AstraSec

Yeet
Security Audit Report

June 26, 2025

Contents

1 Introduction

1.1 About Yeet Protocol

1.2 Source Code

1.3 Revision History

2 Overall Assessment

3 Vulnerability Summary

3.1 Overview

3.2 Security Level Reference

3.3 Vulnerability Details

4 Appendix

4.1 About AstraSec

4.2 Disclaimer

4.3 Contact

1 Introduction

1.1 About Yeet Protocol

Yeet is an innovative, gamified DeFi protocol built within the Berachain ecosystem, designed to offer a dynamic and unpredictable financial playground. With no dominant game-theoretic strategy, Yeet empowers players to explore a multitude of tactics, where they can win big or face losses through diverse approaches. This unique blend of strategy, chance, and creativity makes Yeet a thrilling addition to decentralized finance, inviting players to engage in a vibrant and ever-evolving economic game.

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/0xKingKoala/contracts-v2/tree/main
- Commit: 148cd666688bf1161396099f3171a36066bc197

This is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/0xKingKoala/contracts-v2/tree/main
- Commit: 4fcf7efcde2996515720321c5fb9644a7ce728d9

1.3 Revision History

Version	Date	Description
v1.0	June 26, 2025	Initial Audit

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the Yeet protocol. Throughout this audit, we identified a total of 4 issues spanning various severity levels. 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	2	–	–	2
Medium	1	1	–	–
Low	1	–	–	1
Informational	–	–	–	–
Total	4	1	–	3

3 Vulnerability Summary

3.1 Overview

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

- ~~H-1~~ Risk of Stuck State in Yeet Game
- ~~H-2~~ Lack of Access Control in CircularAddressBuffer
- M-1 Potential Risks Associated with Centralization
- ~~L-1~~ Incompatibility with Non-Standard ERC20s

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 re-mediation. 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 [H-1] Risk of Stuck State in Yeet Game

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Yeet.sol	Business Logic	High	Medium	Addressed

The Yeet Game smart contract includes validation logic in the _validateAndExecuteYeet() function to ensure a new "yeet" action aligns with the current round's state. One critical validation checks if the yeet time has expired (line 409), blocking further yeets once this end time is reached. Additionally, the restart() function permits any user to initiate a new round, but only if a winner has been determined for the current round (line 417). However, a corner case occurs when no player performs a yeet before the yeet time expires, resulting in no winner being generated. In this scenario, the game enters a stuck state: further yeets are prohibited due to the expired time, and the restart() function cannot be triggered due to the lack of a winner, preventing progression to a new round.

contracts-v2 - Yeet.sol

```
405 function _validateAndExecuteYeet(address yeeter, uint256 requiredPrice, bool isEmergencyYeet) private {
406     ...
407     // Validate timing
408     uint256 currentEndTime = endOfYeetTime();
409     if (timestamp >= currentEndTime) {
410         revert YeetTimePassed(timestamp, currentEndTime);
411     }
412     ...
413 }
414
415 function restart() external whenNotPaused {
416     // Check win condition: King of the Hill victory only
417     if (!hasLeaderWon()) {
418         revert RoundStillLive(roundNumber);
419     }
420
421     // Call prize manager to handle round end - lastYeeted is the winner
422     yeetPrizeManager.roundOverCallback(roundNumber, lastYeeted);
423     ...
424 }
```

Remediation To mitigate this issue, modify the restart() function to allow invocation after the yeet time has passed , even in the absence of a winner.

3.3.2 [H-2] Lack of Access Control in CircularAddressBuffer

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
CircularAddressBuffer	AccessControl	High	High	Addressed

The BGTTracker contract is responsible for tracking users eligible for BGT tokens within the game ecosystem, collaborating with the enhanced CircularAddressBuffer contract to manage token operations during YEET events. When a yeet occurs, the yeeter receives minted tokens staked into the rewardVault, enabling reward accrual. The CircularAddressBuffer implements a circular buffer to manage yeeter addresses and their balances. While these operations (e.g., enqueue(), dequeue(), replace()) are intended to be restricted to the BGTTracker contract, the current implementation lacks proper access control. This allows any user to modify the queue in the CircularAddressBuffer, potentially manipulating the yeeter addresses and balances. Such interference could disrupt reward distribution or block new yeets if staking amounts no longer align with the tracked data.

contracts-v2 - CircularAddressBuffer.sol

```
function dequeue() external returns (address, uint256) {
    require(size > 0, "Queue is empty");

    address person = queue[front];
    uint256 balance = balances[front];
    delete queue[front];
    delete balances[front];

    emit PersonRemoved(person, balance);

    front = (front + 1) % MAX_PHYSICAL_SIZE;
    size--;

    return (person, balance);
}
```

Remediation To address this vulnerability, implement access control by restricting enqueue(), dequeue(), and replace() operations to the BGTTracker contract using an onlyOwner modifier or a similar role-based mechanism. This ensures that only the intended contract can update the CircularAddressBuffer, safeguarding the accuracy of yeeter tracking and reward allocation.

3.3.3 [M-1] Potential Risks Associated with Centralization

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Medium	Medium	Acknowledged

The Yeet protocol centralizes significant protocol-wide control in a single privileged owner account, enforced through the onlyOwner modifier. This owner possesses the authority to perform critical actions, including updating dependency contracts (e.g., settings, taxContract, yeetPrizeManager, _bgtTracker), modifying configuration for settings, emergency withdrawing assets from yeetPrizeManager, and upgrading core contracts. This concentration of power contradicts the protocol's decentralization ethos, exposing it to substantial risks. A compromised or malicious owner could unilaterally alter protocol behavior, misappropriate funds, or disrupt the sale process, jeopardizing user trust and system integrity.

contracts-v2 - Yeet.sol

```
function updateSettings(address _newSettings) external onlyOwner {
    if (_newSettings == address(0)) revert ZeroAddress();
    settings = YeetSettingsStruct(_newSettings);
    emit SettingsUpdated(_newSettings);
}

function setTaxContract(address _taxContract) external onlyOwner {
    if (_taxContract == address(0)) revert ZeroAddress();
    taxContract = ITaxContract(_taxContract);
    emit TaxContractUpdated(_taxContract);
}
```

Remediation To mitigate centralization risks, consider implementing a multi-signature wallet (e.g., using Gnosis Safe) or a decentralized governance mechanism to manage critical actions. Additionally, introduce a time-lock mechanism for sensitive actions to provide users with advance notice and the opportunity to react to changes. If full decentralization is not feasible, ensure the owner's role is transparently documented, and consider transferring ownership to a secure, community-controlled entity over time to align with decentralization principles.

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

3.3.4 [L-1] Incompatibility with Non-Standard ERC20s

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
EmergencyWithdrawable	Compatibility	Low	Low	Addressed

In the EmergencyWithdrawable contract, the emergencyWithdrawERC20() function is designed to withdraw stuck ERC20 tokens from the contract. This function utilizes the ERC20Upgradeable::transfer() function, adhering to the standard ERC20 interface, to perform the token transfer. However, this approach introduces compatibility issues with non-standard ERC20 tokens, such as *USDT*, which does not return a value from its transfer() function. In contrast, ERC20Upgradeable::transfer() expects a boolean return value to confirm success. As a result, transfers involving non-standard tokens may fail, preventing the withdrawal of stuck tokens and locking funds in the contract.

Failure to withdraw stuck tokens due to incompatible ERC20 implementations could lead to permanent loss of funds, undermining the emergency withdrawal mechanism's reliability and exposing users or protocol to financial risk.

contracts-v2 - EmergencyWithdrawable.sol

```
function emergencyWithdrawERC20(address token, uint256 amount, address recipient) onlyOwner {
    require(recipient != address(0), "Cannot withdraw to zero address");
    ERC20Upgradeable(token).transfer(recipient, amount);

    emit EmergencyERC20Withdrawal(token, recipient, amount);
}
```

Remediation To ensure compatibility with all ERC20 token variants, replace the direct use of ERC20Upgradeable::transfer() with OpenZeppelin's SafeERC20 library. The SafeERC20 library handles non-standard token behaviors, such as those without return values (e.g., *USDT*), by incorporating safe transfer functions like safeTransfer().

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

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

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