TECHFUND Audit Score

Smart contract
Audit Report

Nanameue | YaY! staking project

1st Report [2024-08-07]

Score **POSITIVE**

Risk level		
	Critical	0
	High	0
	Medium	0
	Low	5
	Gas	3
	Info	3

Risk level detail

Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

The tester arrives at the likelihood and impact estimates, they can now combine them to get a final severity rating for this risk. Note that if they have good business impact information, they should use that instead of the technical impact information.

https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

Disclaimer

The TECHFUND team diligently endeavors to uncover as many vulnerabilities in the project within the specified time frame, however, it assumes no liability for the discoveries outlined in this document. Please note that a security audit conducted by the team does not constitute an endorsement of the underlying business or product. The audit was conducted within a predetermined timeframe, focusing solely on assessing the security aspects of the project.

Risk Classification

		Likelihood
		High
	High	C
Impact	Medium	H
	Low	M

We use the [OWASP] (https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) severity matrix to determine severity. See the documentation for more details.

Contracts Overview

> The StoneVault contract is a contract for managing deposits and withdrawals in a vault. The contract inherits from ReentrancyGuard and Ownable for security and access control.
>
> The `YayStoneToken` contract is an ERC20 token. It includes functionalities for staking ETH, requesting and canceling unstaking, instant unstaking, and redeeming Stone tokens. The contract uses access management and emergency withdraw capabilities from EmergencyWithdrawable

Key components of the contract:

- Deposits and Withdrawals: Handles user deposits and withdrawals.
- Rebase Mechanism: Periodic rebalancing of strategies.
- Strategies Management: Adding, clearing, and destroying strategies.
- Events: Emitting events for various state transitions and actions.

Executive Summary

Issues found

Severity	Number of issues found
Critical	0
High	0
Medium	0

Severity	Number of issues found
Low	5
Info	3
Gas	3
Total	11

Issues and Code Snippets

1. Critical Role Transfer Not Following Two-Step Pattern [Low]

Issue: The critical role transfer functions do not follow a two-step pattern, which increases the risk of unauthorized changes. This function simply updates the state variable with the new owner's address, provided it is not set to the zero address. If the new owner is an invalid account, ownership might be inadvertently given to an uncontrolled entity, which would render all functions that rely on the `onlyOwner()` modifier inoperative. The lack of a two-step confirmation process for such critical operations introduces a risk of errors and potential security issues.

https://docs.openzeppelin.com/contracts/2.x/api/ownership#Ownable-transferOwnership-address-

Functions:

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(newOwner !=  address(0), "Ownable: new owner is the zero
address");
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
}
```

2. Ownership May Be Renounced [Low]

Issue: If the owner renounces their ownership, all ownable functions guarded by the onlyOwner modifier will no longer be executable.

OpenZepplin `Ownable` https://docs.openzeppelin.com/contracts/2.x/api/ownership#Ownable-renounceOwnership--

Function:

```
function renounceOwnership() public virtual onlyOwner {
    emit OwnershipTransferred(_owner, address(0));
    _owner = address(0);
}
```

3. rebaseTime constrains should be implemented [Low]

Issue: The rebaseTime should be strictly in the future to prevent unexpected behavior.

```
constructor(
    address _minter,
    address _proposal,
    address payable _assetsVault,
    uint256 _rebaseTime,
    address[] memory _strategies,
    uint256[] memory _ratios
) {
    // Add constraint to ensure _rebaseTime is valid
    require(_rebaseTime > block.timestamp, "rebaseTime should be in the
future");
    rebaseTime = _rebaseTime;
}
```

4/5/6. Gas Optimizations [Gas]

Issue: Several functions can be marked as view to save gas. Also consider moving getVaultAvailableAmount inside share != 0 condition.

```
// Mark function as view
function currentSharePrice() public view returns (uint256 price) {}

function getVaultAvailableAmount() public view returns (uint256 idleAmount,
uint256 investedAmount) {}

// Unnecessary call when there are no shares
// In instantWithdraw function
// (uint256 idleAmount, ) = getVaultAvailableAmount();
// is calculated everytime but is used only in

function  instantWithdraw(
    uint256 _amount,
    uint256 _shares
) external nonReentrant returns (uint256 actualWithdrawn) {

    if (_shares != 0) {
        // consider calculating here
    }
}
```

7. Event Emissions for Monitoring [Info]

Issue: Emitting events for important state transitions can improve monitoring and debugging. Specially in case proposal is updated.

Event Emissions:

```
function updateProposal(address _proposal) external onlyProposal {
    emit ProposalUpdated(proposal, _proposal);
    proposal = _proposal;
}
```

8. Inconsistent version of Solidity used in files [Info]

Issue: Different versions of Solidity is used in the files. It is highly suggested to use similar solidity version across the project.

Version Information:

```
# EmergencyWithdrawable.sol : pragma solidity 0.8.23;
# YayStoneToken.sol : pragma solidity ^0.8.23;
# StoneVault.sol : pragma solidity 0.8.21;
```

9. Immutable State Variables [Info]

Issue: `stoneToken` and `stoneVault` should be declared `immutable` as their values are only set once in the constructor and never changed thereafter. This can save gas and enhance security by ensuring their values remain constant.

Recommendation: Update the state variable declarations to use the `immutable` keyword.

10. Unchecked Return Value [Low]

Issue: In the `redeemStoneTokens` function, the return value of `stoneToken.transfer` is ignored. This could result in unintended behavior if the transfer fails.

Recommendation: Check the return value of `stoneToken.transfer` to ensure the transfer succeeds.

11. Breaking "Check-Effect-Interaction Pattern" [Low]

Issue: The `requestUnstake` function performs an external call to `stoneVault.requestWithdraw` before updating the state variable `requestedUnstakes`. This breaks the "check-effect-interaction" pattern, all state variable updates (effect) should happen after external calls (interaction).

Recommendation: Update the state variables before making external calls.