SALUS SECURITY

MAY 2023

# CODE SECURITY ASSESSMENT

ANKR

# Overview

## Project Summary

- Name: Ankr
- Platform: Ethereum
- Language: Solidity
- Repository: https://github.com/Ankr-network/stakefi-smart-contract
- Audit Scope: See Appendix - 1

# Project Dashboard

## Application Summary

| | |
|---|---|
| **Name** | Ankr |
| **Version** | v2 |
| **Type** | Solidity |
| **Dates** | May 19 2023 |
| **Logs** | Apr 28 2023; May 19 2023 |

## Vulnerability Summary

| | |
|---|---|
| **Total High-Severity issues** | 0 |
| **Total Medium-Severity issues** | 5 |
| **Total Low-Severity issues** | 4 |
| **Total informational issues** | 6 |
| **Total** | 15 |

## Contact

E-mail: support@salusec.io

## Risk Level Description

| | |
|---|---|
| **High Risk** | The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users. |
| **Medium Risk** | The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact. |
| **Low Risk** | The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances. |
| **Informational** | The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth. |

# Content

# Introduction

## 1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

## 1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

## 1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

# Findings

## 2.1 Summary of  Findings

| ID | Title | Severity | Category | Status |
|---|---|---|---|---|
| 1 | Centralization risk | Medium | Centralization | Acknowledged |
| 2 | No function sets the _exits variable | Medium | Business logic | Resolved |
| 3 | Lack of access control on cleanUserLocks() | Medium | Access control | Acknowledged |
| 4 | Slashed amount for each providers is not tracked | Medium | Business logic | Resolved |
| 5 | Return value of _unsafeTransfer is not being checked in several places | Medium | Validation | Resolved |
| 6 | Lack of access control on claim() | Low | Access control | Acknowledged |
| 7 | Flawed implementation of totalSupply() | Low | Business logic | Acknowledged |
| 8 | Incorrect value logged in event | Low | Logging | Resolved |
| 9 | mint() does not return correct value | Low | Business logic | Resolved |
| 10 | Mismatch between code and description | Informational | Code quality | Resolved |
| 11 | Mixed use of reentrancy prevention modifiers | Informational | Reentrancy | Resolved |
| 12 | Missing reentrancy guard | Informational | Reentrancy | Resolved |
| 13 | Can add old parameters to SwapFeeParamsChanged() event | Informational | Logging | Resolved |
| 14 | Lack of events | Informational | Logging | Acknowledged |
| 15 | Redundant code | Informational | Redundancy | Resolved |

## 2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

| | |
|---|---|
| **1. Centralization risk** | |
| Severity: Medium | Category: Centralization |
| Target:<br>- All | |

### Description

Throughout the code base, there are several privileged roles.

The owner of the GlobalPool_R42 contract:
- can update the address of _aethContract, _fethContract, _configContract, _withdrawalPool, and_stakingContract
- can change the operator
- can add or remove vault from allowlist

The operator of the GlobalPool_R42 contract:
- can push ethers to allowed vault or beacon chain
- can distribute rewards using distributeRewards(). Notice that the feeAmount that was transferred to the treasury is inputted by the operator
- can make force exit for providers by using forceAdminProviderExit()
- can reset the provider rewards by using resetLockedEthForProviders()
- can slash provider by using slashETH()

The owner of the AETH_R18 contract:
- can reset the ratio of bonds to shares by using repairRatio()
- can set the address of _globalPoolContract, _bscBridgeContract, and _feeRecipient
- can change the operator
- can act as an operator
- can pause or unpause the contract

The operator of the AETH_R18 contract:
- can update the ratio to a value within the threshold
- can set name and symbol of the token

The owner of the FETH_R18 contract:
- can set the address of _globalPoolContract and _aEthContract,

The owner of the WithdrawalPool contract:
- can change the _pool address, which is the recipient of withdrawals

The pauseAddr in the DepositWrapper contract:
- can pause and unpause the contract

The governanceAddr in the DepositWrapper contract:
- can set the depositAddr and the pauseAddr

If these privileged accounts are plain EOA accounts, this poses a risk to the users. If any of the private keys is compromised, an attacker could exploit the privileged operations to attack the project.

Moreover, the upgradeable proxy pattern is used in the GlobalPool, AETH, and FETH contracts. The proxy admin controls the upgrade mechanism to upgradeable proxies, and can change the respective implementations. Should the admin's private key be compromised, an attacker could upgrade the logic contract to execute their own malicious logic on the proxy state.

## Recommendation

We recommend transferring privileged accounts to multi-signature accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties, with a designated time limit for approval.

## Status

This issue has been acknowledged by the team.

## 2. No function sets the _exits variable

| | |
|---|---|
| Severity: Medium | Category: Business logic |
| Target:<br>- legacy/contracts/upgrades/GlobalPool_R42.sol | |

## Description

The GlobalPool_R42 contract uses **_exits** to track the exit block number for a provider.

legacy/contracts/upgrades/GlobalPool_R42.sol:L150-L154

```
modifier notExitRecently(address provider) {
    require(block.number >
_exits[provider].add(_configContract.getConfig("EXIT_BLOCKS")), "Recently exited");
    delete _exits[msg.sender];
    _;
}
```

The _exits state variable is used in the notExitRencently() modifier, which modifies the claimAETH() and claimFETH() function.

legacy/contracts/upgrades/GlobalPool_R42.sol:L606-L613

```
function softLockBlockNumber(address provider) public view returns (uint256) {
    uint256 exitedAt = _exits[provider];
    if (exitedAt == 0) {
        return 0;
    }
    uint256 waitFor = _configContract.getConfig("EXIT_BLOCKS");
    return exitedAt.add(waitFor);
}
```

The _exits is also used in the softLockBlockNumber() function.

However, the _exits variable is not being set by any function in the GlobalPool_R42 contract.

## Recommendation

The operator can call forceAdminProviderExit() to make providers exit. This function uses the _forceProviderExitFor() internal function to handle its business logic. We suspect _forceProviderExitFor() should update _exits but does not. If that is the case, we recommend updating _exits in _forceProviderExitFor().

## Status

The team has resolved this issue by removing the logic related to _exits.

### 3. Lack of access control on cleanUserLocks()

| Severity: Medium | Category: Access control |
|---|---|
| Target:<br>- legacy/contracts/upgrades/AnkrDeposit_R3.sol | |

## Description

The cleanUserLocks() function in AnkrDeposit_R3 contract is used by other functions (e.g. _claimAndDeposit(), _unfreeze()) to clear the lock information for a user.

However, this function is defined as a **public** function, meaning anyone can call it, which can result in state-changing logic being executed in an unintended context.

## Recommendation

If cleanUserLocks() is designed for internal use only, change its visibility to internal or private. If it is also for external use, add proper access control to it.

## Status

This issue has been acknowledged by the team. The team stated that public is okay for cleanUserLocks because it has an if statement to skip locks that do not end.

| 4. Slashed amount for each providers is not tracked | |
|---|---|
| Severity: Medium | Category: Business logic |
| Target:<br>- legacy/contracts/upgrades/GlobalPool_R42.sol | |

## Description

The slashETH() function in the GlobalPool_R42 contract can be used by the operator to slash a provider. However, this function only tracks the total slashed amount, _totalSlashedETH, and does not track the amount slashed for each individual provider.

The slashingsOf() function in the GlobalPool_R42 contract is intended to return the amount slashed for a provider, but currently it returns zero. This may cause confusion for end users and developers who wish to integrate the protocol.

## Recommendation

We recommend adding a state variable to track the amount slashed for a provider, and updating it in the slashETH() function. Then, we can use this state variable in the slashingsOf() function to return the amount slashed for the provider.

## Status

The team has resolved this issue by removing the logic related to slashed ETH.

| 5. Return value of _unsafeTransfer is not being checked in several places | |
|---|---|
| Severity: Medium | Category: Validation |
| Target:<br>- legacy/contracts/upgrades/GlobalPool_R42.sol | |

## Description

The _unsafeTransfer() function in the GlobalPool_R42 contract returns a boolean value indicating whether the transfer is successful or not.

However, the pushToVault() and distributeRewards() functions use _unsafeTransfer() without checking its return value. This can result in _unsafeTransfer() failing silently and the subsequent logic being executed regardless.

## Recommendation

Add a check for _unsafeTransfer()’s return value in these functions.

## Status

The team has resolved this issue.

## 6. Lack of access control on claim()

| | |
|---|---|
| Severity: Low | Category: Access control |
| Target:<br>- legacy/contracts/upgrades/FeeRecipient_R1.sol | |

## Description

The claim() function in the FeeRecipient_R1 contract is intended to transfer the balance to the GlobalPool and the treasury. Currently, anyone can call this function, which could result in it being executed in an unintended context.

## Recommendation

Currently, the claim() function is called by the updateRatioAndClaim() function in the AETH_R18 contract. We recommend adding proper access control to the claim() function if it is designed to be used only by AETH.

Alternatively, if the function is intended for public use, we suggest adding a comment to indicate its public visibility and prevent the addition of sensitive operations in future upgrades.

## Status

This issue has been acknowledged by the team. The team has stated that the claim doesn’t have any special context, it can be triggered without restrictions, the goal is to transfer ETH to the treasury and staking pool.

## 7. Flawed implementation of totalSupply()

| Severity: Low | Category: Business logic |
|---|---|

Target:
- legacy/contracts/upgrades/FETH_R18.sol

## Description

In the FETH_R18 contract, users can use the lockShares() and unlockShares() functions to swap between AETH (i.e. ankrETH) and FETH (i.e. aETHb). The _shares state variable is used to track the locked amount for a user, and the balanceOf() function returns the FETH balance of a user based on the _shares[user] value.

legacy/contracts/upgrades/FETH_R18.sol:L124-L127

```
function totalSupply() public view override returns (uint256) {
    uint256 totalLocked = IERC20(_aEthContract).balanceOf(address(this));
    return sharesToBonds(totalLocked);
}
```

However, the totalSupply() function in the FETH contract uses the contract's AETH balance as the total locked AETH shares. This could lead to issues because people can send AETH token directly to the contract without using the lockShares() function. As a result, the sum of locked user AETH shares may not equal the AETH balance in the contract. This means that totalSupply()'s return value may not equal the sum of the balanceOf() for all the users, and the caller of FETH's totalSupply() may use this incorrect value in its subsequent logic.

## Recommendation

Consider adding a state variable to track the total locked AETH amount in the contract. This state variable can be updated whenever a user locks or unlocks AETH shares. We can then use this variable in the totalSupply() function to return the correct total supply of FETH.

## Status

This issue has been acknowledged by the team.

## 8. Incorrect value logged in event

| Severity: Low | Category: Logging |
|---|---|

Target:

- legacy/contracts/upgrades/FETH_R18.sol

## Description

legacy/contracts/upgrades/FETH_R18.sol:L73-L87

```
function lockShares(uint256 shares) external {
    address spender = msg.sender;
    // transfer tokens from aETHc to aETHb
    require(IERC20(_aEthContract).transferFrom(spender, address(this), shares), "can't
transfer");
    // calc swap fee (default swap fee ratio is 0.3%=0.3/100*1e18, fee can't be greater
than 1%)
    uint256 fee = shares.mul(_swapFeeRatio).div(1e18);
    if (msg.sender == _swapFeeOperator) {
        fee = 0;
    }
    uint256 sharesWithFee = shares.sub(fee);
    // increase senders and operator balances
    _shares[_swapFeeOperator] = _shares[_swapFeeOperator].add(fee);
    _shares[spender] = _shares[spender].add(sharesWithFee);
    emit Locked(spender, shares);
}
```

If the Locked() event in the code is meant to record the shares locked for a user, then the lockShares() function should `emit Locked(spender, shares.sub(fee))` instead.

## Recommendation

Log the actual shares locked for a user in the Locked() event.

Additionally, to improve clarity, we recommend redesigning the Locked() event to `event Locked(address account, uint256 lockedShare, uint256 fee)`.

## Status

The team has resolved this issue.

## 9. mint() does not return correct value

| | |
|---|---|
| Severity: Low | Category: Business logic |

Target:
- legacy/contracts/upgrades/AETH_R18.sol

## Description

legacy/contracts/upgrades/AETH_R18.sol:L92-L95

```
function mint(address account, uint256 amount) external returns (uint256 _amount) {
    require(msg.sender == address(_bscBridgeContract) || msg.sender ==
address(_globalPoolContract), 'Not allowed');
    _mint(account, amount);
}
```

The mint() function in the AETH_R18 contract is defined to return a uint256 value. However, there is currently no return statement in this function, and the return parameter, _amount, is not being updated in the function.

## Recommendation

If the mint() function is meant to return a value, we suggest adding a return statement in the function to return the correct value.

Alternatively, if the mint() function does not need to return a value, we recommend changing its definition to `function mint(address account, uint256 amount) external {...}`. This will make it clear that the function does not return a value.

## Status

The team has resolved this issue.

## 2.3 Informational Findings

| | |
|---|---|
| **10. Mismatch between code and description** | |
| Severity: Informational | Category: Code quality |
| Target:<br>- deposit-wrapper/contracts/DepositWrapper.sol<br>- legacy/contracts/upgrades/GlobalPool_R42.sol | |

### Description

1. deposit-wrapper/contracts/DepositWrapper.sol:L186-L189

```
require(
    newDepositAddr != pauseAddr,
    "Deposit address must be different from deposit address"
);
```

The error message should be "Deposit address must be different from pause address".

2. legacy/contracts/upgrades/GlobalPool_R42.sol:L125

```
mapping(address => uint256) private _aETHProviderRewards; // slot:323
```

The comment states that the slot for _aETHProviderRewards is 323, but in reality, it is 328.

3. legacy/contracts/upgrades/GlobalPool_R42.sol:L615-L618

```
/**
    @dev Slash eth, returns remaining needs to be slashed
*/
function slashETH(address provider, uint256 amount) public onlyOperator {
```

The comment indicates that the slashETH() should return a value. However, the slashETH() is defined without a return value.

4. legacy/contracts/upgrades/GlobalPool_R42.sol:L361

```
require(amount >= _configContract.getConfig("UNSTAKE_MIN_AMOUNT"), "Value must be
greater than minimum amount");
```

The “greater than” should be “no less than”.

### Recommendation

Align the description with the code.

### Status

The team has resolved this issue.

## 11. Mixed use of reentrancy prevention modifiers

| Severity: Informational | Category: Reentrancy |
|---|---|
| Target:<br>- legacy/contracts/upgrades/GlobalPool_R42.sol | |

## Description

In the GlobalPool_R42 contract, both the unlock() modifier and nonReentrant() modifier are used to prevent reentrancy attacks. The issue with mixed use of them is that a function modified by unlock() can enter a function modified by nonReentrant() and vice versa. In other words, cross-function reentrancy is possible when you mix these two reentrancy prevention modifiers in one contract.

## Recommendation

Replace the unlock() modifier with the nonReentrant() modifier throughout the contract.

## Status

The team has resolved this issue.

| 12. Missing reentrancy guard | |
|---|---|
| Severity: Informational | Category: Reentrancy |
| Target:<br>- legacy/contracts/upgrades/GlobalPool_R42.sol | |

## Description

The nonReentrant() modifier can be added to the following functions in the GlobalPool_R42 contract to prevent the introduction of future reentrancy vulnerabilities:

- distributeRewards()
- claimManually()

## Recommendation

Consider adding the nonReentrant() modifier to these functions.

## Status

The team has resolved this issue.

## 13. Can add old parameters to SwapFeeParamsChanged() event

| | |
|---|---|
| Severity: Informational | Category: Logging |
| Target:<br>- legacy/contracts/upgrades/FETH_R18.sol | |

## Description

The SwapFeeParamsChanged() event only logs the new values, while the other “changed” events (e.g. GlobalPoolAddressChanged()) log both the previous value and the new value.

To provide more information and context in the SwapFeeParamsChanged() event, we suggest adding the old parameters to the event as well. This will allow developers and users to see the previous values of the parameters and better understand the changes made to the swap fee parameters.

## Recommendation

Change

```
event SwapFeeParamsChanged(address operator, uint256 ratio);
```

To

```
event SwapFeeParamsChanged(address prevOperator, address newOperator, uint256 prevRatio,
uint256 newRatio);
```

and updating the event emitting code accordingly.

## Status

The team has resolved this issue.

| ### 14. Lack of events | |
|---|---|
| Severity: Informational | Category: Logging |
| Target:<br>- legacy/contracts/upgrades/GlobalPool_R42.sol<br>- legacy/contracts/upgrades/FeeRecipient_R1.sol | |

## Description

1. Lack of event when poolBalance is insufficient in distributeRewards()

When poolBalance is insufficient to distribute further rewards, the distributeRewards() function does not emit relevant events. To enable off-chain tracking for this situation, an event for this situation should be added to notify the team and allow them to take appropriate action.

2. Lack of event in FeeRecipient's claim() function

The claim() function in the FeeRecipient_R1 contract is a critical function that transfers funds to the pool and the treasury. However, no relevant events are emitted in this function. To enable off-chain tracking, we recommend defining an appropriate event and emitting it in the claim() function.

## Recommendation

Design proper events and add them to aforementioned functions.

## Status

The team has acknowledged this issue.

## 15. Redundant code

| Severity: Informational | Category: Redundancy |
|---|---|

Target:

- timelock/time-lock/contracts/AnkrTimeLock.sol
- legacy/contracts/upgrades/FETH_R18.sol
- legacy/contracts/upgrades/GlobalPool_R42.sol

## Description

1. timelock/time-lock/contracts/AnkrTimeLock.sol:L6-L7

```
// Uncomment this line to use console.log
// import "hardhat/console.sol";
```

The above lines are unnecessary and can be removed before deploying the contract.

2. The onlySwapFeeOperator() modifier in the FETH_R18 is defined but not used.

3. Logic that has been commented out can be removed.

## Recommendation

Remove the redundant code.

## Status

The team has resolved this issue.

# Appendix

## Appendix 1 - Files in Scope

This audit covered the following files:

1. The DepositWrapper contract in pull request 514:

| File | SHA-1 hash |
|---|---|
| eth-deposit-wrapper/contracts/DepositWrapper.sol | 7649c3e8b91e01fe7cab7228380f88d7c4fe2f76 |

2. The AnkrTimeLock contract in pull request 505:

| File | SHA-1 hash |
|---|---|
| time-lock/contracts/AnkrTimeLock.sol | e6ddc47cafb6ae6c50c66d773c2fcd9ad48e3392 |

3. The following files in commit 71a4183:

| File | SHA-1 hash |
|---|---|
| legacy/contracts/Config.sol | 6417bffeb53287b84a3cca96ffd6f2764c183f1c |
| legacy/contracts/Governable.sol | a33d161fbee771da246e78643d69c942ec4b92d6 |
| legacy/contracts/SystemParameters.sol | 206aaaf1bf2e8a8add4e32b8593596031edaa433 |
| legacy/contracts/WithdrawalPool.sol | 59d754f3d47a5d02ce9fd72a13dbebd9dc9c5d63 |
| legacy/contracts/lib/Lockable.sol | d4cd0ba14a368e85494c28ae8a805628713e193d |
| legacy/contracts/lib/MathUtils.sol | 744b1d9c802aa2720eb411c9399482ba9aa8f485 |
| legacy/contracts/lib/Ownable_R1.sol | 3292409bf77499da7569ba68dcf9bcc530e9a607 |
| legacy/contracts/lib/Pausable.sol | 47447228ed508e26ffea4bc42f2be3ec4325396e |
| legacy/contracts/lib/interfaces/IAETH.sol | 876e6df562909ab6094dfe97e58a925438df420d |
| legacy/contracts/lib/interfaces/IConfig.sol | 8fa3a101ba9369d5ced7fd323433d3dd93245c65 |
| legacy/contracts/lib/interfaces/IDepositContract.sol | e30792721e4299995549a296663fc19cbcaa27bf |
| legacy/contracts/lib/interfaces/IFETH.sol | 1ef777e075775643d3a8883404e0ccf7cb36c048 |
| legacy/contracts/lib/interfaces/IFeeRecipient.sol | 23846760b12254494e94295177b9bc1d21461abc |
| legacy/contracts/lib/interfaces/IGlobalPool.sol | 80dfe187de9eb9251b113f30ec2be11e65d541c9 |

| | |
|---|---|
| legacy/contracts/lib/interfaces/IWithdrawalPool.sol | 5ff64705c97f87e87e1c742b768f6fff3f8b9c04 |
| legacy/contracts/lib/openzeppelin/ERC20UpgradeSafe.sol | 1001e78c89b070e04c249242ebab224fc98e02cb |
| legacy/contracts/upgrades/AETH_R18.sol | 7bd5a51b994825b6a005eee3cd122efd07b7027c |
| legacy/contracts/upgrades/FETH_R18.sol | 206b34e1ebdf9cd0786afdc45d0cfcf620f9f13e |
| legacy/contracts/upgrades/GlobalPool_R42.sol | a04393756d6dad654c501bb14c07876312fd4568 |
| legacy/contracts/FeeRecipient.sol | 11b399f4e60fb8f7d197cb081c441653fdeb000f |