Smart Contract Security Audit Report

Collectif Finance

1. Contents

2. General Information

This report contains information about the results of the security audit of the Collectif Finance (hereafter referred to as "Customer") Liquid Staking smart contracts, conducted by Decurity in the period from 05/01/2023 to 05/15/2023.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the contracts in the following repository: https://github.com/collective-dao/liquid-staking. Initial review was done for the commit c1a41ed7a6ed3332b93c54acb4474cdcc13c0c8c and the re-testing was done for the commit bcbaed0cf9e153a06b48f3d7a200b91b50066750.

The following contracts have been tested:

- contracts/types/StorageProviderTypes.sol
- contracts/ClFIL.sol
- contracts/libraries/DateTimeLibraryCompressed.sol
- contracts/libraries/SafeTransferLib.sol
- contracts/libraries/BigInts.sol
- contracts/libraries/tokens/ERC20.sol
- contracts/libraries/tokens/IWFIL.sol
- contracts/libraries/tokens/ERC4626.sol
- contracts/StorageProviderRegistry.sol

- contracts/LiquidStaking.sol
- contracts/StorageProviderCollateral.sol
- contracts/interfaces/IStorageProviderCollateralClient.sol
- contracts/interfaces/IStorageProviderCollateral.sol
- contracts/interfaces/ILiquidStaking.sol
- contracts/interfaces/IStorageProviderRegistry.sol
- contracts/interfaces/ILiquidStakingClient.sol
- contracts/interfaces/IStorageProviderRegistryClient.sol

2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- User,
- Protocol owner,
- Liquidity Token owner/contract.

The table below contains sample attacks that malicious attackers might carry out.

Table. Theoretically possible attacks

Attack	Actor
Contract code or data hijacking *Deploying a malicious contract or submitting malicious data*	Contract owner Token owner
Financial fraud *A malicious manipulation of the business logic and balances, such as a re-entrancy attack or a flash loan attack*	Anyone
Attacks on implementation *Exploiting the weaknesses in the compiler or the runtime of the smart contracts*	Anyone

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

3. Summary

As a result of this work, we have discovered a few high-risk issues that could lead to the loss of funds.

The other suggestions included fixing the low-risk issues and some best practices (see Security Process Improvement).

The Collectif Finance team has given the feedback for the suggested changes and explanation for the underlying code.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of May 3, 2023.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
ERC4626 Inflation	LiquidStaking.sol	**High**	Fixed
_executeChangeBeneficiary will revert, because called not from appropriate contract.	BeneficiaryManager.sol	**High**	Fixed
`acceptBeneficiaryAddress` function lacks check that beneficiary has actually changed	StorageProviderRegistry.sol	**High**	Fixed
Loss of deposited collateral in withdraw()	contracts/StorageProviderCollateral.sol	**High**	Fixed
changeBeneficiaryAddress() function doesn't have any purpose	BeneficiaryManager.sol	**Medium**	Fixed
Global daily allocation	contracts/StorageProviderRegistry.sol	**Medium**	Fixed
Centralization risks	RewardCollector.sol	**Medium**	Acknowledged
Incorrect check in deactivateStorageProvider function	StorageProviderRegistry.sol	**Low**	Fixed
deactivateStorageProvider doesn't have to transfer beneficiary back to owner	StorageProviderRegistry.sol	**Low**	Fixed
Repayment can be less than the allocation limit	contracts/StorageProviderRegistry.sol	**Low**	Fixed
Incorrect pool requirements	liquid-staking/contracts/LiquidStaking.sol	**Low**	Fixed

Issue	Contract	Risk Level	Status
Re-registration changes totalStorageProviders and totalInactiveStorageProviders	contracts/StorageProviderRegistry.sol	Low	Fixed
Redundant ternary operator	contracts/StorageProviderCollateral.sol	Info	Fixed
The cached variable can be used	contracts/StorageProviderRegistry.sol	Info	Fixed
Unused imports	contracts/StorageProviderCollateral.sol	Info	Fixed
Extra variable initialization		Info	Fixed
The variable can be constant	contracts/RewardCollector.sol	Info	Fixed
String error messages are used instead of custom errors	contracts/StorageProviderCollateral.sol contracts/StorageProviderRegistry.sol contracts/LiquidStaking.sol	Info	Fixed
Unnecessary transfer in _wrapWETH9	contracts/LiquidStaking.sol	Info	Fixed

4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,

- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. ERC4626 Inflation

Risk Level: High

Status: Fixed in the commit a9c07f63.

Contracts:

- LiquidStaking.sol

Location: Function: `stake(), unstake(uint256 shares, address owner)`.

Description:

`LiquidStaking` contract is vulnerable to ERC4626 inflation attack. Attacker can manipulate a totalAssets() function, which will cause a victim to receive 1 share.

Attack example:

1. The hacker back-runs a transaction of an LiquidStaking pool creation.
2. The hacker mints for themself one share: deposit(1). Thus, totalAsset()==1, totalSupply()==1.
3. The hacker front-runs the deposit of the victim who wants to deposit 20,000 WFIL (20,000 * 1e18).
4. The hacker inflates the denominator right in front of the victim: WFIL.transfer(10_000e18). Now totalAsset()==10_000e18 + 1, totalSupply()==1.
5. Next, the victim's tx takes place. The victim gets 1 * 20_000e18 / (10_000e18 + 1) == 1 shares. The victim gets only one share, which is the same amount as the hacker has.
6. The hacker burns their share and gets half of the pool, which is approximately 30_000e18 / 2 == 15_000e18, so their profit is +5,000 (25% of the victim's deposit).

POC:

```
function testStakingAttack(uint256 amount) public {
        // initial balances
        hevm.deal(attacker, 10000 * 1e18 + 1);
        hevm.deal(alice, 20000 * 1e18);

        // stake 1 wei, get 1 share
        hevm.startPrank(attacker);
        staking.stake{value: 1}();
```

```
        // transfer 10k WFIL directly to staking
        wfil.deposit{value: 10000 * 1e18}();
        console.log("WFIL Balance of attacker:", wfil.balanceOf(attacker));
        wfil.transfer(address(staking), 10000 * 1e18);
        console.log("WFIL Balance of staking:",
wfil.balanceOf(address(staking)));
        hevm.stopPrank();

        // now alice deposits 20k and also gets 1 share
        hevm.prank(alice);
        console.log("Initial Alice Balance:", alice.balance);
        staking.stake{value: 20000 * 1e18}();

        // attacker withdraws 1 share
        hevm.prank(attacker);
        console.log("Attacker Balance Before Attack:", attacker.balance);
        staking.unstake(1, attacker);
        console.log("Attacker Balance After Attack:", attacker.balance);

        hevm.prank(alice);
        staking.unstake(1, alice);
        console.log("Alice balance after attack:", alice.balance);
}
```

This POC test can be included in `contracts/test/LiquidStaking.t.sol`

Remediation:

5.2. _executeChangeBeneficiary will revert, because called not from appropriate contract.

Risk Level: High

Status: Fixed in the commit 12863d59.

Contracts:

- BeneficiaryManager.sol

Location: Lines: 97. Function: `_executeChangeBeneficiary(CommonTypes.FilActorId, uint256, int64)`.

Description:

```
function _executeChangeBeneficiary(
        CommonTypes.FilActorId minerId,
        uint256 quota,
        int64 expiration
    ) internal virtual {
        MinerTypes.ChangeBeneficiaryParams memory params;
        params.new_beneficiary =
FilAddresses.fromEthAddress(resolver.getRewardCollector());
        params.new_quota = BigInts.fromUint256(quota);
        params.new_expiration = CommonTypes.ChainEpoch.wrap(expiration);

        MinerAPI.changeBeneficiary(minerId, params);
}
```

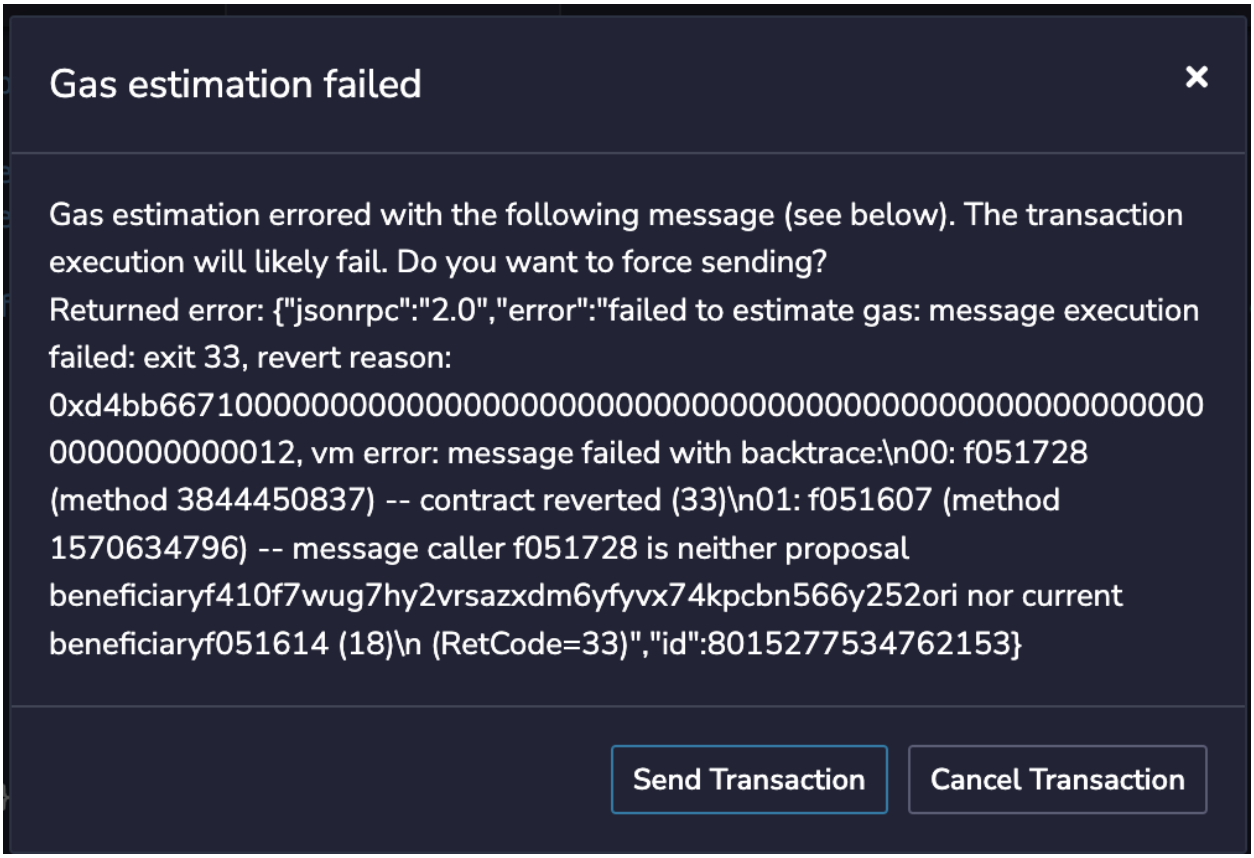
In current implementation `_executeChangeBeneficiary` is called from BeneficiaryManager address. However the address which needs to become beneficiary is the `RewardCollector` . This call will revert because in order to accept beneficiary address this function should be called directly from the address of new beneficiary (`RewardCollector` in this case).

Remediation:

Consider changing logic and calling `changeBeneficiary` directly from new beneficiary.

****POC:**

Trying to call `changeBeneficiary` not from the address of the new beneficiary will result in the following error:



References:

- https://github.com/filecoin-project/FIPs/blob/master/FIPS/fip-0029.md

5.3. `acceptBeneficiaryAddress` function lacks check that beneficiary has actually changed

Risk Level: High

Status: Fixed in the commit 4ab3ebd6.

Contracts:

- StorageProviderRegistry.sol

Location: Lines: 217. Function: `acceptBeneficiaryAddress(uint64 _ownerId)`.

Description:

The following function performs beneficiary acceptation, however it doesn't check that beneficiary has really changed before making SP active. Since changing beneficiary requires approve from both old beneficiary and the nominee, this may lead to a situation, when SP is active, however, their beneficiary is not the one that is required. They will be able to pledge, but it won't be possible to take pledge back from them.

```
function acceptBeneficiaryAddress(uint64 _ownerId) public virtual override
onlyAdmin nonReentrant {
        StorageProviderTypes.StorageProvider storage storageProvider =
storageProviders[_ownerId];
        if (!storageProvider.onboarded) revert InactiveSP();

        (bool isID, uint64 beneficiaryId) =
resolver.getRewardCollector().getActorID();
        if (!isID) revert InactiveActor();


IRewardCollectorClient(resolver.getRewardCollector()).forwardChangeBeneficiary
(
            storageProvider.minerId,
            beneficiaryId,
            allocations[_ownerId].repayment,
            storageProvider.lastEpoch
        );

        storageProvider.active = true;
        beneficiaryStatus[_ownerId] = true;

        emit StorageProviderBeneficiaryAddressAccepted(_ownerId);
    }
```

Remediation:

Consider checking that beneficiary has really changed via `getBeneficiary` method from `MinerAPI`.

References:

- https://github.com/filecoin-project/FIPs/blob/master/FIPS/fip-0029.md

5.4. Loss of deposited collateral in withdraw()

Risk Level: High

Status: Fixed in commit a14d557b.

Contracts:

- contracts/StorageProviderCollateral.sol

Location: Lines: 158. Function: `withdraw()`.

Description:

Storage provider has the ability to call the `withdraw()` function to withdraw funds from the `StorageProviderCollateral` contract. The amount of possible funds to withdraw is calculated through the `calcMaximumWithdraw()` function.

If the storage provider has an unpaid debt after the `updateCollateralRequirements()` or slashing, `calcMaximumWithdraw()` will subtract it from the available collateral amount.

Thus, the `finalAmount` variable always contains the amount of FIL tokens, that the storage provider can withdraw without any problems.

However, there is no transfer after its subtraction from `collaterals[ownerId].availableCollateral`. Storage Provider just losing the `finalAmount` of FIL tokens.

```
function withdraw(uint256 _amount) public nonReentrant {
        ...
        (uint256 lockedWithdraw, uint256 availableWithdraw, bool isUnlock) =
calcMaximumWithdraw(ownerId);
        uint256 maxWithdraw = lockedWithdraw + availableWithdraw;
        uint256 finalAmount = _amount > maxWithdraw ? maxWithdraw : _amount;
        uint256 delta;

        if (isUnlock) {
            delta = finalAmount - lockedWithdraw;
            collaterals[ownerId].lockedCollateral =
collaterals[ownerId].lockedCollateral - lockedWithdraw;
            collaterals[ownerId].availableCollateral =
collaterals[ownerId].availableCollateral - delta;

            _unwrapWFIL(msg.sender, finalAmount);
        } else { // isUnlock = false, because user had a dept
            collaterals[ownerId].availableCollateral =
collaterals[ownerId].availableCollateral - finalAmount;
            // No FIL transfer
        }
```

```
        emit StorageProviderCollateralWithdraw(ownerId, finalAmount);
    }
```

Remediation:

Consider fixing `withdraw()` function for the locked collateral case.

5.5. changeBeneficiaryAddress() function doesn't have any purpose

Risk Level: Medium

Status: Fixed in the commit 12863d59.

Contracts:

- BeneficiaryManager.sol

Location: Lines: 46. Function: `changeBeneficiaryAddress()`.

Description:

```
function changeBeneficiaryAddress() external virtual {
        address ownerAddr = msg.sender.normalize();

        (bool isID, uint64 ownerId) = ownerAddr.getActorID();
        if (!isID) revert InactiveActor();

        (, bool onboarded, address targetPool, uint64 minerId, int64
lastEpoch) = IRegistryClient(
            resolver.getRegistry()
        ).storageProviders(ownerId);

        if (!onboarded) revert InactiveSP();

        uint256 quota =
IRegistryClient(resolver.getRegistry()).getRepayment(ownerId);
        CommonTypes.FilActorId actorId = CommonTypes.FilActorId.wrap(minerId);

        MinerTypes.GetOwnerReturn memory ownerReturn =
MinerAPI.getOwner(actorId);
        if (keccak256(ownerReturn.proposed.data) != keccak256(bytes("")))
revert OwnerProposed();

        uint64 actualOwnerId =
PrecompilesAPI.resolveAddress(ownerReturn.owner);
        if (ownerId != actualOwnerId) revert InvalidOwner();
```

```
        _executeChangeBeneficiary(actorId, quota, lastEpoch);

        emit BeneficiaryAddressUpdated(msg.sender, minerId, targetPool, quota,
lastEpoch);
    }
```

The following function is probably supposed to propose new beneficiary from SP, however it doesn't act as intended. Firstly, new beneficiary can be proposed only from the owner address, which means that owner of the miner has to call propose beneficiary method directly. This function will act the same as `acceptBeneficiaryAddress` at the `StorageProviderRegistry` contract, except the fact it will not make SP active. This means that it will approve the nominee when changing beneficiary. Also, it will revert, because the address of nominee will not match with the caller.

Remediation:

This function can be removed, because it has no logic value.

References:

- https://github.com/filecoin-project/FIPs/blob/master/FIPS/fip-0029.md

5.6. Global daily allocation

Risk Level: Medium

Status: Fixed in the commit 864a4edef.

Contracts:

- contracts/StorageProviderRegistry.sol

Location: Lines: 439. Function: `increaseUsedAllocation()`.

Description:

Each provider has a daily allocation limit, which is the amount of FIL tokens that he can pledge per day.

However, the `StorageProviderRegistry` contract only keeps one shared counter for daily allocations from all storage providers. Providers may not be able to make a pledge if the limit has already been exceeded by someone else, as the counter is shared in the function `increaseUsedAllocation()`:

```
bytes32 dateHash = keccak256(abi.encodePacked(year, month, day));

uint256 usedDailyAlloc = dailyUsages[dateHash];
uint256 totalDailyUsage = usedDailyAlloc + _allocated; // getting global daily
used allocation

StorageProviderTypes.SPAllocation storage spAllocation =
allocations[_ownerId]; // getting daily limit for unique storage provider

// global usage is less, than private limit
require(totalDailyUsage <= spAllocation.dailyAllocation,
"DAILY_ALLOCATION_OVERFLOW");
```

Remediation:

Consider counting used allocation separately for every storage provider.

5.7. Centralization risks

Risk Level: Medium

Status: Acknowledged

Contracts:

- RewardCollector.sol

Location: Lines: `82, 129`. Function: `withdrawPledge & withdrawRewards`.

Description:

Both `withdrawPledge` and `withdrawRewards` perform withdrawals from miner's account based on the input amount. Functions can be called by `FEE_DISTRIBUTOR` role. Withdrawals are only limited by quota, which provides some risks:

7. In case miner already has some balance(excluding pledge) it will be possible to take part of this balance from miner, because quota has to be bigger than allocation amount, due to rewards.
8. In case incorrect amount is passed into one of this functions, pledge or rewards can be distributed incorrectly. For example if full pledge + rewards amount is passed in `withdrawRewards` function, full amount will be distributed as rewards, which will result in stakers losing part of their staked FIL.

Remediation:

5.8. Incorrect check in deactivateStorageProvider function

Risk Level: Low

Status: Fixed in the commit 7568d591.

Contracts:

- StorageProviderRegistry.sol

Location: Lines: 243. Function: `deactivateStorageProvider(uint64 _ownerId)`.

Description:

`deactivateStorageProvider` function has the following check at line 243:

```
if (allocations[_ownerId].accruedRewards != allocations[_ownerId].repayment) revert InvalidRepayment();
```

Since quota equals repayment this check is supposed to allow deactivating SP only in case all quota was used. However, `repaidPledge` is also included in quota, because it is transferred from beneficiary address. This results in a fact that this check will never be fulfilled, because it is not possible for `accruedRewards` to reach an amount of repayment.

Remediation:

Consider changing check to `allocations[_ownerId].accruedRewards + allocations[_ownerId].repaidPledge != allocations[_ownerId].repayment`

5.9. deactivateStorageProvider doesn't have to transfer beneficiary back to owner

Risk Level: Low

Status: Fixed in the commit 03c79ecd.

Contracts:

- StorageProviderRegistry.sol

Location: Lines: 242. Function: `deactivateStorageProvider(uint64 _ownerId)`.

Description:

```
function deactivateStorageProvider(uint64 _ownerId) public onlyAdmin
activeStorageProvider(_ownerId) {
        if (allocations[_ownerId].accruedRewards !=
allocations[_ownerId].repayment) revert InvalidRepayment();

        // renounce beneficiary for SP

IRewardCollectorClient(resolver.getRewardCollector()).forwardChangeBeneficiary
(
            storageProviders[_ownerId].minerId,
            _ownerId,
            0,
            0
        );

        storageProviders[_ownerId].active = false;
        delete beneficiaryStatus[_ownerId];

        emit StorageProviderDeactivated(_ownerId);
}
```

This function is supposed to transfer beneficiary back to owner in case all quota was used. However, if all quota was user owner can claim the beneficiary back by himself, because it gets auto-approved.

Remediation:

Beneficiary transfer logic can be removed.

References:

- https://github.com/filecoin-project/builtin-actors/blob/master/actors/miner/src/lib.rs#L3392-L3395

5.10. Repayment can be less than the allocation limit

Risk Level: Low

Status: Fixed in commit 44f7ad76.

Contracts:

- contracts/StorageProviderRegistry.sol

Location: Lines: 298. Function: `updateAllocationLimit()`.

Description:

Storage providers can call the function `requestAllocationLimitUpdate()` in the registry contract to update allocation limits.

After the review, the admin can set new limits and repayment amounts with the function `updateAllocationLimit()`.

This function does not have any requirements for repayment, while `onboardStorageProvider()` has it.

Admin may mistakenly make repayment less than the allocation limit, which can result in the loss of funds.

** Remediation:**

Consider adding the next requirement to the `updateAllocationLimit()`

```
require(_repayment > _allocationLimit, "INCORRECT_REPAYMENT");
```

5.11. Incorrect pool requirements

Risk Level: Low

Status: Fixed in commit 20be1132.

Contracts:

- liquid-staking/contracts/LiquidStaking.sol

Location: Function: `constructor()`.

Description:

From the Pool contract natspec and functions `updateAdminFee()`, `updateBaseProfitShare()`:

- Admin fee must be not greater than 20%
- Profit sharing must be not greater than 80%

However, in the pool's `constructor()` there are different requirements:

```
require(_adminFee <= 10000, "INVALID_ADMIN_FEE");
baseProfitShare = _baseProfitShare; // any
```

This may lead to function `withdrawRewards()` DoS when variable `protocolShare`becomes larger than 100% of the withdrawn reward.

```
function withdrawRewards(uint64 ownerId, uint256 amount) external virtual
nonReentrant {
    ...
    vars.stakingProfit = (vars.withdrawn * profitShares[ownerId]) /
BASIS_POINTS;
    vars.protocolFees = (vars.withdrawn * adminFee) / BASIS_POINTS;
    vars.protocolShare = vars.stakingProfit + vars.protocolFees;

    if (vars.isRestaking) {
        // 1. underflow and revert in vars.withdrawn - vars.protocolShare
        vars.restakingAmt = ((vars.withdrawn - vars.protocolShare) *
vars.restakingRatio) / BASIS_POINTS;
    }
    // 2. underflow and revert
    vars.spShare = vars.withdrawn - (vars.protocolShare + vars.restakingAmt);
    ...
}
```

Remediation:

Consider adding requirements to the `constructor()`:

```
require(_adminFee <= 2000, "INVALID_ADMIN_FEE");
require(_baseProfitShare <= 8000, "INVALID_BASE_PROFIT");
```

5.12. Re-registration changes totalStorageProviders and totalInactiveStorageProviders

Risk Level: Low

Status: Fixed in the commit 8c7d1a12.

Contracts:

- contracts/StorageProviderRegistry.sol

Description:

In the contract StorageProviderRegistry.sol it is possible to re-register the storage provider until it is onboarded. The re-registration affects the values of `totalStorageProviders` and `totalInactiveStorageProviders` which are increased every time. As a result, both variables show wrong values after fake registrations.

Remediation:

Consider correcting counting logic.

5.13. Redundant ternary operator

Risk Level: Info

Status: Fixed in commit 0b1fd5d5.

Contracts:

- contracts/StorageProviderCollateral.sol

Location: Lines: 395. Function: `calcCollateralRequirements()`.

Description:

Function `calcCollateralRequirements()` in `StorageProviderCollateral` contract can be gas optimized. It contains a ternary operator, which can be removed without affecting the logic.

Remediation:

Consider changing from

```
uint256 usedAllocation = _allocationToUse > 0 ? _usedAllocation +
_allocationToUse : _usedAllocation;
```

to

```
uint256 usedAllocation = _usedAllocation + _allocationToUse;
```

5.14. The cached variable can be used

Risk Level: Info

Status: Fixed in commit 0b1fd5d5.

Contracts:

- contracts/StorageProviderRegistry.sol

Description:

Structure `storageProviders[ownerId]` is cached in variable `storageProvider` in function `changeBeneficiaryAddress()`and `acceptBeneficiaryAddress()`.

However, the cached variable is not used in the call to the `Pool` contract.

```
function changeBeneficiaryAddress() public virtual override nonReentrant {

        address ownerAddr = msg.sender.normalize();
```

```
        (bool isID, uint64 ownerId) = ownerAddr.getActorID();
        require(isID, "INACTIVE_ACTOR_ID");

        StorageProviderTypes.StorageProvider memory storageProvider =
storageProviders[ownerId];
        require(storageProvider.onboarded, "NON_ONBOARDED_SP");

        // can be ILiquidStakingClient(storageProvider.targetPool)

ILiquidStakingClient(storageProviders[ownerId].targetPool).forwardChangeBenefi
ciary(
            storageProvider.minerId,
            storageProvider.targetPool,
            allocations[ownerId].repayment,
            storageProvider.lastEpoch
        );

        emit
StorageProviderBeneficiaryAddressUpdated(storageProvider.targetPool);
    }
```

Remediation:

Consider using the cached value in the call for gas optimization:

```
ILiquidStakingClient(storageProvider.targetPool).forwardChangeBeneficiary(
            storageProvider.minerId,
            storageProvider.targetPool,
            allocations[ownerId].repayment,
            storageProvider.lastEpoch
        );
```

5.15. Unused imports

Risk Level: Info

Status: Fixed in the commit [b01681ab](b01681ab).

Contracts:

- contracts/StorageProviderCollateral.sol

Location: Lines: 10.

Description:

The PrecompilesAPI import is never used.

```
contracts/StorageProviderCollateral.sol:
  10: import {PrecompilesAPI} from "filecoin-
solidity/contracts/v0.8/PrecompilesAPI.sol";
```

Remediation:

Consider removing the unused import.

5.16. Extra variable initialization

Risk Level: Info

Status: Fixed in the commit 34172367.

Description:

Functions for state updates create new `prevVariable` in memory only for comparison with the new value.

Example:

```
function setCollateralAddress(address newAddr) public {
        require(hasRole(LIQUID_STAKING_ADMIN, msg.sender), "INVALID_ACCESS");
        require(newAddr != address(0), "INVALID_ADDRESS");

        address prevCollateral = address(collateral);
        require(prevCollateral != newAddr, "SAME_ADDRESS");

        collateral = IStorageProviderCollateralClient(newAddr);

        emit SetCollateralAddress(newAddr);
    }
```

Consider making a comparison without creating a new variable for gas optimization in the next functions:

`contracts/LiquidStaking.sol`:

- `updateProfitShare()`
- `setCollateralAddress()`
- `setRegistryAddress()`
- `updateAdminFee()`
- `updateBaseProfitShare()`
- `updateRewardsCollector()`

contracts/StorageProviderRegistry.sol:

- setCollateralAddress()
- updateMaxAllocation()
- setMinerAddress()

contracts/StorageProviderCollateral.sol:

- updateBaseCollateralRequirements()
- setRegistryAddress()

Remediation:

Consider making a comparison without creating a new variable for gas optimization.

5.17. The variable can be constant

Risk Level: Info

Status: Fixed in the commit a8a9fada.

Contracts:

- contracts/RewardCollector.sol

Location: Lines: 61.

Description:

Consider marking the variable BASIS_POINTS in storage as constant to save gas in contract RewardCollector. The same constant variable is constant in the LiquidStaking and StorageProviderCollateralcontracts.

Remediation:

Consider marking BASIS_POINTS variable as constant:

```
uint256 private constant BASIS_POINTS = 10000;
```

5.18. String error messages are used instead of custom errors

Risk Level: Info

Status: Fixed in commit 2484fed7.

Contracts:

- contracts/StorageProviderCollateral.sol
- contracts/StorageProviderRegistry.sol
- contracts/LiquidStaking.sol

Description:

The contracts make use of the `require()` to emit an error. While this is a perfectly valid way to handle errors in Solidity, it is not always the most efficient.

Remediation:

Consider using custom errors as they are more gas efficient while allowing developers to describe the error in detail using NatSpec.

References:

- https://blog.soliditylang.org/2021/04/21/custom-errors/

5.19. Unnecessary transfer in _wrapWETH9

Risk Level: Info

Status: Fixed in the commit 2484fed7.

Contracts:

- contracts/LiquidStaking.sol

Location: Lines: 485. Function: `_wrapWETH9`.

Description:

Current implementation of `_wrapWETH9()` is used to wrap `FIL` into `WFIL` that have been sent to the `LiquidStaking` contract. It does not need to send them to itself because they are already on balance of the `LiquidStaking` contract.

Remediation:

Remove unnecessary transfer.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.