SALUS SECURITY

MAR 2025

CODE SECURITY ASSESSMENT

FILUS

Overview

Project Summary

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

Project Dashboard

Application Summary

Name	Filus - Incremental Audit
Version	v2
Type	Solidity
Dates	Mar 31 2025
Logs	Mar 24 2025, Mar 31 2025

Vulnerability Summary

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

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

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	Non-weth debt cannot be liquidated	High	Business Logic	Resolved
2	Missing unpause function	Medium	Business Logic	Resolved
3	Liquidation dos because of incorrect amountOut	Medium	Business Logic	Resolved
4	Centralization risk	Medium	Centralization	Acknowledged
5	HubOwner's pause request can be bypassed	Low	Business Logic	Resolved
6	Incorrect amountOut calculation	Low	Business Logic	Resolved
7	Improper premium calculation	Low	Business Logic	Acknowledged
8	The key functions cannot be paused	Informational	Business Logic	Resolved

2.2 Notable Findings

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

1. Non-weth debt cannot be liquidated

Severity: High	Category: Business Logic

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

Description

In Filus, when SPHub's borrow position is not healthy, the liquidator role can liquidate via repaying part of the debt.
The problem here is that if we need to repay non-weth debt, the liquidator will trigger the `repay` function, and the `repay` function has one `onlyHubOwner` modifier. This will cause the liquidation to be reverted.
Filus/contracts/protocol/StorageProvider/SPHub.sol:L518-L573

```
function liquidationCallWithoutPrepare(address asset, uint256 liquidationAmount, uint256
maxAmountIn) public payable onlyLiquidator {
    ...
    if(asset == weth) {
      ...
    } else {
        if (assetDebt > 0) {
            assetDebt = assetDebt > liquidatedAmount ? liquidatedAmount : assetDebt;
            _swapETHForToken(asset, assetDebt, maxAmountIn);

            IERC20(asset).safeTransfer(treasury, premiumAmount);
            // It means that it's impossible to work this.
            repay(asset, assetDebt, true);
        }
    }
}
```

Filus/contracts/protocol/StorageProvider/SPHub.sol:L319-L374

```
function repay(
    address asset,
    uint256 amount,
    bool useContractBalance
   ) public onlyHubOwner returns (uint256 repaidAmount) {
}
```

Recommendation

Add one internal function for `repay`.

Status

This issue has been resolved by the team with commit 3ec88d7.

2. Missing unpause function

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

Description

The protocol allows the Sphub owner to request the Sphub contract to enter a paused state. However, it lacks a function to unpause the contract. Since the `addMiner`, `removeMiner`, `pullFunds`, and `borrowETH` functions are expected to be paused when the contract is paused, once the contract is paused, users' Miners will be permanently locked in the contract.

Filus/contracts/protocol/StorageProvider/SPHub.sol:L839-L850

```
function requestPauseOrUnpause() public onlyHubOwner {
    requestedPause = true;
    emit PauseRequested();
}

function confirmPause(bool accept) public onlyOperator {
    if (accept) {
        _pause();
    }
    requestedPause = false;
    emit PauseConfirmed(accept);
}
```

Recommendation

It is recommended to add a function to unpause the contract.

Status

This issue has been resolved by the team with commit 3ec88d7.

3. Liquidation dos because of incorrect amoutOut

Severity: Medium	Category: Business Logic

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

Description

When liquidating a non-WETH asset using `liquidationCallWithoutPrepare`, the function swaps ETH for the asset to be liquidated through `_swapETHForToken`. However, it only swaps an amount equal to `assetDebt` and does not account for `premiumAmount`, which may cause liquidation to fail due to insufficient asset balance in the contract.

Filus/contracts/protocol/StorageProvider/SPHub.sol:L563-L572

```
function liquidationCallWithoutPrepare(address asset, uint256 liquidationAmount, uint256
maxAmountIn) public payable onlyLiquidator {
    ...
        if (assetDebt > 0) {
            assetDebt = assetDebt > liquidatedAmount ? liquidatedAmount : assetDebt;
            // Swap remaining ETH for the borrowed asset and repay
            _swapETHForToken(asset, assetDebt, maxAmountIn);
            // Take care of premium
            IERC20(asset).safeTransfer(treasury, premiumAmount);
            repay(asset, assetDebt, true);
        }
    }
}
```

Recommendation

It is recommended to swap an amount of tokens equal to `premiumAmount + assetDebt` through `_swapETHForToken`.

Status

This issue has been resolved by the team with commit 3ec88d7.

4. 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.

5. HubOwner's pause request can be bypassed

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

Description

In SPHub, we added one new pause feature. The Hub owner can request a pause for this SPHub. Then the operator can confirm this pause. After the operator's confirmation, this SPHub's main functions will be paused.

The problem here is that we miss the request check in `confirmPause`. So the operator can pause this SPHub directly even if the Hub owner does not request the pause.

Filus/contracts/protocol/StorageProvider/SPHub.sol:L874-L885

```
function requestPauseOrUnpause() public onlyHubOwner {
    requestedPause = true;
    emit PauseRequested();
}
function confirmPause(bool accept) public onlyOperator {
    if (accept) {
        _pause();
    }
    requestedPause = false;
    emit PauseConfirmed(accept);
}
```

Recommendation

Add one request check in `confirmPause`.

Status

This issue has been resolved by the team with commit 3ec88d7.

6. Incorrect amountOut calculation

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

Description

In Filus, when we repay the interest if there is not enough `assetBalance`, we will withdraw available balances from miners, and then swap the ETH for the borrowed tokens.

The problem here is that we don't consider the possible `assetBalance` in this contract. The actual target token amount we need is `interestAmount - assetBalance`.

Filus/contracts/protocol/StorageProvider/SPHub.sol:L628-L660

```
function repayInterest(address asset, uint256 interestAmount, uint256 maxAmountIn)
internal {
    (address lendingPool, address weth) =
SPHubUtils.getLendingPoolAndWETH(address(factory));
    uint256 assetBalance = SPHubUtils.getTokenBalance(asset);

    if (asset == weth) {
        ...
    } else if (assetBalance < interestAmount) {
        withdrawBalances();
        _swapETHForToken(asset, interestAmount, maxAmountIn);
    }
}
```

Recommendation

Correct the expected amountOut as `interestAmount - assetBalance`.

Status

This issue has been resolved by the team with commit 3ec88d7.

7. Improper premium calculation

Severity: Low	Category: Business Logic

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

Description

When one SPHub is unhealthy, the liquidator will liquidate this SPHub and charge one part of the premium according to the repaid debt amount.
The problem here is that we will calculate the premium directly via the input `liquidationAmount`. But the actual debt repaid may be less than `liquidatedAmount`.

For example:

1. Alice has 1000 USDC debt and her SPHub is unhealthy.
2. Alice repays 500 USDC.
3. At the same time with Step 2, the liquidator triggers this liquidation via `liquidationCallWithoutPrepare` to liquidate 1000 USDC.
4. Alice will be charged a premium for 1000 USDC. This is unfair for Alice.

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

```
function liquidationCallWithoutPrepare(address asset, uint256 liquidationAmount, uint256
maxAmountIn) public payable onlyLiquidator {
    uint256 premiumAmount = (liquidationAmount * factory.getLiquidationPremium()) /
10000;
    uint256 liquidatedAmount = liquidationAmount - premiumAmount;

    if(asset == weth) {
        // Make sure that we have enough FIL in the contract to pay.
        uint256 collateralInContract = address(this).balance;
        require(collateralInContract >= liquidationAmount, "Insufficient collateral to
liquidate");

        // Take care of premium
        SPHubUtils.safeTransferETH(treasury, premiumAmount);

        uint256 wethRepayAmount = wethDebt < liquidatedAmount ? wethDebt :
liquidatedAmount;
        if (wethRepayAmount > 0) {
            _handleETHRepayment(wethRepayAmount, true);
        }
    }
}
```

Recommendation

Calculate the premium according to the actual repaid debt.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

8. The key functions cannot be paused

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

Description

The protocol allows the spHub owner to request the spHub contract to enter a paused state. However, due to the lack of the `whenNotPaused` modifier, no functions will be paused when the contract is in a paused state.

Filus/contracts/protocol/StorageProvider/SPHub.sol:L839-L850

```
function requestPauseOrUnpause() public onlyHubOwner {
    requestedPause = true;
    emit PauseRequested();
}

function confirmPause(bool accept) public onlyOperator {
    if (accept) {
        _pause();
    }
    requestedPause = false;
    emit PauseConfirmed(accept);
}
```

Recommendation

The protocol is expected to pause the `addMiner`, `removeMiner`, `pullFunds`, `borrowETH`, and `borrow` functions when the contract is paused. Therefore, it is recommended to add the `whenNotPaused` modifier to these functions.

Status

This issue has been resolved by the team with commit 3ec88d7.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 9e899b2:

File	SHA-1 hash
SPHubFactory.sol	c184ac66ec77ca1f5e881bb510513473bbd5ee4c
SPHub.sol	4777aed78e6231585763639f1fb5e4aa99dd432d
SPDataProvider.sol	51e32e1b0bfe8ecb20e2400f64da9266a2cd34f2
SPValidationLogic.sol	8b882f8ce83f460ad7217213ebd32985c59c6346
SPHubUtils.sol	84fbb478e057787d6abce63898479710398ff519