FLAT Protocol Smart Contract Audit Report

Contents

1. Disclaimer

A smart contract security review can't find every vulnerability. It is limited by time, resources, and expertise. The goal is to catch as many issues as possible, but I can't promise complete security. I also can't guarantee that the review will find any problems. To stay safer, it's best to do more reviews, run bug bounty programs, and keep monitoring on-chain activity.

2. Introduction

A time-boxed security review was done, with a focus on the security aspects of the smart contracts implementation. The FLAT Protocol team has been very responsive to auditors inquiries, demonstrating a strong commitment to security by taking into account all the recommendations and fixing suggestions from the researchers.

2.1 About FLAT Protocol

The AFLAT contract allows users to mint AFLAT tokens by depositing stablecoins (USDC, DAI, or USDT). FLATEngine.sol enables token holders to directly convert their AFLAT tokens into FLAT tokens by receiving AFLAT via AFLATToken.sol, calculating the exact FLAT amount using a fixed conversion ratio, and minting the new tokens through FLATToken.sol. Concurrently, Vesting.sol enforces a strict vesting schedule that locks the converted FLAT tokens and gradually releases them over time, ensuring controlled liquidity. Together, AFLATToken.sol, FLATEngine.sol, FLATToken.sol, and Vesting.sol establish a direct, secure, and automated workflow for token migration and vesting.

3. Vulnerability Classifications

Vulnerability Level	**Classification**
Critical	Easily exploitable by anyone, causing loss/manipulation of assets or data.
High	Arduously exploitable by a subset of addresses, causing loss/manipulation of assets or data.
Medium	Inherent risk of future exploits that may or may not impact the smart contract execution.
Low	Minor deviation from best practices.
Gas	Best practices for optimaizting gas.

4. Executive Summary

4.1 Protocol Info

Name:	FLAT Protocol
Security Review Type:	Smart Contract Audit

4.2 Scope

The following smart contracts were in the scope of the security review:

Contract	Address
AFLAT	0x1caDF57c2f8dfE096b579f149DA56434824f2F61
FLAT	0x8EB03ddb0A0c97Be83d38d9aD259a2DC57d40A85
Engine	0x41f6309ff01e5ee663e7c7a3efce77843e43d935
Treasury	0xD5Ad95D47fE88D8DDCcda7ABC4EEE48A2A5c3cDe
CexVesting	0xAc5f6B2A80d462C34569FA5Da299ae01F275262e
DevVesting	0x953CD4e021Dad7Db0d171472935EC8D5bB77C5cb
Investor Vesting	0x378BF335bB0043A4e7dd6510f89a456B96dCb3b1
MarkVesting	0x26c2150569166E247400585BA87041EFF1cdA151
Withdrawal	0xd30AF269Cc1d0b223D986E136663Ac0D44f34A77

4.3 Findings Count

Severity	Count
Critical	01
High	01
Medium	01
Low	10
Gas	02
Total	15

4.4 Findings Summary

Overall, the code is well-written. During the review, a total of 15 issues were found. Out if which 2 were Critical and High issues.

ID	Title	Severity	Status

ID	Title	Severity	Status
[C-01]	Centralization risk whereby owner can withdraw all the vested `FLAT` tokens	Critical	Acknowledged
[H-01]	Swap functions lack slippage protection	High	Acknowledged
[M-01]	Pool fee tiers hardcoded to `3000` limiting swap flexibility	Medium	Acknowledged
[L-01]	User supplied deadline for swaps is not implemented correctly	Low	Acknowledged
[L-02]	Missing storage gap in upgradable contracts	Low	Acknowledged
[L-03]	Missing validation for admin status changes	Low	Acknowledged
[L-04]	Missing validation in allowlist management functions	Low	Acknowledged
[L-05]	Incorrect condition logic	Low	Acknowledged
[L-06]	Use of `transfer`/`transferFrom`	Low	Acknowledged
[L-07]	Floating/Outdated pragma	Low	Acknowledged
[L-08]	use `Ownable2step`	Low	Acknowledged
[L-09]	Missing event emission	Low	Acknowledged
[L-10]	Missing zero address validation	Low	Acknowledged
[G-01]	Long `require` statements	Gas	Acknowledged
[G-02]	Use `++i` instead of `i++`	Gas	Acknowledged

5. Findings

5.1 Critical Risk

[C-01] Centralization risk whereby owner can withdraw all the vested `FLAT` tokens

Status: Acknowledged.

Context:

- CEXvesting.sol#L153-L156
- DevVesting.sol#L104-L107
- InvestorVesting.sol#L166-L169
- MarkVesting.sol#L96-L99

Description: The `withdrawTokens` function allows the owner to withdraw any amount of `FLAT` tokens from the contract. This can reduce or fully drain the token balance, preventing beneficiaries from claiming their vested tokens. If the contract lacks sufficient funds, users may lose their vested tokens permanently.

```
function withdrawTokens(address to, uint256 amount) external onlyOwner {
        totalVestedAmount -= amount;
        flatToken.transfer(to, amount);
    }
```

Impact: Users may face delays in receiving their vested tokens.

Recommendation: It is recommended to implement a check to ensure the owner cannot withdraw more then the `totalAllocatedAmount`.

Team Response

Owner is trusted so this is not an issue.

5.2 High Risk

[H-01] Swap functions lack slippage protection

Status: Acknowledged.

Context:

- FLATEngine.sol#153
- FLATEngine.sol#172

Description: The `swapExactInputSingle` and `swapExactInputMultihop` functions set `amountOutMinimum` to 0, which disables slippage protection. Without a minimum output amount, swaps can be executed at any price, regardless of how unfavorable it may be. This exposes users to MEV sandwich attacks and significant value loss during high volatility or low liquidity conditions.

Impact: Malicious actors can front-run transactions and manipulate pool prices, forcing trades to execute at extremely unfavorable rates. Users could receive significantly fewer tokens than expected, potentially losing a large portion of their funds. MEV bots can sandwich these transactions, extracting value from every swap. Recommendation: Add a parameter for minimum output amount and enforce slippage protection in both functions.

Recommendation: Add a parameter for minimum output amount and enforce slippage protection in both functions.

```diff
ISwapRouter.ExactInputSingleParams memory params =
ISwapRouter.ExactInputSingleParams({
   // ...
-  amountOutMinimum: 0,
+  amountOutMinimum: minAmountOut,
   // ...
});
```

Team Response

To execute sandwich attack, Attacker have to sell bought tokens at higher price so they can make profit. We are not giving option user to swap flat token so there is no issue of slippage. Cause there is no profit for attacker to do sandwich attack.

5.3 Medium Risk

[M-01] Pool fee tiers hardcoded to `3000` limiting swap flexibility

Status: Acknowledged.

Context:

- FLATEngine.sol#168

Description: The `swapExactInputSingle` function hardcodes the Uniswap V3 pool fee to 3000 (0.3%) in the ExactInputSingleParams struct. While 0.3% is a common fee tier, hardcoding this value restricts the contract's ability to utilize pools with different fee tiers like 0.05% (500) or 1% (10000). This design choice limits the contract's versatility in interacting with various Uniswap V3 liquidity pools.

Impact: Users are restricted to pools with 0.3% fees only, preventing access to potentially more cost-efficient pools or pools that only exist in other fee tiers. This could result in higher transaction costs or failed transactions if the desired trading pair isn't available in the 0.3% fee tier.

Recommendation: Make the fee parameter configurable by modifying the function to accept a fee parameter.

```
- function swapExactInputSingle(uint256 amountIn) internal returns (uint
amountOut) {
+ function swapExactInputSingle(uint256 amountIn, uint24 poolFee) internal
returns (uint amountOut) {
       TransferHelper.safeApprove(usdc, address(swapRouter), amountIn);
       ISwapRouter.ExactInputSingleParams memory params =
ISwapRouter.ExactInputSingleParams({
           tokenIn: usdc,
           tokenOut: spot,
-           fee: 3000,
+           fee: poolFee,
           recipient: address(treasuryContract),
           deadline: block.timestamp + 10 minutes,
           amountIn: amountIn,
           amountOutMinimum: 0,
           sqrtPriceLimitX96: 0
       });
       amountOut = swapRouter.exactInputSingle(params);
   }
```

Team Response

I don't think this is the issue for what we should change the deployed contracts. cause another pool is charging 1% fees which is more than what we have executed.

5.4 Low Risk

[L-01] User supplied deadline for swaps is not implemented correctly

Status: Acknowledged.

Context:

- FLATEngine.sol#150
- FLATEngine.sol#170

Description: In the `swapExactInputMultihop` and `swapExactInputSingle` function, swaps are executed with a deadline of `block.timestamp + 10 minutes`. This deadline will always be valid when the transaction is included in a block, as the hardcoded addition of 10 minutes to the current timestamp serves no protective purpose. The timestamp of execution will always be `block.timestamp`, making this deadline check ineffective.

Impact: The ineffective deadline mechanism means swaps could be executed under significantly different market conditions than when the transaction was submitted. This could lead to trades being executed at unfavorable prices if the transaction remains pending in the mempool for extended periods.

Recommendation: Allow users to specify their own deadline parameter when initiating swaps.

```
- function swapExactInputMultihop(uint256 amountIn, address token)
internal returns (uint256 amountOut) {
+ function swapExactInputMultihop(uint256 amountIn, address token, uint256
deadline) internal returns (uint256 amountOut) {
        TransferHelper.safeApprove(token, address(swapRouter), amountIn);
        ISwapRouter.ExactInputParams memory params =
ISwapRouter.ExactInputParams({
            path: abi.encodePacked(token, uint24(3000), usdc, uint24(3000),
spot),
            recipient: address(treasuryContract),
-            deadline: block.timestamp + 10 minutes,
+            deadline: deadline,
            amountIn: amountIn,
            amountOutMinimum: 0
        });
        amountOut = swapRouter.exactInput(params);
    }
```

Team Response

Are not the issues for which we should change the deployed smart contracts.

[L-02] Missing storage gap in upgradable contracts

Status: Acknowledged.

Context:

- CEXvesting.sol
- DevVesting.sol
- FLAT.sol
- FLATEngine.sol
- InvestorVesting.sol
- MarkVesting.sol
- Treasury.sol
- Withdrawl.sol

Description: The contract inherits from UUPSUpgradeable but fails to include storage gap variables. In upgradeable contracts, adding new state variables in future versions can override existing storage slots of the child contracts. When a contract is upgraded, the storage layout must remain consistent to prevent data corruption. Without storage gaps, new variables added to the parent contract in an upgrade will shift the storage slots of the child contract, causing data to be read from or written to incorrect slots.

Impact: If there is no gap in storage, new variables added to the Vault contract could overwrite the beginning of the storage layout. This could lead to unwanted behavior and serious security issues

Recommendation: Add a storage gap to the contract to reserve storage slots for future use and prevent storage collisions.

Team Response

If we add new state variable at last than it will not affect storage slots in bad way.

[L-03] Missing validation for admin status changes

Status: Acknowledged.

Description: The `setAdmin` and `removeAdmin` functions directly modify admin status without first validating the current state. When setting a new admin, the function does not check if the address is already an admin. Similarly, when removing an admin, there's no verification that the address is currently an admin. This allows redundant state changes that emit misleading events and waste gas.

Impact: While this doesn't pose a direct security risk, it leads to unnecessary gas consumption.

Recommendation: Add validation checks before modifying admin status.

```
function setAdmin(address _admin) public onlyOwner {
+       require(!isAdmin[_admin], "Address is already admin");
       isAdmin[_admin] = true;
   }

function removeAdmin(address _admin) public onlyOwner {
+       require(isAdmin[_admin], "Address is not admin");
       isAdmin[_admin] = false;
   }
```

Team Response

Are not the issues for which we should change the deployed smart contracts.

[L-04] Missing validation in allowlist management functions

Status: Acknowledged.

Context: Contract.sol#L160-L165

Description: The `addToAllowlist` and `removeFromAllowlist` functions modify allowlist status without checking the current state. The functions don't verify if an address is already allowlisted before adding it, or if it's actually on the allowlist before removal. This allows redundant operations that consume gas without making effective state changes.

Impact: While this doesn't pose a direct security risk, it leads to unnecessary gas consumption.

Recommendation: Include validation checks before modifying allowlist status.

Team Response

Are not the issues for which we should change the deployed smart contracts.

[L-05] Incorrect condition logic

Status: Acknowledged.

Context:

- MarkVesting.sol#L79

Description: The `if` statement in the `release()` function uses an "and" condition (`&&`) to check if the `amount` is zero and greater than the contract’s token balance. This logic flaw means that if either condition is not met, the transaction will proceed, even if one of the conditions that should trigger a revert is satisfied.

Impact: This incorrect conditional check may allow transactions to execute when there are no tokens to release or when the token balance is insufficient.

Recommendation: Replace the && operator with an || operator so that the transaction reverts if either the amount is zero or it exceeds the contract's token balance.

```diff
- if (amount == 0 && amount > flatToken.balanceOf(address(this))) {
+ if (amount == 0 || amount > flatToken.balanceOf(address(this))) {
```

[L-06] Use of `transfer`/`transferFrom`

Status: Acknowledged.

Description: Not all ERC20 implementations are well behaved and revert on failure. Some simply return false and some might not return anything at all (USDT on mainnet).

Recommendation: Consider using OpenZeppelin `SafeERC20`s `safeTransfer` and `safeTransferFrom` to be as safe as possible for any atypical ERC20 tokens.

[L-07] Floating/Outdated pragma

Status: Acknowledged.

Description: All contracts accross the codebase use the following pragma statement:

```
pragma solidity ^0.8.24;
pragma solidity ^0.8.24;
```

Contracts should be deployed with the same compiler version and flags used during development and testing. An outdated pragma version might introduce bugs that affect the contract system negatively or recent compiler versions may have unknown security vulnerabilities.

Recommendation: It is recommended to lock the pragma to a latest and specific version of the compiler.

[L-08] use `Ownable2step`

Status: Acknowledged.

Description: The "Ownable2Step" pattern adds safety to smart contract ownership transfers. The new owner must accept before the transfer is complete, preventing mistakes.

Recommendation: It is recommended to use `Ownable2StepUpgradeable`.

[L-09] Missing event emission

Status: Acknowledged.

Context:

- MarkVesting.sol#L96-L99
- MarkVesting.sol#L101-L103
- Withdrawl.sol#L72-L74
- Withdrawl.sol#L77-L79
- InvestorVesting.sol#L158-L160
- InvestorVesting.sol#L166-L169
- FLATEngine.sol#L217-L219
- FLATEngine.sol#L224-L226
- FLATEngine.sol#L232-L234
- FLATEngine.sol#L240-L242
- FLATEngine.sol#L248-L250
- FLATEngine.sol#L270-L272
- FLATEngine.sol#L278-L280
- FLAT.sol#L61-L63
- AFLAT.sol#L251-L253
- AFLAT.sol#L255-L257
- AFLAT.sol#L260-L262
- Treasury.sol#L71-L74
- Treasury.sol#L77-L80
- Treasury.sol#L107-L109
- DevVesting.sol#L104-L107
- DevVesting.sol#L109-L111

Description: Important functions are missing event emissions. Without events, it's hard to track owner changes off-chain, making audits and monitoring difficult.

Recommendation: Emit an event for critical parameter changes.

[L-10] Missing zero address validation

Status: Acknowledged.

Description: The contracts set new addresses without checking for zero addresses. If a zero address is used, the contract may stop working, or tokens could be lost forever.

Recommendation: Add zero address validation to all the instances where addresses are being set.

5.5 Gas Optimizations

[G-01] Long `require` statements

Status: Acknowledged.

Context:

- Withdrawl.sol#L106-L106
- Withdrawl.sol#L109-L109
- InvestorVesting.sol#L142-L142

Description: The `require` statements use a string to show errors when validation fails. If the string is over 32 bytes, it needs extra memory storage and calculations. Keeping it under 32 bytes saves gas.

Recommendation: Keep strings in `require`statements under 32 bytes. This reduces gas costs during deployment and when the check runs.

[G-02] Use `++i` instead of `i++`

Status: Acknowledged.

Context:

- AFLAT.sol#L180
- AFLAT.sol#L191
- Treasury.sol#L59

Description: `++i` costs less gas than `i++`, especially when it's used in for loops.