ChainAudits

Smart Contract Audit

Security Assessment

20. 08. 2024

BaseBrosFi

Project Details

CODEBASE	NETWORK	LANGUAGE
Not deployed	BASE CHAIN	Solidity
NAME	**TOTAL SUPPLY**	**SYMBOL**
BaseBrosFi	N/A	N/A
WEBSITE	**UNIT TESTS**	**FORK**
basebros.fi	Not Provided	Beefy Finance

ABOUT THE PROJECT

The ecosystems surrounding L1 and L2 networks, and other chains are unable to interact with each other efficiently, resulting in poor user and development experiences alongside significant friction such as requiring the knowledge to know which bridge offers the best experience for any specific chain. Base Bros aim to revolutionize this landscape by developing a solution that will create a much smoother, frictionless and intuitive user experience especially for users new to DeFi.

COMMIT	CODEBASE LINK
71b5dc3	https://github.com/BaseBrosFi/brewerycontracts/tree/main

Social Media Links

Telegram	https://t.me/basebros_fi
Twitter	https://x.com/BaseBrosFi
Facebook	N/A
Instagram	N/A
Github	N/A
Reddit	N/A
Medium	https://medium.com/@basebros
Discord	N/A
Youtube	N/A
TikTok	N/A
LinkedIn	N/A

Version	Delivery Date	Changelog
v1.0	20. August 2024	• Layout Project • Automated- /Manual-Security Testing • Summary

Vulnerability Summary

Critical	Critical risks are those that affect the platform's safe operation and must be resolved before launch. Users should avoid investing in any project with unresolved critical risks.
Major	Major risks include centralization issues and logical errors. These risks can potentially result in the loss of funds or control over the project under certain conditions.
Medium	Medium risks might not directly threaten users' funds, but they can impact the platform's overall functionality.
Minor	Minor risks are similar to the above categories but on a smaller scale. They typically do not compromise the project's overall integrity but may lead to less efficient solutions.
Informational	Informational errors are recommendations aimed at improving code style or aligning operations with industry best practices. These usually do not affect the code's overall functionality.

Note - The provided audit report thoroughly examines the security aspects of the smart contract employed in the project, encompassing potential malicious manipulation of the contract's functions from external sources. However, it's important to note that this analysis does not incorporate functional or unit testing of the contract's logic. Therefore, we cannot ensure the absolute correctness of the contract's logic, including internal calculations within the formulae utilised in the contract.

Overview

In-Scope Files

The team provided us with files to review during security audits. This audit covered the following files listed below with their respective SHA-1 Hash.

SHA-1	File
src/interfaces/common/ IVelodromeGauge.sol	bf8733c5c5ef44c0fab16a60432a0bcab02cd73b
src/interfaces/common/ IFeeConfig.sol	f8124ef60f121ea8950463a3ddcae6c8c6981856
src/interfaces/common/ ISolidlyRouter.sol	127496e3d2441a7a0114b697c11585a1b036d98f
src/interfaces/common/ ISolidlyPair.sol	df57de0a8de60cfc1d14ad4650ed04caaf1d408d
src/interfaces/common/ IERC20Extended.sol	85a8fc12757b5e073d062275456f0f6c92d3f840
src/interfaces/IStrategy.sol	8e169a992ad42e5e265f54b138c1b7713f65f107
src/brewery/Brewery.sol	588d39cc32c811a3c14387e76dd57b7e1fbeec58
src/strategies/Staking/Staking.sol	fb54baf36ea7429b25604ad809bcbf1da7861c00
src/strategies/Common/ StratFeeManagerInitializable.sol	fc859410835d7f9bc7b2dcc7c3ad08dbc858cb3c
src/strategies/Aerodrome/ StrategyVelodromeGaugeV2.sol	a131f4a9f29d397409f849d1830847fa2ca07020

Please note that files with hash values different from those listed in this table have been changed after security checks, intentionally or unintentionally, as a particular hash value may indicate a changed state or potential vulnerabilities not checked in this scan.

Note for Investors: We only examined agreements indicated in the indicated ratings. No contracts associated with the project beyond this range have been audited, therefore, we cannot provide insight or assume responsibility for their security

Imported packages

Used code from other Frameworks/Smart Contracts (direct imports).

Dependency/Import Path	Count
@openzeppelin-4/contracts/access/Ownable.sol	1
@openzeppelin-4/contracts/token/ERC20/ERC20.sol	2
@openzeppelin-4/contracts/token/ERC20/utils/SafeERC20.sol	1
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	2
@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol	1
@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol	1
@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol	1
@openzeppelin-4/contracts/access/Ownable.sol	1

External/Public functions

External/public functions can be invoked outside of the contract, i.e., accessed by other contracts or external accounts on the blockchain. These functions are identified using external or public visibility modifiers in the function declaration.

State variables

State variables are stored on the blockchain as part of the contract conditions. They are declared at the contract level and can be accessed and changed by any action in the contract (except with modifiers like onlyOwner, etc.). State transitions can be described using a visibility modifier, such as public, private, or internal, which refers to the access to the transition.

Components

Contracts	4
Libraries	0
Interfaces	6
Abstract	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	102
Payable	1
External	76
Internal	78
Private	0
Pure	1
View	45

State Variables

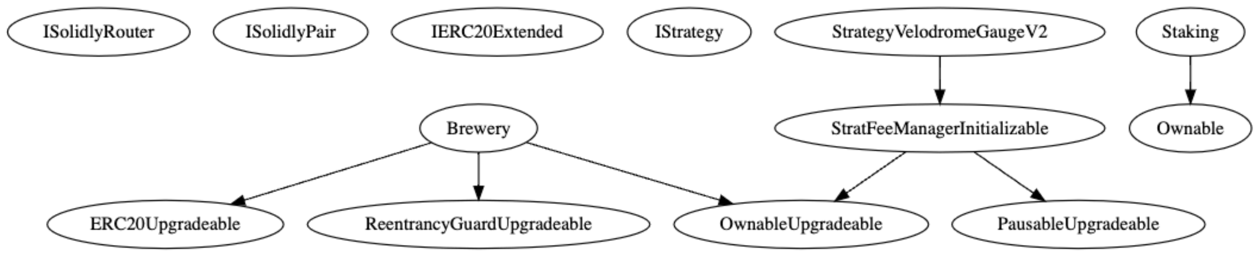
Total	39
Public	37

Capabilities

Solidity Versions observed	>=0.6.0 <0.9.0 , ^0.8.0
Transfers ETH	Yes
Can Receive Funds	Yes
Uses Assembly	No
Has Destroyable Contracts	No

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

Strategies

While our projects undergo rigorous testing to address all known vulnerabilities, it's important to provide further clarification and understanding by outlining specific vulnerabilities that have been identified. This additional information enhances transparency and helps stakeholders comprehend the security measures taken and any potential risks mitigated.

Title	Issue
Re-entrancy	Improper Enforcement of Behavioral workflow
Unexpected Ether Balance	Improper Locking
Code With No Effects	Irrelevant Code
Flash Loan Attacks	External Oracle Manipulation
Snadwich Attacks	A Form of Front Running
Write to Arbitrary Storage Location	Improper Write to Arbitrary Storage Location
Variable Shadowing	Improper Coding Standards
Unprotected SELF-DESTRUCT	Improper Access Control
Potential Honeypot	Improper Function
Unprotected ETH withdrawal	Improper Access Control
Outdated Compiler Version	Using components with Known vulnerabilities
Weak Sources of Randomness from Chain Attributes	Use of Insufficiently Random values
Unsafe use of libraries	Improper Implementation
Wrong Implementation of Token Standards	Improper Coding Standards

Auditing Strategy and Techniques Applied

All Projects at ChainAudits undergo our in-house developed "4-Eye" method. This process ensures that the code is analyzed not by a single auditor but by our entire technical team. Moreover, this is accomplished most efficiently, leaving no stone unturned.

We manually check every file, line by line. Automated tools are used solely to help us achieve faster and better results.

Code Analysis Methodology

The auditing process follows a routine series of steps:

- Manual Code Review:
    - Evaluate the overall structure and organization of the smart contract code line-by-line.
    - Review the implementation of access control mechanisms to ensure that sensitive functions are appropriately restricted to authorized users.
    - Ensure that critical vulnerabilities are adequately tested and that edge cases and boundary conditions are covered.
- Static Code Analysis:
    - Utilise static analysis tools to scan the codebase for vulnerabilities and security issues.
    - Identify common vulnerabilities like reentrancy, integer overflow/underflow, and unchecked external calls.
    - Evaluate compliance with coding standards and best practices, ensuring adherence to security guidelines.
- Code Structure and Architecture Review:
    - Analyze the overall structure and architecture of the smart contract code.
    - Assess the modularity, readability, and maintainability of the code.
    - Review the separation of concerns and adherence to design patterns for robustness and security.
- Security Best Practices Evaluation:
    - Evaluate the implementation of security best practices, including access control mechanisms and input validation.
    - Verify proper error handling to mitigate potential vulnerabilities and ensure contract robustness.

  - Check for gas optimization techniques to enhance efficiency and reduce transaction costs.
- External Dependency Assessment:
  - Assess the integration with external contracts, libraries, or services.
  - Review the security of external dependencies and their impact on the overall security of the smart contract codebase.
  - Ensure secure interactions with external components to prevent vulnerabilities and attack vectors.
- Post-Analysis Support:
  - Offer support and guidance to assist the development team in addressing identified vulnerabilities.
  - Collaborate with stakeholders to ensure effective implementation of recommended security enhancements.
  - Provide ongoing assistance and consultation to maintain the security of the smart contract codebase.

Discovered Issues

Privileges

Centralization emerges when one or multiple entities possess privileged access or authority over a smart contract's functionalities, data, or decision-making processes. This situation may arise if a singular entity exercises complete control over the contract or if certain participants hold unique permissions or capabilities inaccessible to others within the ecosystem.

In the project, some authorities have access to the following functions:

No.:	File	Privileges
1	Brewery.sol	• Propose Strategy Implementation • Upgrade Strategy • Withdraw foreign tokens from the contract • Set Reward Per Block and reward blocks
2	StrategyVelodromeGauge V2	• Enable/Disable harvest on Deposit • Pause deposits and withdraw all funds from the contract
3	StratFeeManagerInitializa ble	• Set Strategy FeeId • Set withdrawal fee • Set Vault and router Address • Set Keeper Address • Set Beefy Config address
4	Staking.sol	• Set Reward per block

Recommendations

To mitigate potential hacking risks, the team must meticulously handle the private key associated with the privileged account. Moreover, we advise bolstering the security measures surrounding centralized privileges or roles within the protocol by implementing decentralized mechanisms or employing smart-contract-based accounts like multi-signature wallets. This approach can enhance security by distributing control among multiple parties and requiring consensus for executing critical actions, thereby reducing the likelihood of unauthorized access or malicious activity.

Here are some suggestions of what the project owners can do:

- **Adopting multi-signature wallets**: Utilize wallets like Gnosis Safe that mandate approval from multiple parties before executing transactions. This additional layer of security helps safeguard against unauthorized actions.
- **Implementing a timelock**: Introduce a timelock feature with a delay of, say, 48-72 hours for sensitive operations. This delay period ensures stakeholders have ample time to review and respond to proposed changes, reducing the risk of impulsive or unauthorized modifications.

- **Incorporating a DAO/Governance/Voting module**: Integrating a decentralized governance system enhances transparency and community involvement. This empowers users to participate in decision-making processes, fostering a sense of ownership and accountability within the community.
- **Renouncing ownership**: Consider relinquishing ownership rights once all necessary configurations are in place. By doing so, the owner forfeits the ability to alter contract variables, enhancing trust and decentralization. It's crucial to ensure all settings are finalized before renouncing ownership to prevent unintended consequences.
- These measures collectively strengthen the security and integrity of the smart contract, mitigating risks and promoting a more resilient and transparent ecosystem.

Upgradeability

Issue-ID	Severity	Location	Status
BAS–01	Major	Brewery.sol StratFeeManagerInitializable IStrategy.sol	Pending

Deployer can update the contract with new functionalities

The deployer can replace the old contract with a new one with new features. Be aware of this, because the owner can add new features that may have a negative impact on your investments.

Ownership

Issue-ID	Severity	Location	Status
BAS–02	Minor	All	Pending

The owner is not renounced and can modify the contract

The owner is able to change the state variables. The contract remains under the control of the owner. This means that the owner can change the state variables at any time and prevent transactions if necessary.

Note - If the contract remains undeployed, we would regard the ownership as not relinquished. Additionally, without any ownership functionalities, the ownership is presumed to be automatically renounced.

Authorities Permissions

Functions that alter the state and are equipped with access control can pose risks. Recognizing that misuse of these functionalities can result in financial losses is essential. We offer a guide to understand the implications of presence/absence functions further.

Minting tokens

Minting involves creating new tokens within a cryptocurrency or blockchain network. This task is commonly undertaken by the project's owner or an assigned authority, granting them the capability to increase the network's total token supply.

Authorities cannot mint new tokens

Authorities are not able to mint new tokens once the contract is deployed.

Burning Tokens

Burning tokens involves permanently removing a specific quantity of tokens from circulation, decreasing the total supply of a cryptocurrency or token. This practice is typically undertaken to enhance the value of the remaining tokens. By reducing the overall supply, burning creates a sense of scarcity, potentially stimulating demand and subsequently driving up the token's value.

Authorities cannot burn tokens without approval

Authorities are not able to burn tokens without any allowances.

Blacklist Addresses

Blacklisting addresses in smart contracts involves adding specific addresses to a blacklist, thereby preventing them from accessing or participating in certain functionalities or transactions within the contract. This measure is useful for preventing fraudulent or malicious activities, such as hacking attempts or money laundering.

Authorities cannot prevent function calls

Authorities are not able to prevent addresses from buying/selling.

Fees and Tax

In specific smart contracts, the individual or entity responsible for creating the contract retains the ability to establish fees for specific actions or functionalities within it. These fees serve various purposes, including covering operational expenses such as gas fees or compensating the contract's creator for their contributions in developing and managing the contract.

Authorities cannot update fees

Authorities are not able to update the fees to an unfair value for contract operations

Lock User Funds

In the context of a smart contract, locking entails confining access to specific tokens or assets for a predetermined duration. While tokens or assets are locked within the smart contract, they become inaccessible for transfer or utilization until the conclusion of the lock-up period or the fulfilment of specific conditions stipulated within the contract.

Authorities cannot lock funds

Authorities are not able to lock investor funds.

Logical Error

Issue-ID	Severity	Location	Status
BAS–3	Medium	Brewery.sol: L158	Fixed

Description

Since in solidity, negative numbers cannot be used on the uint256 type. If you want to go in the negative range, you have to use the int type. In this logical issue, they are checking for the negative number on an uint type, which will not work.

The logic here in the code below implemented is incorrect as the uint value cannot be less than zero, so the condition will become redundant and it will never be true.

Affected Code

```
158         uint _rDiff = user.depositAmt - r;
159         if(_rDiff < 0) {
160             user.depositAmt = 0;
161             removeUser(msg.sender);
162         }else{
163             user.depositAmt = user.depositAmt - r;
164         }
165         user.lastUpdate = block.number;
166
167         want().safeTransfer(msg.sender, r);
168
169         emit Withdraw(msg.sender, _shares);
170     }
171
```

Recommendation

We advise implementing a different logic according to the business logic where the deposit amount will be checked properly.

Potential Incorrect Implementation

Issue-ID	Severity	Location	Status
BAS–04	Medium	Brewery.sol:L182	Acknowledged

Description

If the new strategy contract doesn't have a "retireStrat" function, the strategy could be set inappropriately. If the owner accidentally puts a wrong contract address that does not have the retireStrat function on the next upgrade, then this function will not work.
Moreover, this could also pose a centralization risk where the Strat is upgraded to an address that doesn't have a retire function and in that case because the Strat was not retired, the "want()" token address will change and the owner will be able to claim the want tokens from the previous strategy. So, in this case, instead of 'want' tokens getting to the vault, they will be credited to the owner

Recommendation

Make sure to update the strategy address carefully.

Alleviation

Understood the concerns, we will take note when deploying upgrades - and we see that as highly unlikely.

As this is forked off Beefy, we do not intend to modify or remove any of their base code. We are just merely adding on additional calculation of points in the vault. Owners will be multi-sig wallets as well, and upgrades will be vetted before the proposal and before the deployment.

NatSpec Documentation missing

Issue-ID	Severity	Location	Status
BAS–05	Informational	N/A	Pending

Description

If you have started to comment on your code, comment on all other functions, variables, etc. Commenting on the code makes it more readable.

Disable Initializing

Issue-ID	Severity	Location	Status
BAS–06	Informational	Brewery.sol	Pending

Description

If the owner updates the contract, a disableInitializer call in the constructor must be implemented. This prevents the owner from calling the initialize function again to set the state variables in the contract. This should be implemented only if the contract was deployed before. Otherwise, the owner cannot call the initialize function to set the variables.

Recommendation

If the contract hasn't been deployed, remove the disableInitializer in the constructor. Otherwise, you cannot initialize the contract. When the contract has a deployed version already, leave it as it is.

Contract imports are not up-to-date

Issue-ID	Severity	Location	Status
BAS–07	Informational	Staking.sol StrategyVelodromeGaugeV2.sol	Pending

Description

We recommend importing all packages either with a certain version or using the latest one. In this case, the contracts are using a '–4' version that doesn't specify the complete version which is being used for production.

Code

```
import "@openzeppelin-4/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin-4/contracts/token/ERC20/utils/SafeERC20.sol";

```

Disclaimer

ChainAudits reports should not be construed as an endorsement or disapproval of any particular business or group. These reports do not reflect the economic value of any of the products or assets developed by the group. Also, ChainAudits does not consider integration with external contracts or services (e.g., Unicrypt, Uniswap, PancakeSwap).

ChainAudits reports aim to identify successful audit processes to help our clients improve the quality of the code and manage the risks associated with cryptographic tokens and blockchain technology. It is important to understand that blockchain technology and cryptographic assets pose significant ongoing risks. Each company and individual should conduct its own due diligence to maintain consistent safety measures. ChainAudits makes no representations about the security or performance of the technologies we audit.

ChainAudits does not provide any warranty or guarantee that the analyzed technology is completely defect-free, nor does it imply approval by the technology's owners. These audits should not be used to make input or output decisions; they will be involved in any project. They are not giving financial advice and should not be construed as such.