StakedFLIP Security Review

Reviewer
Hans

October 27, 2023

Contents

1 Executive Summary

Over the course of 5 business days in total, Staked FLIP engaged with Hans to review stflip-contracts.

Summary

Type of Project	Defi
Timeline	9th Oct, 2023 - 13rd Oct, 2023
Methods	Manual Review

A comprehensive security review identified a total of 9 issues and 8 Gas optimization suggestions.

Repository	**Initial Commit**
Staked FLIP	f2b55518d6e65342ff55773f456dc048273a758c

Total Issues

High Risk	0
Medium Risk	3
Low Risk	3
Informational	3
Gas Optimization	8

The reported vulnerabilities were addressed by the Staked FLIP team, and the mitigation underwent a review process and was verified by Hans.

2 Scope of the Audit

This audit was conducted for 8 files in `src/token` folder and `src/utils` folder.

3 About Hans

Hans is an esteemed security analyst in the realm of smart contracts, boasting a firm grounding in mathematics that has sharpened his logical abilities and critical thinking skills. These attributes have fast-tracked his journey to the peak of the Code4rena leaderboard, marking him as the number one auditor in a record span of time. In addition to his auditor role, he also serves as a judge on the same platform. Hans' innovative insight is evident in his creation of Solodit, a vital resource for navigating consolidated security reports. In addition, he is a co-founder of Cyfrin, where he is dedicated to enhancing the security of the blockchain ecosystem through continuous efforts.

4 Disclaimer

I endeavor to meticulously identify as many vulnerabilities as possible within the designated time frame; however, I must emphasize that I cannot accept liability for any findings that are not explicitly documented herein. It is essential to note that my security audit should not be construed as an endorsement of the underlying business or product. The audit was conducted within a specified timeframe, with a sole focus on evaluating the security aspects of the solidity implementation of the contracts.

While I have exerted utmost effort in this process, I must stress that I cannot guarantee absolute security. It is a well-recognized fact that no system can be deemed completely impervious to vulnerabilities, regardless of the level of scrutiny applied.

5 Protocol Summary

Chainflip is a cross-chain DEX for performing native cross-chain swaps by utilizing a 150 validator network with a threshold signature scheme to control addresses on supported chains. StakedFLIP is a liquid staking token for the Chainflip network. Users can stake or purchase a greater amount of stFLIP than FLIP given. This FLIP will then be staked to validators. As validators accrue rewards, StakedFLIP will rebase to distribute protocol rewards to holders (stFLIP is always backed 1:1 by stFLIP).

6 Additional Comments

The protocol relies on the trusted off-chain actor in managing the operators and validators. Concerns were raised regarding the "correctness" of the input data from the off-chain actor.

The security assessment was carried out with a narrow focus on the contracts. Due to time limitations and the incremental nature of the reviews, the results might not be comprehensive and might not represent the complete security profile of the protocol.

7 Findings

7.1 Medium Risk

7.1.1 There is no option to change a `whitelisted` state of operators.

Severity: Medium

Context: OutputV1.sol#L165

Description: When the admin adds new operators, they are whitelisted by default but there is no option to change later.

```
function addOperator(address manager, string calldata name, uint256 serviceFeeBps, uint256
↪ validatorFeeBps, uint256 validatorAllowance) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(serviceFeeBps + validatorFeeBps <= 10000, "Output: fees must be less than 100%");
    Operator memory operator = Operator(0, 0, SafeCast.toUint16(serviceFeeBps),
    ↪ SafeCast.toUint16(validatorFeeBps),true, SafeCast.toUint8(validatorAllowance), manager,
    ↪ manager,name);
    operators.push(operator);
}
```

`operators[operatorId].whitelisted` is used in `addValidators()` and `fundValidators()`, and all operators won't be blacklisted once they are added.

Impact The whitelist mechanism wouldn't work as expected.

Recommendation: Recommend adding a function like `setOperatorWhitelisted()`.

Client: Added `setOperatorWhitelist()` as recommended.

Hans: Verified at commit 44c8031.

7.1.2 Possible revert in `RebaserV1._updateOperator()`

Severity: Medium

Context: RebaserV1.sol#L201

Description: During the `previousBalance` calculation, it might revert due to underflow like the below scenario.

```
function _updateOperator(uint256 operatorBalance, uint256 operatorId, bool takeFee) internal
↪ returns (uint256) {
    uint256 rewardIncrement;
    uint96 staked;
    uint96 unstaked;
    uint16 serviceFeeBps;
    uint16 validatorFeeBps;
    uint256 previousBalance;
    (staked,unstaked,serviceFeeBps, validatorFeeBps) =
    ↪ wrappedOutputProxy.getOperatorInfo(operatorId);

    uint256 slashCounter_ = operators[operatorId].slashCounter;
    previousBalance = staked + operators[operatorId].rewards - unstaked - slashCounter_;
```

1. At the first time, we assume `staked = 100, unstaked = 0, rewards = 0, slashCounter = 0`.
2. A validator earned 5 rewards and the whole amounts are unstaked using `OutputV1.redeemValidators()`. Then `staked = 100, unstaked = 105, rewards = 0, slashCounter = 0`.
3. During the next rebasing, `previousBalance` will be `staked + operators[operatorId].rewards - unstaked - slashCounter_ = 100 + 0 - 105 - 0 = -5` and `_updateOperator()` will revert due to underflow.

This situation can be resolved only if the manager adds more funds to that operator using `fundValidators()`.

Impact The rebasing logic wouldn't work as expected due to underflow.

Recommendation: We should modify `_updateOperator()` to handle negative balances properly without reverting.

Client: We split the `previousBalance` into `positivePreviousBalanceComponent` (the sum of the staked and rewards counters) and the `negativePreviousBalanceComponent` (the sum of the unstaked and slash counter). We rearrange our inequality to check whether there are rewards while maintaining mathematical equality with the original statement. We also add an additional case for `rewardIncrement` when the negative component exceeds the positive.

Hans: Verified at commit 07d5b1f.

7.1.3 The redemption invariant might be broken if the output contract doesn't have enough `FLIP`

Severity: Medium

Context: BurnerV1.sol#L151

Description: After users request to redeem their funds, `redeem()` handles them after checking some invariants in `_redeemable()`.

```
    /**
    * @notice is a burn redeemable
    * @param burnId The id of the burn to check
    * @dev Firstly, burn can obviously not be redeemable if it has already been redeemed.
    * Secondly, we ensure that there is enough FLIP to satisfy all prior burns in the burn queue,
    * and the burn of `burnId` itself. `Sums[burnId]` is the sum of all burns up to and including
    ↪   `burnId`.
    * redeemed is the sum of all burns that have been redeemed. If the difference between the two is <=
    ↪   than the
    * balance of FLIP in the contract, then the burn is redeemable.
     */
    function _redeemable(uint256 burnId) internal view returns (bool) {
        uint256 difference = sums[burnId] < redeemed ? 0 : sums[burnId] - redeemed;
        return burns[burnId].completed == false && difference <= flip.balanceOf(address(output));
    }
```

The second assumption is `there is enough FLIP to satisfy all prior burns in the burn queue` but it might be broken like the below scenario.

1. Let's assume there are 2 valid burn requests. `burns.amount = {100, 100}`, `sums = {100, 200}`, `FLIP balance in output = 200, redeemed = 0`.
2. The second burner calls `redeem(2)`. It works properly and `redeemed = 100` now.
3. After that, `OutputV1.fundValidators()` is called and there are no FLIP in the output contract.
4. When the first burner calls `redeem(1)`, `uint256 difference = sums[burnId] < redeemed ? 0 : sums[burnId] - redeemed` will return 0 because `sums[1] = redeemed = 100` and `_redeemable()` will return true.
5. But `redeem()` will revert due to the lack of FLIP balance.

It's because the `MANAGER_ROLE` has transferred all funds using `OutputV1.fundValidators()`.

As we can see from `RebaserV1.claimFee()/claimServiceFee()`, `OutputV1` contract contains operator/service fees that might be claimed anytime and these amounts should be checked during the validation.

Impact Earlier burners wouldn't be able to redeem their funds after the latter ones have done.

Recommendation: We should add a validation in `OutputV1.fundValidators()` to make sure the output contract keep enough balances for the prior burns.

Client: Added a relevant validation in `fundValidators()` as recommended.

Hans: Verified at commit 842d8e1.

7.2 Low Issues

7.2.1 Not following the Checks-Effects-Interaction pattern

Context: BurnerV1.sol#L78 RebaserV1.sol#L272 RebaserV1.sol#L294

Description: Some functions `redeem()`, `claimFee()`, `claimServiceFee()` don't follow the CEI pattern.

Solidity recommends the usage of the Check-Effects-Interaction Pattern to avoid potential security issues, such as reentrancy.

Currently, there is no direct fund loss because `FLIP/stFLIP` tokens don't have any callbacks/hooks but it should be mitigated for safety.

Impact A reentrancy attack might be possible if the protocol uses other tokens later.

Recommendation: Recommend modifying the contract's state before making any external calls.

Client: Modified `redeem`, `claimFee`, `claimServiceFee` to have fund transfer as the last call.

Hans: Verified at commit a9f4d7b.

7.2.2 Admin-level vulnerabilities

Context: stFlip.sol#L112 RebaserV1.sol#L91 RebaserV1.sol#L100 RebaserV1.sol#L108

Description: In protocol, some main settings are set by an admin without any validations. A single private key may be taken in a hack, or the sole holder of the key may become unable to retrieve the key when necessary, or the single owner may become malicious and perform a rug-pull. Although we assume the admin is trusted, users should acknowledge it before interacting with the protocol.

Impact The admin can change the protocol's behavior in unexpected ways.

Recommendation: The centralization risk exists in most protocols and the protocol documentation should include a clear explanation about that.

Client: The admin is a multisig that is controlled by `fraxGovernorOmega`. Holders have the right to veto all transactions that come out of the multisig.

Hans: Acknowledged.

7.2.3 Wrong comment

`burn()` doesn't transfer FLIP tokens from `msg.sender`.

```
File: stflip-contracts\src\utils\BurnerV1.sol
    /**
     * @notice Burns stflip tokens, transfers FLIP tokens from msg.sender, adds entry to burns/sums list//@audit wrong comment
     * @param to, the owner of the burn, the address that will receive the burn once completed
     * @param amount, the amount to burn
     */
    function burn(address to, uint256 amount) external returns (uint256) {
```

Client: Fixed the comment.

Hans: Verified at commit 6dc6660.

7.3 Informational Findings

7.3.1 `2**256 - 1` should be rewritten as `type(uint256).max`

```
File: stflip-contracts\src\utils\AggregatorV1.sol
        if (liquidityPool_ != address(0)) {
            flip.approve(address(liquidityPool_), 2**256-1);
        }
        flip.approve(address(minter), 2**256-1);
        stflip.approve(address(burner), 2**256-1);
        stflip.approve(address(liquidityPool_), 2**256-1);

File: stflip-contracts\src\utils\AggregatorV1.sol
        flip.approve(address(canonicalPool), 2**256 - 1);
        stflip.approve(address(canonicalPool), 2**256 - 1);


File: stflip-contracts\src\utils\OutputV1.sol
        flip.approve(address(rebaser_), 2**256-1);
        flip.approve(address(burnerProxy_), 2**256 - 1);
        flip.approve(address(stateChainGateway), 2**256 - 1);
```

Client: Changed to `type(uint256).max` as recommended.

Hans: Verified at commit e6504ed.

7.3.2 Improper naming

`slashCounter` saves the slashed FLIP amount but it's not easy to understand the meaning of the name. `slashedAmount` would be better because `counter` is usually used for counting in integers.

```
File: stflip-contracts\src\utils\RebaserV1.sol
    struct Operator {
        uint88 rewards;            // uint88 sufficient
        uint80 pendingFee;         // uint80 sufficient
        uint88 slashCounter;       // uint88 sufficient //@audit improper naming
    }
```

Client: I called it a counter because it keeps track of the amount of slash an operator has to pay off prior to earning fees again.

Hans: Acknowledged.

7.3.3 Event is not properly `indexed`

Index event fields make the field more quickly accessible to off-chain tools that parse events. This is especially useful when it comes to filtering based on an address. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Where applicable, each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question.

```
File: stflip-contracts\src\token\stFlip.sol
35:     event Rebase(uint256 epoch, uint256 prevYamsScalingFactor, uint256 newYamsScalingFactor, uint256 rebaseInterval);


File: stflip-contracts\src\utils\BurnerV1.sol
53:     event Burn(address burner, address recipient, uint256 amount, uint256 burnId); // emits the person who sent burn tx along with the recipient, amount and ID


File: stflip-contracts\src\utils\RebaserV1.sol
51:     event FeeClaim(address feeRecipient, uint256 amount, bool receivedFlip, uint256 operatorId);
52:     event RebaserRebase(uint256 apr, uint256 stateChainBalance, uint256 previousSupply, uint256 newSupply);
53:
```

Client: Fixed as recommended.

Hans: Verified at commit c6812ea.

7.4 Gas Optimizations

7.4.1 Using bools for storage incurs overhead

Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas), and to avoid Gsset (20000 gas) when changing from 'false' to 'true', after having been 'true' in the past. See source.

Instances (1):

```
File: src/token/tStorage.sol

31:     bool public frozen;
```

Client: I am going to leave it as is since the gas benefit is just for writes. The idea is to never have to use the function and would rather not have to think about an unintuitive 1 vs. 2 in a stressful time.

Hans: Acknowledged.

7.4.2 Cache array length outside of loop

If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

Instances (12):

```
File: src/utils/BurnerV1.sol

98:         for (uint256 i = 0; i < burns.length; i++) {

132:         for (uint256 i = 0; i < burnIds.length; i++) {
```

```
File: src/utils/OutputV1.sol

97:        for (uint256 i = 0; i < addresses.length; i++) {

122:        for (uint256 i = 0; i < addresses.length; i++) {

134:        for (uint256 i = 0; i < addresses.length; i++) {

146:        for (uint256 i = 0; i < addresses.length; i++) {

192:        for (uint i = 0; i < addresses.length; i++) {

212:        for (uint i = 0; i < addresses.length; i++) {

280:        ValidatorInfo[] memory validatorInfo = new ValidatorInfo[](addresses.length);

281:        for (uint256 i = 0; i < addresses.length; i++) {

288:        return (validatorInfo, operators.length, addressesEqual);
```

```
File: src/utils/RebaserV1.sol

164:        for (uint i = 0; i < validatorInfo.length; i++) {
```

Client: Fixed as recommended.

Hans: Verified at commit bf47b84.

7.4.3 Use Custom Errors

Source Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost.

Instances (24):

```
File: src/token/stFlip.sol

59:        require(frozen==false, "frozen");

135:        require(nextYamScalingFactor <= _maxScalingFactor(), "max scaling factor too low");

200:        require(nextYamScalingFactor <= _maxScalingFactor(), "max scaling factor too low");

282:        require(value < _maxScalingFactor(), "stFLIP: rebaseFactor too large");
```

```
File: src/utils/AggregatorV1.sol

176:            require(attempts > 0, "Aggregator: no attempts left");
```

```
File: src/utils/BurnerV1.sol

75:        require(_redeemable(burnId), "Burner: not redeemable. either already claimed or
↪  insufficient balance");
```

```
File: src/utils/OutputV1.sol

93:         require(operators[operatorId].manager == msg.sender, "Output: not manager of operator");

94:         require(operators[operatorId].whitelisted == true, "Output: operator not whitelisted");

95:         require(operatorId != 0, "Output: cannot add to null operator");

98:             require(validators[addresses[i]].operatorId == 0, "Output: validator already added");

164:         require(serviceFeeBps + validatorFeeBps <= 10000, "Output: fees must be less than 100%");

188:         require(addresses.length == amounts.length, "lengths must match");

195:             require(validator.whitelisted == true, "Output: validator not whitelisted");

196:             require(operators[operatorId_].whitelisted == true, "Output: operator not
↪   whitelisted");

224:         require(serviceFeeBps + validatorFeeBps <= 10000, "Output: fees must be less than 100%");
```

```
File: src/utils/RebaserV1.sol

127:         require(timeElapsed >= rebaseInterval, "Rebaser: rebase too soon");

128:         require(validatorBalances.length == addresses.length, "Rebaser: length mismatch");

161:         require(addressesEqual, "Rebaser: validator addresses do not match");

162:         require(validatorBalances.length == addresses.length, "Rebaser: length mismatch");

235:             require(apr + 1 < aprThresholdBps, "Rebaser: apr too high");

237:             require(10000 - (newSupply * 10000 / currentSupply) < slashThresholdBps, "Rebaser:
↪   supply decrease too high");

261:         require(max == true || amount <= pendingFee, "Rebaser: fee claim requested exceeds pending
↪   fees");

262:         require(msg.sender == feeRecipient || msg.sender == manager, "Rebaser: not fee recipient
↪   or manager");

284:         require(max == true || amount <= servicePendingFee, "Rebaser: fee claim requested exceeds
↪   pending fees");
```

Client: Fixed as recommended.

Hans: Verified at commit 1a7f1c8.

7.4.4 Don't initialize variables with default value

Instances (16):

```
File: src/utils/AggregatorV1.sol

163:         uint256 first = 0;

164:         uint256 mid = 0;
```

```
File: src/utils/BurnerV1.sol

96:         uint256 t = 0;

98:         for (uint256 i = 0; i < burns.length; i++) {

106:         for (uint256 i = 0; i < t; i++) {

132:         for (uint256 i = 0; i < burnIds.length; i++) {
```

```
File: src/utils/OutputV1.sol

97:         for (uint256 i = 0; i < addresses.length; i++) {

122:         for (uint256 i = 0; i < addresses.length; i++) {

134:         for (uint256 i = 0; i < addresses.length; i++) {

146:         for (uint256 i = 0; i < addresses.length; i++) {

192:         for (uint i = 0; i < addresses.length; i++) {

212:         for (uint i = 0; i < addresses.length; i++) {

281:         for (uint256 i = 0; i < addresses.length; i++) {

300:         for (uint i = 0; i < length; i++) {

309:         for (uint i = 0; i < count; i++) {
```

```
File: src/utils/RebaserV1.sol

164:         for (uint i = 0; i < validatorInfo.length; i++) {
```

Client: Fixed as recommended.

Hans: Verified at commit e47046b.

7.4.5 Long revert strings

Instances (10):

```
File: src/utils/BurnerV1.sol

75:         require(_redeemable(burnId), "Burner: not redeemable. either already claimed or
↪  insufficient balance");
```

```
File: src/utils/OutputV1.sol

95:         require(operatorId != 0, "Output: cannot add to null operator");

164:         require(serviceFeeBps + validatorFeeBps <= 10000, "Output: fees must be less than 100%");

195:             require(validator.whitelisted == true, "Output: validator not whitelisted");

224:         require(serviceFeeBps + validatorFeeBps <= 10000, "Output: fees must be less than 100%");
```

```
File: src/utils/RebaserV1.sol

161:         require(addressesEqual, "Rebaser: validator addresses do not match");

237:             require(10000 - (newSupply * 10000 / currentSupply) < slashThresholdBps, "Rebaser:
↪   supply decrease too high");

261:         require(max == true || amount <= pendingFee, "Rebaser: fee claim requested exceeds pending
↪   fees");

262:         require(msg.sender == feeRecipient || msg.sender == manager, "Rebaser: not fee recipient
↪   or manager");

284:         require(max == true || amount <= servicePendingFee, "Rebaser: fee claim requested exceeds
↪   pending fees");
```

Client: Mitigated using custom errors.

Hans: Verified at commit 1a7f1c8.

7.4.6 `++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i`/`i--` too)

Saves 5 gas per loop

Instances (17):

```
File: src/utils/BurnerV1.sol

98:         for (uint256 i = 0; i < burns.length; i++) {

101:                t++;

106:         for (uint256 i = 0; i < t; i++) {

132:         for (uint256 i = 0; i < burnIds.length; i++) {
```

```
File: src/utils/OutputV1.sol

97:        for (uint256 i = 0; i < addresses.length; i++) {

122:        for (uint256 i = 0; i < addresses.length; i++) {

134:        for (uint256 i = 0; i < addresses.length; i++) {

146:        for (uint256 i = 0; i < addresses.length; i++) {

192:        for (uint i = 0; i < addresses.length; i++) {

212:        for (uint i = 0; i < addresses.length; i++) {

281:        for (uint256 i = 0; i < addresses.length; i++) {

300:        for (uint i = 0; i < length; i++) {

303:                countableAddresses_[count++] = validatorToCheck;

309:        for (uint i = 0; i < count; i++) {
```

```
File: src/utils/RebaserV1.sol

164:        for (uint i = 0; i < validatorInfo.length; i++) {

171:        for (operatorId = 1; operatorId < operatorCount; operatorId++) {

305:        for (uint256 operatorId; operatorId < operatorCount; operatorId++) {
```

Client: Fixed as recommended.

Hans: Verified at commit ab6673b.

7.4.7 Use shift Right/Left instead of division/multiplication if possible

Instances (1):

```
File: src/utils/AggregatorV1.sol

178:            mid = (last+first) / 2;
```

Client: Fixed as recommended.

Hans: Verified at commit 3bb4626.

7.4.8 Use != 0 instead of > 0 for unsigned integer comparison

Instances (6):

```
File: src/utils/AggregatorV1.sol

78:         if (amountInstantBurn > 0) {

83:         if (amountBurn > 0) {

87:         if (amountSwap > 0) {

109:         if (amountSwap > 0){

113:         if (mintAmount > 0) {

176:             require(attempts > 0, "Aggregator: no attempts left");
```

Client: Fixed as recommended.

Hans: Verified at commit bf3fb53.