SALUS SECURITY

JAN 2025

CODE SECURITY ASSESSMENT

FILUS

Overview

Project Summary

- Name: Filus
- Platform: FileCoin chain
- Language: Solidity
- Repository:
    - https://github.com/FilUs-Labs/Filus
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Filus
Version	v4
Type	Solidity
Dates	Jan 06 2025
Logs	Dec 23 2024, Dec 31 2024, Jan 03 2025, Jan 06 2025

Vulnerability Summary

Total High-Severity issues	8
Total Medium-Severity issues	6
Total Low-Severity issues	3
Total informational issues	1
Total	18

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 **4**
1.1 About SALUS 4
1.2 Audit Breakdown 4
1.3 Disclaimer 4
Findings **5**
2.1 Summary of Findings 5
2.2 Notable Findings 6
1. Users can steal funds through flashLoans 6
2. Incorrect collateral balance calculation 7
3. Incorrect health factor calculation 8
4. Incorrect debt owner 9
5. Lack of length check for the minerIds array in the validateRawPower function 10
6. SP owner can avoid the full liquidation 11
7. Improper slippage control in liquidation 12
8. Byte conversion error when calculating the balance 14
9. Incorrect quota parameter in removeMiner 15
10. Possible dos in liquidation 16
11. Incorrect remaining Ether check 17
12. Incorrect beneficiary change 18
13. The `updatePriceFeeds` function lacks a refund feature. 19
14. Centralization risk 20
15.Improper validation check 21
16.Missing disableInitializer in implement contract 22
17. Collateral can be freely set 23
2.3 Informational Findings 24
18. Gas optimization 24
Appendix **25**
Appendix 1 - Files in Scope 25

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	Users can steal funds through flashLoans	High	Business Logic	Resolved
2	Incorrect collateral balance calculation	High	Business Logic	Resolved
3	Incorrect health factor calculation	High	Business Logic	Resolved
4	Incorrect debt owner	High	Business Logic	Resolved
5	Lack of length check for the minerIds array in the validateRawPower function	High	Business Logic	Resolved
6	SP owner can avoid the full liquidation	High	Business Logic	Resolved
7	Improper slippage control in liquidation	High	Business Logic	Resolved
8	Bytes conversion error when calculating the balance	High	Business Logic	Resolved
9	Incorrect quote parameter in removeMiner	Medium	Business Logic	Resolved
10	Possible dos in liquidation	Medium	Business Logic	Resolved
11	Incorrect remaining Ether check	Medium	Business Logic	Resolved
12	Incorrect beneficiary change	Medium	Business Logic	Resolved
13	The `updatePriceFeeds` function lacks a refund feature.	Medium	Business Logic	Resolved
14	Centralization risk	Medium	Centralization	Acknowledged
15	Improper validation check	Low	Business Logic	Resolved
16	Missing disableInitializer in implement contract	Low	Business Logic	Resolved
17	Collateral can be freely set	Low	Business Logic	Resolved
18	Gas optimization	Informational	Gas Optimization	Acknowledged

2.2 Notable Findings

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

1. Users can steal funds through flashLoans

Severity: High	Category: Business Logic
Target: - Filus/contracts/protocol/lendingpool.sol	

Description

The LendingPool health checks for borrowing have been removed, restricting borrowing exclusively to SPs via the spHub. However, in the `flashloan` function, if a user opts not to repay immediately, the contract creates a borrowing position for them. Without health checks, users can borrow funds without collateral, enabling them to drain the liquidity pool through the flashloan function.

Filus/contracts/protocol/lendingpool.sol:L554-L565

```
function flashLoan(
  address receiverAddress, address[] calldata assets, uint256[] calldata amounts,
  uint256[] calldata modes, address onBehalfOf,
  bytes calldata params, uint16 referralCode
) external override whenNotPaused {
   ...
  for (vars.i = 0; vars.i < assets.length; vars.i++) {
    ...
    if (DataTypes.InterestRateMode(modes[vars.i]) == DataTypes.InterestRateMode.NONE) {
      ...
    } else {
      _executeBorrow(
        ExecuteBorrowParams(
          vars.currentAsset, msg.sender, onBehalfOf,
          vars.currentAmount, modes[vars.i], vars.currentFuTokenAddress,
          referralCode, false
        )
      );
    }
    ...
  }
}
```

Recommendation

Remove the logic that allows repayment of the `` `flashLoan` `` by opening a borrowing position.

Status

This issue has been resolved by the team with commit 8b9510a.

2. Incorrect collateral balance calculation

Severity: High	Category: Business Logic

Target:
- Filus/contracts/protocol/StorageProvider/SPHub.sol

Description

The calculateCollateral function in the spHub contract calculates the collateral value. However, when retrieving the availableBalance of the associated miner, the protocol overlooks the balance's sign. If the miner's collateral is low and a miscalculation occurs, the availableBalance could become negative. Since the calculateCollateral function does not account for the balance's sign, this results in an inaccurate collateral value calculation.

Filus/contracts/protocol/StorageProvider/SPHub.sol:L644-L660

```
function calculateCollateral() public view returns (uint256) {
    uint256 totalCollateral;
    uint256 length = minerIds.length;
    for (uint256 i; i < length; ) {
        uint64 minerId = minerIds[i];
        CommonTypes.BigInt memory availableBalance = FilecoinAPI
            .getAvailableBalance(minerId);
        uint256 availableAmount = uint256(bytes32(availableBalance.val));
        totalCollateral += availableAmount;
        totalCollateral += liquidationValue[minerId];
        unchecked {
            ++i;
        }
    }
    totalCollateral += _getTokenBalance(address(0));
    return totalCollateral;
}
```

Recommendation

When calculating the collateral value, the sign of the miner's account balance should be taken into account.

Status

This issue has been resolved by the team with commit 8b9510a.

3. Incorrect health factor calculation

Severity: High	Category: Business Logic

Target:
- Filus/contracts/protocol/StorageProvider/utils/SPValidationLogic.sol

Description

The calculateUserAccountData function calculates the health factor of an SPHub. However, if the SPHub's collateral is zero, its health factor is set to the maximum value. This allows the SPHub owner to remove their last miner without repaying the debt, as the health factor remains at its maximum.

Filus/contracts/protocol/StorageProvider/SPValidationLogic.sol:L139-L221

```
function calculateUserAccountData(
    address user,
    DataTypes.UserConfigurationMap memory userConfig,
    address[] memory reservesList,
    address oracle,
    address WETH,
    uint256 collateralETH,
    address lendingPoolAddress
) internal view returns (uint256, uint256, uint256, uint256, uint256) {
    ...
    if (collateralETH == 0) {
        return (0, 0, 0, 0, type(uint256).max);
    }
    ...
}
```

Recommendation

According to AAVE's logic, we should return maximum if this SPHub's debt is zero.

Status

This issue has been resolved by the team with commit 8b9510a.

4. Incorrect debt owner

Severity: High	Category: Business Logic

Target:
- Filus/contracts/protocol/StorageProvider/SPHub.sol

Description

When the SPHub owner borrows assets, the protocol checks the SPHub's health factor. However, the issue lies in using msg.sender as the user parameter. The health factor calculation is based on msg.sender's debt token balance, but this balance actually belongs to the SPHub, not the owner.

Filus/contracts/protocol/StorageProvider/SPHub.sol:L253-L279

```
function borrow(
    address asset,
    uint256 amount,
    uint16 referralCode,
    address onBehalfOf
) public onlyHubOwner whenNotLiquidating {
    ...
    _executeBorrow(
        ExecuteBorrowParams(
            asset,
            msg.sender,
            onBehalfOf,
            amount,
            INTEREST_RATE_MODE,
            reserveData.fuTokenAddress,
            referralCode,
            true
        )
    );
```

Recommendation

Change `msg.sender` to address(this).

Status

This issue has been resolved by the team with commit 8b9510a.

5. Lack of length check for the minerIds array in the validateRawPower function

Severity: High	Category: Business Logic
Target: - Filus/contracts/protocol/StorageProvider/utils/SPValidationLogic.sol	

Description

In the current liquidation logic, the liquidator first takes control of a miner associated with the spHub, withdraws its balance, and uses the withdrawn FIL to repay the debt via liquidationCall. However, the collateral calculation includes the FIL balance of the spHub itself, yet the contract lacks a direct method to utilize this balance for loan repayment. This can result in an incomplete liquidation if the miner's withdrawn balance is insufficient to cover the debt.

Filus/contracts/protocol/StorageProvider/utils/SPValidationLogic.sol:L72-L99

```
function validateRawPower(
    address factory,
    uint64[] memory minerIds,
    mapping(uint64 => uint256) storage initialRawBytePower
) internal view {
    uint256 initialTotalRawPower;
    uint256 currentTotalRawPower;
    uint256 length = minerIds.length;
    for (uint256 i; i < length; ) {
        uint64 minerId = minerIds[i];
        initialTotalRawPower += initialRawBytePower[minerId];
        PowerTypes.MinerRawPowerReturn memory rawPower = FilecoinAPI
            .getRawPower(minerId);
        currentTotalRawPower += rawPower.raw_byte_power.bigInt2Uint();
        unchecked {
            ++i;
        }
    }
    ...

}
```

Recommendation

Check if the `minerIds` array is empty in the `validateRawPower` function. If it is, revert the transaction.

Status

This issue has been resolved by the team with commit 8b9510a.

6. SP owner can avoid the full liquidation

Severity: High	Category: Business Logic

Target:
- Filus/contracts/protocol/StorageProvider/SPHub.sol

Description

The `spHub` contract provides borrowing services for SPs, where the balance of the associated miner account and the balance in the `spHub` contract are used as collateral.

When calling the `borrow` function, the `_validateRawPower` function is invoked to verify changes in the miner account's weight compared to the initial value. However, the `SPValidationLogic.validateRawPower` function does not check whether the `minerIds` array is empty. If the array is empty, the check passes automatically. This means that non-SP users can also borrow from `spHub` without adding any miners. They can directly send FIL to the `spHub` contract and successfully borrow assets from the lending pool.
Furthermore, because the liquidation process requires the liquidator to first take control of the miners associated with `spHub`, the lack of miners would prevent liquidation from taking place, thereby increasing the system's risk.
Attach Scenario:
1. The sp adds a miner with a balance of 10 ETH.
2. The sp directly transfers 90 ETH to the `spHub`, making the total collateral value 100 ETH (10 ETH from the miner and 90 ETH transferred).
3. The sp borrows 11 ETH in WFIL debt and 60 ETH in other debts.
4. The sp needs to be liquidated, but the liquidator can only access up to 10 ETH from the miner to settle the debt. The liquidator can only repay part of the WFIL debt and will not be able to continue liquidating the remaining debt.
Filus/contracts/protocol/StorageProvider/SPHub.sol:L442-L448

```
function liquidationCall() public payable onlyLiquidator {
    if (!liquidationProcessInfo.isLiquidationProcess)
        revert NotInLiquidationProcess();
    ...
    // Clean up liquidation process
    liquidationProcessInfo.isLiquidationProcess = false;
    liquidationProcessInfo.minerId = 0;
    liquidationValue[liquidationProcessInfo.minerId] = 0;
    initialPledgedAmount[liquidationProcessInfo.minerId] = 0;
    initialRawBytePower[liquidationProcessInfo.minerId] = 0;
}
```

Recommendation

We suggest supporting the direct use of the contract's internal balance for liquidation.

Status

This issue has been resolved by the team with commit 8b9510a.

7. Improper slippage control in liquidation

Severity: High	Category: Business Logic

Target:
- Filus/contracts/protocol/StorageProvider/SPHub.sol

Description

When an SPHub becomes unhealthy, the liquidator is responsible for liquidating the position. During liquidation, all borrowed debts are iterated through, and the expected borrowed assets are swapped using input FIL.

However, the SPHub's balance is used as the maximum amountIn for these swaps, creating a potential vulnerability to MEV attacks.

Attack Vector:

1. Liquidator triggers liquidationCall with input FIL valued 1100 USD.
2. The first borrowed asset is USDC, debt amount is 10 USD. The second borrowed asset is USDT, debt amount is 1000 USD.
3. In normal cases, we can use the input FIL to pay for both borrowed assets. But if malicious users manipulate the price in Sushiswap, we may swap back 10 USD using 1100 USD FIL. The SPHub owner will take one huge loss.

Filus/contracts/protocol/StorageProvider/SPHub.sol:L442-L496

```
function liquidationCall() public payable onlyLiquidator {
    ...
    // If there's remaining ETH, swap and repay other assets
    if (msg.value > wethRepayAmount) {
        // Iterate through borrowed assets to repay
        for (uint256 i = 0; i < borrowedAssets.length; i++) {
            address asset = borrowedAssets[i];
            if (asset != weth) {
                uint256 assetDebt = getDebtBalances(asset);
                if (assetDebt > 0) {
                    // Swap remaining ETH for the borrowed asset and repay
                    _swapETHForToken(asset, assetDebt);
                    uint256 swappedBalance = IERC20(asset).balanceOf(address(this));
                    if (swappedBalance > 0) {
                        repay(asset, swappedBalance);
                    }
                }
            }
        }
    }
   ...
}
```

Filus/contracts/protocol/StorageProvider/SPHub.sol:L253-L279

```
    function _swapETHForToken(address token, uint256 amountOut) internal {
        address swapAdapter = factory.getSwapAdapter();
        ISwapAdapter swapAdapterInstance = ISwapAdapter(payable(swapAdapter));
        uint256 amountIn = swapAdapterInstance.swap{
```

```
            value: _getTokenBalance(address(0))
        }(
            factory.getWETH(),
            token,
            _getTokenBalance(address(0)),
            amountOut
        );
        emit ETHSwappedForToken(token, amountIn, amountOut);
    }
```

Recommendation

Provide one specific maximum amountIn for each borrowed asset.

Status

This issue has been resolved by the team with commit 8b9510a.

8. Bytes conversion error when calculating the balance

Severity: High	Category: Business Logic

Target:
- Filus/contracts/protocol/StorageProvider/SPHub.sol

Description

In Filus, we use miners' balances as collateral. When calculating a miner's balance, we convert the availableBalance.val from `bytes` to `uint256`. However, during this conversion, casting `bytes` to `bytes32` adds unintended zero padding, leading to inaccurate results. This issue affects the final calculations and needs to be addressed.

Filus/contracts/protocol/StorageProvider/SPHub.sol:L657-L670

```
function calculateCollateral() public view returns (uint256) {
    int256 totalCollateral;
    uint256 length = minerIds.length;
    for (uint256 i; i < length; ) {
        uint64 minerId = minerIds[i];
        CommonTypes.BigInt memory availableBalance = FilecoinAPI
            .getAvailableBalance(minerId);

        totalCollateral += int256(liquidationValue[minerId]);
        if(availableBalance.neg) {
            totalCollateral -= int256(uint256(bytes32(availableBalance.val)));
        } else {
            totalCollateral += int256(uint256(bytes32(availableBalance.val)));
        }
...
}
```

Recommendation

Convert the bytes val to the uint256 safely.

```
function bytesToUint256(bytes memory input) public pure returns (uint256) {
    require(input.length <= 32, "Input too long");
    bytes32 result;

    assembly {
        result := mload(add(input, 32))
    }
    uint shiftAmount = (32 - input.length) * 8;
    result = bytes32(uint(result) >> shiftAmount);

    return uint256(result);
}
```

Status

This issue has been resolved by the team with commit 270561a.

9. Incorrect quota parameter in removeMiner

Severity: Medium	Category: Business Logic
Target: - Filus/contracts/protocol/StorageProvider/SPHub.sol	

Description

In Filus, when removing a miner from an spHub, we update the beneficiary to the miner's previous owner. However, we encountered an issue where setting the quota to zero is not permitted. According to the actor implementation, a zero quota is invalid when the miner owner attempts to change the beneficiary to another address.

Filus/contracts/protocol/StorageProvider/SPHub.sol:L162-L183

```
function removeMiner(uint64 minerId) public onlyHubOwner onlyActivated
whenNotLiquidating {
    ...
    // 2. Change Owner
    FilecoinAPI.changeOwnerAddress(minerId, minerOwnerInfo[minerId]);

    // 3. Change Beneficiary to the new owner
    CommonTypes.FilAddress memory minerOwner = FilecoinAPI
        .getOwner(minerId)
        .owner;
    FilecoinAPI.changeBeneficiary(
        minerId,
        minerOwnerInfo[minerId],
        CommonTypes.BigInt(hex"00", false),
        CommonTypes.ChainEpoch.wrap(0)
    );
```

Recommendation

Using one positive quote value.

Status

This issue has been resolved by the team with commit 270561a.

10. Possible dos in liquidation

Severity: Medium	Category: Business Logic

Target:
- Filus/contracts/protocol/StorageProvider/SPHub.sol

Description

During liquidation, the liquidator uses FIL to swap for the expected borrowed assets and repay the debt.

However, if the remaining FIL is insufficient to complete the swap for the expected borrowed assets, the transaction will revert due to the use of exactOut mode.

Filus/contracts/protocol/StorageProvider/SPHub.sol:L442-L471

```
function liquidationCall() public payable onlyLiquidator {
    ...
    if (msg.value > wethRepayAmount) {
        for (uint256 i = 0; i < borrowedAssets.length; i++) {
            address asset = borrowedAssets[i];
            if (asset != weth) {
                uint256 assetDebt = getDebtBalances(asset);
                if (assetDebt > 0) {
                    _swapETHForToken(asset, assetDebt);
                    uint256 swappedBalance = IERC20(asset).balanceOf(address(this));
                    if (swappedBalance > 0) {
                        repay(asset, swappedBalance);
                    }
                }
            }
        }
    }
    ...
}
```

Filus/contracts/protocol/StorageProvider/SPHub.sol:L442-L471

```
function swap(
    address tokenIn, address tokenOut, uint256 maxAmountIn, uint256 amountOut
) external payable returns (uint256 amountIn) {
    ...
}
```

Recommendation

Before we swap tokens, we should check whether the remaining FIL can swap enough expected borrowed assets.

Status

This issue has been resolved by the team with commit 8b9510a.

11. Incorrect remaining Ether check

Severity: Medium	Category: Business Logic
Target: - Filus/contracts/protocol/StorageProvider/SPHub.sol	

Description

During liquidation, the liquidator deducts a premiumAmount and sends it to the treasury. Any remaining FIL after repaying the WFIL debt is used to repay other borrowed assets' debt.

However, the issue arises when the contract attempts to repay other borrowed assets' debt even if no FIL remains. This happens because msg.value is always larger than the liquidatedAmount due to the inclusion of the premiumAmount.

Filus/contracts/protocol/StorageProvider/SPHub.sol:L442-L471

```
function liquidationCall() public payable onlyLiquidator {
    ...
    uint256 premiumAmount = (msg.value * factory.getLiquidationPremium()) / 10000;
    uint256 liquidatedAmount = msg.value - premiumAmount;
    address treasury = factory.getTreasury();
    _safeTransferETH(treasury, premiumAmount);

    // Repay WETH debt first
    uint256 wethRepayAmount = wethDebt < liquidatedAmount ? wethDebt : liquidatedAmount;
    if (wethRepayAmount > 0) {
        _handleETHRepayment(wethRepayAmount, true);
    }

       // If there's remaining ETH, swap and repay other assets
    if (msg.value > wethRepayAmount) {
     ...
    }
}
```

Recommendation

```
diff --git a/contracts/protocol/StorageProvider/SPHub.sol
b/contracts/protocol/StorageProvider/SPHub.sol
index 276fa44..b0e4fd3 100644
--- a/contracts/protocol/StorageProvider/SPHub.sol
+++ b/contracts/protocol/StorageProvider/SPHub.sol
@@ -468,7 +468,7 @@ contract SPHub is ISPHub, Initializable {
         }

         // If there's remaining ETH, swap and repay other assets
-        if (msg.value > wethRepayAmount) {
+        if (liquidatedAmount > wethRepayAmount) {
```

Status

This issue has been resolved by the team with commit 8b9510a.

12. Incorrect beneficiary change

Severity: Medium	Category: Business Logic

Target:
- Filus/contracts/protocol/StorageProvider/SPHub.sol

Description

When the SPHub owner removes a miner, the contract updates the miner's owner to the previous owner and sets the beneficiary to the same address.
However, when the changeOwnerAddress function is called, it only creates a change owner proposal. The actual owner change occurs only after the proposal is confirmed. As a result, the minerOwner used in the changeBeneficiary function may refer to an unexpected owner, leading to inconsistencies.

Filus/contracts/protocol/StorageProvider/SPHub.sol:L442-L471

```
function removeMiner(uint64 minerId) public onlyHubOwner onlyActivated
whenNotLiquidating {
    ...
    FilecoinAPI.changeOwnerAddress(minerId, minerOwnerInfo[minerId]);

    CommonTypes.FilAddress memory minerOwner = FilecoinAPI
        .getOwner(minerId)
         .owner;
    FilecoinAPI.changeBeneficiary(
        minerId,
        minerOwner,
        CommonTypes.BigInt(hex"00", false),
        CommonTypes.ChainEpoch.wrap(0)
    );
}
```

Recommendation

```
index 276fa44..dea5bbe 100644
--- a/contracts/protocol/StorageProvider/SPHub.sol
+++ b/contracts/protocol/StorageProvider/SPHub.sol
@@ -170,7 +170,7 @@ contract SPHub is ISPHub, Initializable {
             .owner;
         FilecoinAPI.changeBeneficiary(
             minerId,
-            minerOwner,
+            minerOwnerInfo[minerId],
             CommonTypes.BigInt(hex"00", false),
             CommonTypes.ChainEpoch.wrap(0)
         );
```

Status

This issue has been resolved by the team with commit 8b9510a.

13. The `updatePriceFeeds` function lacks a refund feature.

Severity: High	Category: Business Logic
Target: - Filus/contracts/misc/FilUsOracle.sol	

Description

The updatePriceFeeds function allows users to update oracle price data and calculates the required fees in real-time during the transaction. These fees may differ from the amount the user inputs. Users are responsible for ensuring the success of the price update and may provide more fees than necessary. The protocol should refund any excess amount to the user to prevent excess funds from getting stuck in the contract.

Filus/contracts/misc/FilUsOracle.sol:L188-L196

```
function updatePriceFeeds(bytes[] memory pythUpdateData) public payable {
    // update pyth price feeds
    uint updateFee = IPyth(pyth).getUpdateFee(pythUpdateData);
    try
        IPyth(pyth).updatePriceFeeds{value: updateFee}(pythUpdateData)
    {} catch {
        // if update failed, assuming that the user (Front is also a user)
    }
}
```

Recommendation

It is recommended to refund any unused fees in the `updatePriceFeeds` function.

Status

This issue has been resolved by the team with commit 8b9510a.

14. Centralization risk

Severity: Medium	Category: Centralization
Target: - Filus/contracts/protocol/StorageProvider/SPHub.sol	

Description

The SPHub contract has privileged accounts: owner, Operator, and Liquidator. They can modify many critical parameters, upgrade contracts, control borrowers' miners, etc.

If the private keys of these accounts are compromised, it could have a devastating impact on the system.

Recommendation

We recommend transferring privileged accounts to multi-sig 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.

Status

This issue has been acknowledged by the team.

15.Improper validation check

Severity: Low	Category: Business Logic
Target: - Filus/contracts/protocol/StorageProvider/SPHub.sol	

Description

The removeMiner function contains an improper validation check. It only reverts if both minerExists[minerId] returns false and FilecoinAPI.getOwner(minerId).proposed.data.length is zero. This check could potentially bypass the minerExists[minerId] validation.

Filus/contracts/protocol/StorageProvider/SPHub.sol:L157-L162

```
function removeMiner(uint64 minerId) public onlyHubOwner onlyActivated
whenNotLiquidating {
    // 1. Validation
    if (
        !minerExists[minerId] &&
        FilecoinAPI.getOwner(minerId).proposed.data.length != 0
    ) {
        revert MinerNotManagedOrNoPendingOwnerChange();
    }
    ...
}
```

Recommendation

Use `||` instead of `&&`.

Status

This issue has been resolved by the team with commit 8b9510a.

16.Missing disableInitializer in implement contract

Severity: Low	Category: Business Logic
Target: - Filus/contracts/protocol/StorageProvider/SPHub.sol	

Description

The `spHub` contract is an upgradeable logic contract, but it is missing the `disableInitializer`, which could allow malicious actors to initialize the logic contract.

Filus/contracts/protocol/StorageProvider/SPHub.sol:L84-L87

```
function initialize(address _factory, address _owner) public initializer {
    factory = ISPHubFactory(_factory);
    hubOwner = _owner;
}
```

Recommendation

In the constructor, call `disableInitializer` to disable the initialization of the logic contract.

Status

This issue has been resolved by the team with commit 8b9510a.

17. Collateral can be freely set

Severity: Low	Category: Business Logic
Target: - Filus/contracts/protocol/lendingpool/LendingPool.sol	

Description

The logic for health checks and related functions has been removed from the LendingPool, meaning the deposit function no longer automatically sets assets as collateral. However, the LendingPool still includes the setUserUseReserveAsCollateral function, allowing users to manually designate their assets as collateral.

Filus/contracts/protocol/lendingpool/LendingPool.sol:L499-L424

```
function setUserUseReserveAsCollateral(address asset, bool useAsCollateral)
  external
  override
  whenNotPaused
{
  DataTypes.ReserveData storage reserve = _reserves[asset];

  ValidationLogic.validateSetUseReserveAsCollateral(
    reserve,
    asset,
    useAsCollateral,
    _reserves,
    _usersConfig[msg.sender],
    _reservesList,
    _reservesCount,
    _addressesProvider.getPriceOracle()
  );

  _usersConfig[msg.sender].setUsingAsCollateral(reserve.id, useAsCollateral);

  if (useAsCollateral) {
    emit ReserveUsedAsCollateralEnabled(asset, msg.sender);
  } else {
    emit ReserveUsedAsCollateralDisabled(asset, msg.sender);
  }
}
```

Recommendation

Delete the `setUserUseReserveAsCollateral` function in the `LendingPool`.

Status

This issue has been resolved by the team with commit 8b9510a.

2.3 Informational Findings

18. Gas optimization

Severity: Informational	Category: Gas Optimization
Target: - Filus/contracts/protocol/StorageProvider/SPHubFactory.sol	

Description

Each time a new spHub is created, the spHubFactory contract deploys a new proxyAdmin contract. However, a single proxyAdmin contract can manage upgrades for multiple proxy contracts, making it unnecessary to create a new proxyAdmin for each spHub proxy.

Filus/contracts/protocol/StorageProvider/SPHubFactory.sol:L69-L70

```
function createSPHub() public payable returns (address) {
    ...
    // Create new ProxyAdmin and map it to the user
    ProxyAdmin proxyAdmin = new ProxyAdmin();
    userToProxyAdmin[msg.sender] = address(proxyAdmin);
    ...
}
```

Recommendation

Consider using a single `proxyAdmin` to control all proxies.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 7924021:

File	SHA-1 hash
FilUsOracle.sol	3031bbdffd34680620b7873e6919bb32f5a0f434
lendingpool.sol	597bffca27f25ea22b41a92100506cc226f543af
LendingPoolConfigurator.sol	fc3b7e51abf305efa67e958204c705c2bd69c76d
SPHubFactory.sol	4e628524188824cd8a549a3634da610475ee7b40
SPHub.sol	f44dcc492ba60bd6e1beeb119cb738cbd49adb8e
SPDataProvider.sol	880d3108a5c0a9699da3821292b1a0dc081ad07f
UiPoolDataProviderV2.sol	0b36f1730f74b59af4efb964cc43d3135152d926
Conversion.sol	0bcadf5cbc59ce7f8f7c62141cc641dc51439717
FileCoinAPI.sol	7f1ce42452caf710cf0ce463299cc883747ff32f
SPValidationLogic.sol	f448dacb7e273150b2d17ecc7e84d10d4ed4c46b