AstraSec

Kodiak Bault Security Audit Report

May 5, 2025

Contents

1 Introduction

1.1 About Kodiak Bault

The Kodiak Bault is a yield-optimizing vault system where users stake LP tokens to earn BGT rewards. Rewards are auto-compounded through an innovative auction mechanism that sells BGT for more LP tokens. The system includes a factory and an ERC4626-compliant vault. In auctions, solvers provide LP tokens to claim all unclaimed BGT and receive a fixed LP bounty. BGT can also be claimed directly or minted into liquid wrappers such as iBGT or LBGT.

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/Kodiak-Finance/bgt-baults
- CommitID: 6cd780a

  src/aultFactory.sol
  src/Bault.sol:
  src/BgtConverter.sol

- https://github.com/Kodiak-Finance/kodiak-periphery
- CommitID: 8863c91

  src/vaults/BaultRouter.sol (The inherited IslandRouter contract is not within the scope of the audit)

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the Kodiak Bault protocol. Throughout this audit, we identified a total of 3 low severity level issues. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won’t Do	Addressed
Critical	–	–	–	–
High	–	–	–	–
Medium	–	–	–	–
Low	3	2	–	1
Informational	–	–	–	–
Undetermined	–	–	–	–

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~L-1~~	Revisited Slippage Control in Bault::claimBgt()
L-2	Improved Corner Case Handling in Bault::withdraw()/redeem()
L-3	Potential Risks Associated with Centralization

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Acknowledged
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

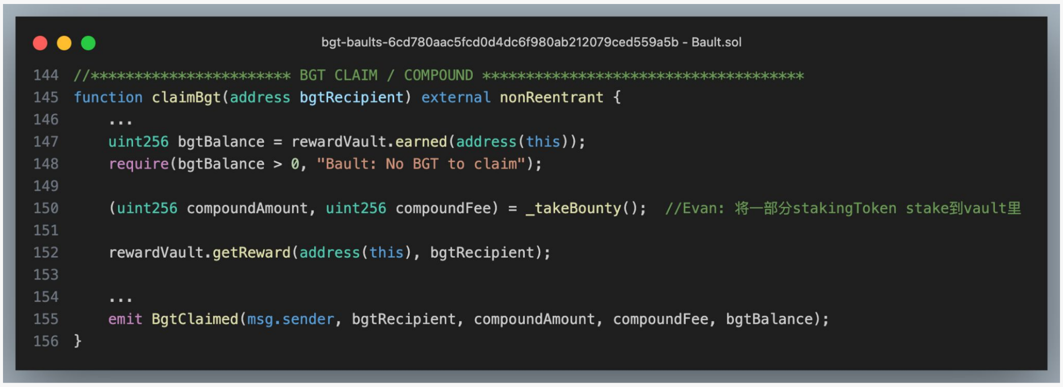
3.3 Vulnerability Details

3.3.1 [L-1] Revisited Slippage Control in Bault::claimBgt()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Bault.sol	Business Logic	Low	Low	Addressed

The claimBgt() function in the Bault contract retrieves the amount of BGT rewards using rewardVault.earned(address(this)) and requires that bgtBalance > 0. It then calls _takeBounty() to allow the caller to claim BGT rewards by providing a bounty amount of staking tokens. However, this implementation introduces a front-running arbitrage opportunity. A malicious actor could front-run the transaction, manipulating the BGT balance or bounty, resulting in the actual BGT received by the caller to significantly differ from the expected amount. To mitigate this risk, enhanced slippage protection mechanisms, such as specifying a minimum acceptable BGT output, should be implemented to ensure fair reward distribution.

Note that the claimBgtWrapper() function in the same contract shares the same issue.



bgt-baults-6cd780aac5fcd0d4dc6f980ab212079ced559a5b - Bault.sol

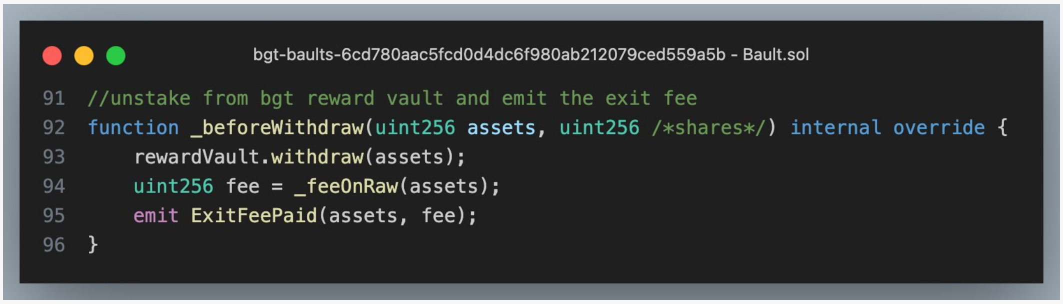
```
//*********************** BGT CLAIM / COMPOUND *************************************
function claimBgt(address bgtRecipient) external nonReentrant {
    ...
    uint256 bgtBalance = rewardVault.earned(address(this));
    require(bgtBalance > 0, "Bault: No BGT to claim");

    (uint256 compoundAmount, uint256 compoundFee) = _takeBounty();  //Evan: 将一部分stakingToken stake到vault里

    rewardVault.getReward(address(this), bgtRecipient);

    ...
    emit BgtClaimed(msg.sender, bgtRecipient, compoundAmount, compoundFee, bgtBalance);
}
```

Remediation To address this issue, implement slippage protection mechanisms, such as minimum output checks, in both claimBgt() and claimBgtWrapper() functions to ensure fair BGT value for the caller.

3.3.2 [L-2] Improved Corner Case Handling in Bault::withdraw()/redeem()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Bault.sol	Business Logic	Low	Low	Acknowledged

To prevent malicious users from hijacking rewards by entering the vault right before a reward claim and exiting right after, the Bault contract imposes an exit fee on users who withdraw. These exit fees are distributed as rewards to the remaining depositors. However, a corner case arises when the last user withdraws or redeems all assets: the fee collected from this last user remains in the reward vault as dust since the totalSupply becomes 0, making it unrecoverable. This could lead to a permanent loss of funds. To address this, the contract should either refund the fee to the last user or send the collected exit fee to the treasury contract before the totalSupply reaches zero.


bgt-baults-6cd780aac5fcd0d4dc6f980ab212079ced559a5b - Bault.sol

```
//unstake from bgt reward vault and emit the exit fee
function _beforeWithdraw(uint256 assets, uint256 /*shares*/) internal override {
    rewardVault.withdraw(assets);
    uint256 fee = _feeOnRaw(assets);
    emit ExitFeePaid(assets, fee);
}
```

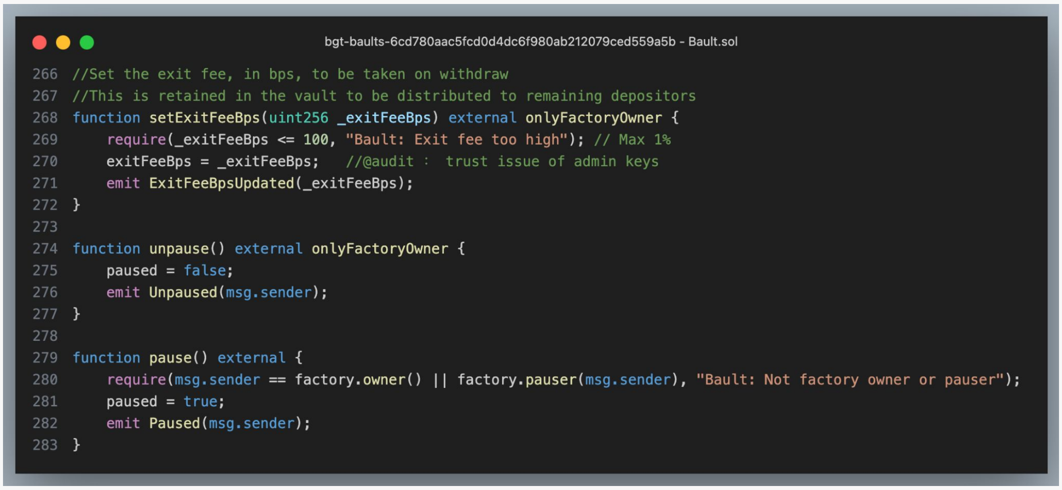
Remediation To address this issue, implement a mechanism to refund the exit fee to the last user or send it to the treasury contract before the totalSupply reaches 0.

Response By Team This issue has been confirmed by the team. The team has chosen not to implement a fix and leave it simple. The risk of leaving a dust amount of exitFee in the vault from the last depositor is known and that's ok.

3.3.3 [L-3] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	High	Low	Acknowledged

In the Kodiak Bault project, the existence of a privileged owner account introduces centralization risks, as it holds significant control and authority over critical operations governing the protocol. In the following, we show the representative function potentially affected by the privileges associated with the privileged account.


bgt-baults-6cd780aac5fcd0d4dc6f980ab212079ced559a5b - Bault.sol

```solidity
//Set the exit fee, in bps, to be taken on withdraw
//This is retained in the vault to be distributed to remaining depositors
function setExitFeeBps(uint256 _exitFeeBps) external onlyFactoryOwner {
    require(_exitFeeBps <= 100, "Bault: Exit fee too high"); // Max 1%
    exitFeeBps = _exitFeeBps;   //@audit : trust issue of admin keys
    emit ExitFeeBpsUpdated(_exitFeeBps);
}

function unpause() external onlyFactoryOwner {
    paused = false;
    emit Unpaused(msg.sender);
}

function pause() external {
    require(msg.sender == factory.owner() || factory.pauser(msg.sender), "Bault: Not factory owner or pauser");
    paused = true;
    emit Paused(msg.sender);
}
```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged account. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

Response By Team This issue has been confirmed by the team.

4 Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

4.3 Contact

Phone	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI