Nav is 1 ether,
2. Alice deposits 10 brktETH and gets 10 shares. She is the sole depositor.
3. Nav increases to 2 ether, with >0 fees accumulated.
4. Alice burns 10 shares to get 20 brktETH, which comes from the manager's balances.
5. Fee claim is attempted but manager does not have sufficient balance for the fee and causes a ERC20InsufficientBalance revert.

Recommendation

Ensure managers are aware to set aside brktETH for fees.

Resolution

Bracket Team: Acknowledged.

L-04 | Lack Of Slippage Control

Category	Severity	Location	Status
Slipagge	Low	BrktETH.sol: 64	Resolved

Description

The mint and burn functions in the BrktETH contract rely on the vault's total collateral value to determine the number of shares or collateral token a user will receive. Because the vault uses tokens that can change in value at any time—for example, through rebases or slashings.

This can lead to unexpected outcomes, such as the user ending up with fewer shares minted, or returning a different ratio of collateral when burning tokens.

Recommendation

Implement slippage control that allows users to specify acceptable thresholds.

Resolution

Bracket Team: The issue was resolved in commit b2c2c4a.

L-05 | Unused Params

Category	Severity	Location	Status
Optimization	Low	BracketVault.sol	Resolved

Description

The following param/error is not used in `BracketVault` contract:
• `feeClaimer` state variable
• `DepositIsCurrentEpoch` error

Recommendation

Remove unused code or consider adding it to the current implementation.

Resolution

Bracket Team: The issue was resolved in commit 8637d47.

L-06 | Increased Gas Cost For Multiple Collaterals

Category	Severity	Location	Status
Configuration	Low	BrktETH.sol: 224	Acknowledged

Description

The `BracketOracle` currently supports 10 collateral tokens. In case all 10 are added and deposits are non zero, the `getTotalValue` function will need to iterate through every token to fetch the rate, adding around 270,000 gas, with Renzo's Staked ETH rate being the most gas intensive.

As a result, main actions will have an increased gas consumption:
• burn : 325671
• mint : 362535

As the protocol is deployed on the Ethereum mainnet , this gas cost can become a barrier for users to invest in the `BrktEth` vault.

Recommendation

Consider this scenario when adding supported collaterals.

Resolution

Bracket Team: Acknowledged.

L-07 | Adding Collateral Without Oracle Support

Category	Severity	Location	Status
Validation	Low	BrktETH.sol: 203	Resolved

Description

There are ten collaterals currently supported in `BracketOracle`, but `BrktETH.addCollateral` does not check if the collateral being added is currently supported. As the `addCollateral` is only called once, it will be wise to make a call to `getRate(token)` to ensure its supported.

In the future, when new collaterals are added (i.e. apxETH, ETHx), this check will ensure `BracketOracle` is updated first.

Recommendation

When adding a new collateral, consider executing `getRate(token)` to verify if the oracle supports it.

Resolution

Bracket Team: The issue was resolved in commit c7c0423.

L-08 | Vault Does Not Validate Zero Amounts

Category	Severity	Location	Status
Validation	Low	BracketVault.sol: 198	Resolved

Description

During `deposit` and `withdraw`, the `assets` param is not validated, so it can be 0. Although there is no major impact, the function does not revert and events are emitted, potentially causing issues in the UI.

Recommendation

Consider validating for zero asset amount during `deposit` and `withdraw`.

Resolution

Bracket Team: The issue was resolved in commit 50e4814.

L-09 | Missing Admin Functions

Category	Severity	Location	Status
Configuration	Low	BracketVault.sol: 90	Resolved

Description

The `manager` address is set during vault initialization. However, if there is an issue with manager or it's compromised, there is no admin function to update this address. Similarly, `withdrawal` delay is set at initialization but can't be updated again.

Recommendation

Consider adding admin functions to update `manager` and `withdrawalDelay`.

Resolution

Bracket Team: The issue was resolved in commit c152859.

L-10 | Unlicensed Smart Contracts

Category	Severity	Location	Status
Best practices	Low	Global	Resolved

Description

The `BracketVault`, `BracketOracle` and `BrktETH`, contracts are currently marked as unlicensed, as indicated by the SPDX license identifier at the top of the file: `SPDX-License-Identifier: UNLICENSED`

Using unlicensed contracts can lead to legal uncertainties and conflicts regarding the usage, modification and distribution rights of the code.

Recommendation

It is recommended to choose and apply an appropriate open-source license to the smart contract.

Some options are:
1. MIT License: A permissive license that allows for reuse with minimal restrictions.
2. GNU General Public License (GPL): A copyleft license that ensures derivative works are also open-source.
3. Apache License 2.0: A permissive license that provides an express grant of patent rights from contributors to users.

Resolution

Bracket Team: Resolved.

L-11 | balanceOf Includes Pending Amounts

Category	Severity	Location	Status
Unexpected behavior	Low	RebasingToken.sol: 17, 21	Acknowledged

Description

The `balanceOf` function relies on the `sharesOf` to determine the balance of a user.

The `sharesOf` function includes the `pendingDepositShares` which can include funds that the user has not yet been minted from the `lastDeposit` and may not be able to presently mint in the case where the `lastDeposit.epoch = epoch`.

As a result the `balanceOf` and `sharesOf` function reflects funds that are not available to the user and may lead to confusion for users and integrators.

Recommendation

Be aware of this behavior, if it is expected then be sure to document this clearly for users and integrators.

Resolution

Bracket Team: Acknowledged.

L-12 | Vanity Nav Updated In Epoch 0

Category	Severity	Location	Status
Validation	Low	BracketVault.sol: 154	Resolved

Description

The `updateVanityNav` function may be called to set the `vanityNav` to something other than 1e18 before the vault has been started with the `startVault` function. This may lead to unexpected behavior and should not be a supported interaction.

Recommendation

Consider validating that the epoch is greater than 0 in the `updateVanityNav` function.

Resolution

Bracket Team: The issue was resolved in commit 664a697.

L-13 | Lacking Pause Mechanism

Category	Severity	Location	Status
Best practices	Low	BrktETH.sol	Resolved

Description

The owner of `BrktETH` is able to blacklist collaterals, preventing any further deposits. However, this does not prevent collateral withdrawals. If one of the external protocols is compromised, it will create a bank run, as users will try to withdraw the other collaterals not affected.

In this case, a pausing mechanism will help to address the situation and implement the appropriate fixes. Pausing the protocol can also allow owner to prevent user actions during an upgrade or fix.

Recommendation

Consider adding a pausing mechanism, inheriting from OZ `PausableUpgradeable` and implement the admin functions `pause` and `unpause`.

Resolution

Bracket Team: The issue was resolved in commit 003f154.

L-14 | Unnecessary Address(0) Check

Category	Severity	Location	Status
Optimization	Low	BrktEth.sol: 53	Resolved

Description

In the `initialize` function there is an address 0 check against all entries of the `tokens` array, however the `addCollateral` function already implements such a check.

Recommendation

Remove the redundant address zero check in the `initialize` function for the `tokens` entries.

Resolution

Bracket Team: The issue was resolved in commit c519f90.

L-15 | Excessive Wait For Manager Withdrawals

Category	Severity	Location	Status
Logical Error	Low	BrktETH.sol: 23	Resolved

Description

The `manager` will use `brktETH` from vault depositors for certain investment strategies. Therefore, the `brktETH` tokens will need to be burned to extract the underlying collateral.

The issue relies on the withdrawal mechanism, as users will need to wait 5 days to be able to claim the collateral. This creates a barrier for the manager and affects investments during this waiting period.

Recommendation

Consider adding an exception for managers, either receiving the collateral immediately or reduce the waiting period.

Resolution

Bracket Team: The issue was resolved in commit dcf04df.

L-16 | Collateral DoS Risk

Category	Severity	Location	Status
DoS	Low	BrktEth.sol: 203	Resolved

Description

The `addCollateral` function does not validate a maximum number of collateral tokens that can be added to the array.

As a result the owner may on accident add too many collateral tokens over time which must all be looped over in the `getTotalValue` function. This may cause operations to require more than the block gas limit or at least be quite expensive to operate.

Recommendation

Consider adding a validation against a maximum number of supported collaterals. Additionally, consider implementing functionality to be able to remove collateral tokens from the `collaterals` list when they are blacklisted and have no deposits so that this does not become an issue over time.

Resolution

Bracket Team: The issue was resolved in commit 1bdc15e.

L-17 | Lacking Slippage Check Allows Griefing

Category	Severity	Location	Status
Griefing	Low	brktEth.sol: 155	Resolved

Description

In the `swapCollateral` function there is no way for the caller to specify the worst rate they are willing to accept for a swap.

There is no slippage at the bracket level, however the rate reported by some tokens may be manipulated by a malicious actor who wishes to grief the owner who is making the swap through bracket.

For example, the Renzo EZEth price may be manipulated by force sending Ether to the `depositQueue` and `withdrawQueue` address.

Recommendation

Consider adding a `maxAmountIn` parameter to the `swapCollateral` function to allow the caller to protect themselves from any potential griefing attacks that may apply to any arbitrary collateral token.

Resolution

Bracket Team: The issue was resolved in commit d016167.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits