Karak Security Review

Pashov Audit Group

Conducted by: T1MOH, rvierdiiev, 0xunforgiven

April 10th 2024 - April 14th 2024

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **karak-restaking** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Karak

Karak enables users to repurpose their staked assets to other applications. Stakers can allocate their assets to a Distributed Secure Service (DSS) on the Karak network and agree to grant additional enforcement rights on their staked assets. The opt-in feature creates additional slashing conditions to meet the conditions of secured services such as data availability protocols, bridges, or oracles.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - 18ab80f7d6c650dd62f570891546c1d47d08afc3

fixes review commit hash - f0971648ce32f59f8a65a265361c41193714115f
(moved to Andalusia-Labs repository)

Scope

The following smart contracts were in scope of the audit:

- `Vault`
- `VaultSupervisor`
- `VaultSupervisorLib`
- `DelegationSupervisor`
- `DelegationSupervisorLib`
- `Staker`
- `Withdraw`

7. Executive Summary

Over the course of the security review, T1MOH, rvierdiiev, 0xunforgiven engaged with Karak to review Karak. In this period of time a total of **5** issues were uncovered.

Protocol Summary

Protocol Name	Karak
Repository	https://github.com/Andalusia-Labs/karak-restaking
Date	April 10th 2024 - April 14th 2024
Protocol Type	Restaking protocol

Findings Count

Severity	**Amount**
Medium	2
Low	3
Total Findings	**5**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Users who deposit via returnShares() can be unable to withdraw	Medium	Resolved
[M-02]	First depositor attack is possible	Medium	Resolved
[L-01]	Withdrawals may encounter OOG as there is no max limit for list length	Low	Acknowledged
[L-02]	Centralization risk as the withdrawals can be paused	Low	Acknowledged
[L-03]	Contract Vault doesn't override the maxMint() function	Low	Acknowledged

8. Findings

8.1. Medium Findings

[M-01] Users who deposit via `returnShares()` can be unable to withdraw

Severity

Impact: High

Likelihood: Low

Description

The problem is that functions `gimmieShares()` and `returnShares()` don't update the array of vaults `stakersVaults`. For example, if the user's first deposit was made via `returnShares`, the vault won't be added to the user's vault array. As a result, withdrawal will revert when trying to remove that vault:

```
function removeShares(address staker, IVault vault, uint256 shares)
        external
        onlyDelegationSupervisor
        onlyChildVault(vault)
        nonReentrant
    {
        ...
        if (userShares == 0) {
            removeVaultFromStaker(staker, vault);
        }
    }
```

The issue can be mitigated by calling `gimmieShares()` to transfer all the shares and then calling normal `deposit`, however, this scenario is not properly documented and is not applicable in immutable integrator contracts.

Recommendations

```
function gimmieShares(IVault vault, uint256 shares) public onlyChildVault
      (vault) nonReentrant {
+       require(shares != 0);
        IERC20 shareToken = IERC20(vault);

        VaultSupervisorLib.Storage storage self = _self();
        // Verify the user is the owner of these shares
        if
          (self.stakerShares[msg.sender][vault] < shares) revert NotEnoughShares();

        self.stakerShares[msg.sender][vault] -= shares;
+       if (userShares == 0) {
+           removeVaultFromStaker(staker, vault);
+       }

        shareToken.transfer(msg.sender, shares);
    }

    function returnShares(IVault vault, uint256 shares) external onlyChildVault
      (vault) nonReentrant {
        IERC20 shareToken = IERC20(vault);

        VaultSupervisorLib.Storage storage self = _self();
+
+       require(shares != 0);
+       if (self.stakerShares[staker][vault] == 0) {
+           if
+ (self.stakersVaults[staker].length >= Constants.MAX_VAULTS_PER_STAKER) revert MaxSta
+           self.stakersVaults[staker].push(vault);
+       }

        self.stakerShares[msg.sender][vault] += shares;

        shareToken.transferFrom(msg.sender, address(this), shares);
    }
```

[M-02] First depositor attack is possible

Severity

Impact: High

Likelihood: Low

Description

It is a classic attack with the following flow:

1. The user is the first to interact with the vault, he sends a deposit transaction.
2. Attacker frontruns user's tx by performing 2 actions: 1) deposit 1 wei asset thus minting 1 wei share; 2) donate a big amount of asset to the vault.
3. User's transaction executes and the user mints 0 shares because of rounding in a formula that is used inside ERC4626: `shares = assets * sharesSupply`

`/ totalAssets`. That is because `sharesSupply` is 1 and `totalAssets` is large due to donation.
4. Then the attacker withdraws donated assets plus the user's deposit.

However, Karak-restaking uses Solady's ERC4626 implementation which makes the attack less profitable and harder to perform. By default it uses a different formula: in calculations of `shares` and `assets` it adds an extra 1. As a result, the attack is profitable in case there are 2 or more first deposits of approximately near size in mempool. Worth mentioning that vaults are the core mechanic of protocol, there will be multiple chances to perform the attack.

Recommendations

There are 2 possible mitigations:

1. Override function `_decimalsOffset()` in Vault.sol. It will increase the decimals of Vault.
2. Create "dead shares" like it's done in UniswapV2, i.e. deposit the initial amount to the vault on the initialization step.

8.2. Low Findings

[L-01] Withdrawals may encounter OOG as there is no max limit for list length

Users have to call `startWithdraw()` and then call `finishWithdraw()` to withdraw their funds from the system. When users call the withdraw function they specify a list of vaults they want to withdraw from, the issue is that the logic inside `finishWithdraw()` is more complex and would consume more gas. There may be scenarios that user calls to `startWithdraw()` executes but later users won't be able to call `finishWithdraw()` because of OOG. The list of vaults the user specifies may have duplicate items too, so the list can have more items than `MAX_VAULTS_PER_STAKER`.

[L-02] Centralization risk as the withdrawals can be paused

Users have to call `startWithdraw()` and `finishWithdraw()` to withdraw their funds but the issue is that those functions would revert when a contract is paused so there is a centralization risk that the user won't be able to access their funds and withdraw them if protocol admin was compromised.

[L-03] Contract Vault doesn't override the `maxMint()` function

In the contract Vault, there is `assetLimit` to make sure that Vault won't receive more than that amount of assets. Function `deposit()` checks this limit but function `mint()` has no such check so it would be possible to bypass that limit by using function `mint()`.

Also, the code doesn't override the function `maxMint()` and even so function `maxDeposit()` exists in Vault code and returns the correct value based on `assetLimit` but function `maxMint()` doesn't exists in Vault and it would return `uint(256).max` which is set in the Solady library. This is against ERC4626 standard as values return by `maxMint()` and `maxDeposit()` don't match.

Add function `maxMint()` and also add `assetLimit` check to function `mint()`.