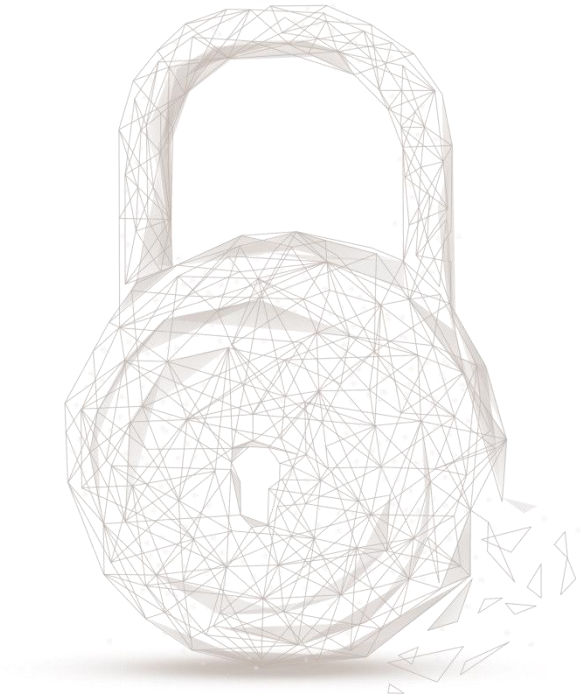
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number: 202107221421

Report Query Name: AaveStrategy

Smart Contract Address Link:

https://github.com/RAMP-DEFI/strategies/blob/master/contracts/strategies/ExtendedBaseStrategy.sol

https://github.com/RAMP-DEFI/strategies/blob/master/contracts/strategies/interfaces/IStakingPool.sol

https://github.com/RAMP-DEFI/strategies/blob/master/contracts/strategies/SingleAssetAccruingStrategy.sol

https://github.com/RAMP-DEFI/strategies/blob/master/contracts/strategies/stakingpools/AaveStakingPoolV2.sol

https://github.com/RAMP-DEFI/strategies/blob/master/contracts/strategies/interfaces/aave/IAaveLendingPoolV2.sol

Commit Hash:

Start: cc4c11ba8ca09a671597ba3c7ae59d18884ca950

Final: 9f48723be079f001b7f472f40565d194f832462a

Start Date: 2021.07.15

Completion Date: 2021.07.22

Overall Result: Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass
		Pseudo-random Number Generator (PRNG)	Pass

		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass
		Access Control of Owner	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	Pass

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of contracts AaveStrategy, including Coding Standards, Security, and Business Logic. **The AaveStrategy contracts passed all audit items. The overall result is Pass. The smart contracts is able to function properly.**

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing BNB.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract.
- Result: Pass

2.10 Replay Attack

- Description: Check whether the implement possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Business Security

The contract function of this audit is used for Aave's investment management. The main contract consists of two parts. One part is the general singleAssertAccruingStrategy contract, which inherits the ExtendedBaseStrategy contract, and the other part is the AaveStakingPool contract for interacting with aave. The assets used for investment are all managed in singleAssertAccruingStrategy, and AaveStaking is only used to provide code logic and can be replaced by the operator.

(1) ExtendedBaseStrategy

- Description: The contract mainly implements the main functions of the strategy, including deposits, withdrawals, liquidation, updates, emergency withdrawals, return of total assets, etc. It also implements *sweep* for cleaning up other assets in the contract. The contract implements *setStakingPool* to set the address of the StakingPool contract, and the operator authority can be called. Since the main function of the contract is to use the delegatecall mechanism, this allows the operator to have greater authority, and the operator permissions need to be properly kept to avoid loss.

```
341     ///@dev Setter for stakingPool
342     function setStakingPool(address _stakingPool) public virtual onlyOperator {
343         address previousStakingPool = stakingPool;
344         stakingPool = _stakingPool;
345         emit StakingPoolUpdated(previousStakingPool, stakingPool);
346     }
```

Figure 1 Source code of function *setStakingPool*

```
94      // Set the Vault
95      function setVault(address _address) public onlyOperator {
96          vault = _address;
97      }
```

Figure 2 Source code of function *setVault*

- Related functions: *ondeposit*, *handleDeposit*, *onwithdraw*, *handleWithdraw*, *onLiquidate*, *handleLiquidate*, *getPoolAmount emergencyWithdraw*, *handleEmergencyWithdraw*, *sweep*, *update*
- Result: Pass

(2) SingleAssetAccruingStrategy

- Description: The singleAssetAccruingStrategy contract is an instance of the ExtendedBaseStrategy contract. The ExtendedBaseStrategy contract cannot be used alone. Instead, the singleAssetAccruingStrategy contract inherited from it is used as a strategic contract for asset investment management. In the singleAssetAccruingStrategy contract, *work* function is used to convert and reinvest the settled rewards. But in this strategy for aave, this part is not needed.

```
function work(
    address // _token unused
) public virtual override {
    // 1. Ensure we have some rewards to reinvest
    uint256 rewardBalance = rewardToken.balanceOf(address(this));
    if (rewardBalance == 0) return;
    // 2. If applicable transfer reward fee to devAddress
    if (devFeePercentage > 0) {
        uint256 devFee = rewardBalance.mul(devFeePercentage).div(100);
        rewardToken.safeTransfer(devAddress, devFee);
        rewardBalance = rewardBalance.sub(devFee);
    }
    // 3. Swap rewardToken to assetToken if needed
    if (address(rewardToken) != address(assetToken)) {
        require(rewardToken.approve(address(swapMarket), rewardBalance), 'SingleAssetAccruingStrategy.work: approve failed');
        swapMarket.swap(rewardBalance, 0, rewardTokenToAssetTokenPath, address(this));
    }
    // 4. Stake the assetToken balance
    uint256 assetTokenBalance = assetToken.balanceOf(address(this));
    (bool success, bytes memory data) = stakingPool.delegatecall(
        abi.encodeWithSignature("deposit(address,address,address,address,uint256)", address(assetToken), address(receiptToken), address(rewardToken), address(this), assetTokenBalance)
    );
    require(success, 'SingleAssetAccruingStrategy.work failed');
}
```

Figure 3 Source code of function *work*

- Related functions: *work, setRewardTokenToAssetTokenPath, setSwapMarket, getStrategyType*
- Result: Pass

(3) AaveStakingPoolV2

- Description: The AaveStakingPoolV2 contract is the specific logic for interacting with aave. The contract itself does not store any funds, but only provides the interactive logic code to the SingleAssetAccruingStrategy contract, which mainly includes deposits to aave, withdrawals, settlement income, emergency withdrawal, and return of the underlying assets of the current investment. On the other hand, this contract must be inherited by other contracts and implement the aaveLendingPoolAddress method before it can be deployed and used.

```
    /**
    @dev Subclasses must return the address of the Aave contract for a specific blockchain
    */
    function aaveLendingPoolAddress() public pure virtual returns(address);

```

Figure 4 Source code of function *aaveLendingPoolAddress*

- Related functions: *aaveLendingPoolAddress, balanceOf, deposit, withdraw, emergencyWithdraw, collectRewards*
- Result: Pass

4. Conclusion

Beosin(Chengdu LianAn) conducted a detailed audit on the design and code implementation of the smart contract AaveStrategy. The contract AaveStrategy passed all audit items, The overall audit result is **Pass.**

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com