SALUS SECURITY

APR 2024

CODE SECURITY ASSESSMENT

FILLIQUID

Overview

Project Summary

- Name: FILLiquid
- Platform: Filecoin
- Language: Solidity
- Repository:
    - https://github.com/FILL-Lab/FILLiquid
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	FILLiquid
Version	v3
Type	Solidity
Dates	Apr 26 2024
Logs	Apr 12 2024; Apr 24 2024; Apr 26 2024

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	4
Total Low-Severity issues	1
Total informational issues	4
Total	10

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	Borrowers can borrow FIL via a low borrow rate	High	Business Logic	Resolved
2	Possible arbitrage via front-run borrower's payback	Medium	Business Logic	Resolved
3	Liquidation causes borrowers to lose rewards they have accumulated	Medium	Business Logic	Resolved
4	Voting results could be changed after the voting period	Medium	Business Logic	Resolved
5	Centralization risk	Medium	Centralization	Resolved
6	Potential cross-contract signature replay	Low	Cryptography	Resolved
7	Inconsistency between interface and implementation	Informational	Inconsistency	Resolved
8	Inconsistency between comments and documentation	Informational	Inconsistency	Resolved
9	Typos	Informational	Code Quality	Resolved
10	Gas optimization suggestions	Informational	Gas Optimization	Resolved

2.2 Notable Findings

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

1. Borrowers can borrow FIL via a low borrow rate

Severity: High	Category: Business Logic

Target:
- contracts/FILLiquid.sol

Description

When borrowers want to borrow FILs, FILLiquid will calculate the current borrow utilization rate and make use of the borrow utilization rate to calculate the borrow interest rate. Borrowers' borrowing interest rate is stable. When borrowers pay back, FILLiquid will make use of this stable borrowing rate to calculate the whole loan interest.

Users can manipulate the borrow utilization easily via the deposit() or redeem() function. For example, users can decrease the borrowing utilization rate to a low value via FILLiquid::deposit(), and then users can borrow FILs via a low borrow interest rate.

contracts/FILLiquid.sol:L409-L417

```
function borrow(uint64 minerId, uint amount, uint expectInterestRate) external
isBindMiner(minerId) haveCollateralizing(minerId) returns (uint, uint) {
    ...
    uint realInterestRate = interestRateBorrow(amount);
    isHigher(expectInterestRate, realInterestRate);
    ...
}
```

contracts/FILLiquid.sol:L674-L676

```
function interestRateBorrow(uint amount) public view returns (uint) {
        return _calculation.getInterestRate(utilizationRateBorrow(amount), _u_1, _r_0,
_r_1, _rateBase, _n);
}
```

contracts/FILLiquid.sol:L650-L656

```
function utilizationRateBorrow(uint amount) public view returns (uint) {
    uint total = totalFILLiquidity();
    require(total != 0, "Total liquidity is 0");
    uint utilized = utilizedLiquidity() + amount;
    require(utilized < total, "Utilized liquidity exceeds total");
    return utilized * _rateBase / total;
}
```

Recommendation

In AAVE, there is one rebalance mechanism. It means when the stable borrow rate is lower than the lend interest rate, this position can be rebalanced and the borrow interest rate will be updated to the latest borrow interest rate. Consider bringing one similar rebalance

mechanism into FILLiquid. For example, if one borrow position's borrow interest rate is much lower than the current borrow rate, or much lower than today/this week's average borrowing interest rate, this borrow position's borrow interest rate can be rebalanced to the latest borrowing rate.

Status

This issue has been resolved by the team with commit c0ea7b5. The project team has added a minimum locking period for newly minted FITs to avoid quick redemption after depositing.

2. Possible arbitrage via front-run borrower's payback

Severity: Medium	Category: Business Logic
Target: - contracts/FILLiquid.sol	

Description

In FILLiquid, lenders earn profit via the growing value of FIT. When borrowers pay back debts, more FILs enter the contract, which leads to the FIT value increase.

If the lender deposits FILs via front-run a huge amount of debt payback, the lender's FIT's price will increase immediately after borrowers pay back via FILLiquid::directPayback()/withdraw4Payback(). When the debt is large enough, lenders can redeem it to earn the arbitrage profit considering the redemption fee.

contracts/FILLiquid.sol:L630-L632

```
function totalFILLiquidity() public view returns (uint) {
    return _accumulatedDepositFIL + _accumulatedInterestFIL - _accumulatedRedeemFIL -
_accumulatedRedeemFee - _accumulatedBadDebt;
}
```

contracts/FILLiquid.sol:L1032-L1083

```
function paybackProcess(uint64 minerId, uint amount) private returns (uint[3] memory r)
{
    ...
    _accumulatedPaybackFIL += r[1];
    _accumulatedInterestFIL += r[2];
    ...
}
```

Recommendation

Consider adding one minimum deposit locking period to mitigate the similar arbitrage possibility.

Status

This issue has been resolved by the team with commit c0ea7b5.

3. Liquidation causes borrowers to lose rewards they have accumulated

Severity: Medium	Category: Business Logic
Target: - contracts/FILLiquid.sol	

Description

Borrowers will get some FIG tokens as one incentive when borrowers pay back their debt. In the liquidation process, the liquidator makes use of the borrower's collateral to pay back the borrower's debt. From the view of the borrower, the principal and related interest are paid back and they deserve to receive some FIG tokens.

contracts/FILLiquid.sol:L484-L491

```
function directPayback(uint64 minerId) external isBorrower(minerId) payable returns
(uint, uint, uint) {
        uint[3] memory r = paybackProcess(minerId, msg.value);
        address sender = _msgSender();
        if (r[0] > 0) payable(sender).transfer(r[0]);
        uint mintedFIG = _fitStake.handleInterest(_minerBindsMap[minerId], r[1], r[2]);
        emit Payback(sender, minerId, minerId, r[1], r[2], 0, mintedFIG);
        return (r[1], r[2], mintedFIG);
}
```

Recommendation

In the liquidation process, consider minting corresponding FIG tokens for borrowers according to the payback amount.

Status

This issue has been resolved by the team with commit c0ea7b5.

4. Voting results could be changed after the voting period

Severity: Medium	Category: Business Logic
Target: - contracts/Governance.sol	

Description

The proposer can propose a proposal via Governance::propose(). Voters can vote for this proposal before the deadline. When the deadline is reached, nobody can vote for this proposal. It is expected to have a certain clear voting result. However, this voting result can be changed by manipulating the variable _totalBondedAmount.

contracts/Governance.sol:L435-L449

```
function _voteResult(VotingStatusInfo storage info) private view returns (voteResult
result) {
        uint amountTotal = info.amountTotal;
        uint amountYes = info.amounts[uint(voteCategory.yes)];
        uint amountNo = info.amounts[uint(voteCategory.no)];
        uint amountNoWithVeto = info.amounts[uint(voteCategory.noWithVeto)];
        if (amountNoWithVeto > 0 && amountNoWithVeto * _rateBase >= amountTotal *
_maxNoWithVeto) {
            result = voteResult.rejectedWithVeto;
        } else if ((amountNo + amountNoWithVeto) * _rateBase >= amountTotal * _maxNo) {
            result = voteResult.rejected;
        } else if (amountYes * _rateBase >= amountTotal * _minYes && amountTotal *
_rateBase >= _totalBondedAmount * _quorum) {
            result = voteResult.approved;
        } else {
            result = voteResult.rejected;
        }
    }
```

contracts/Governance.sol:L142-L150

```
function bond(uint amount) external {
    address sender = _msgSender();
    _tokenFILGovernance.withdraw(sender, amount);
    if (_bondings[sender] == 0) _numberOfBonders++;
    _bondings[sender] += amount;
    _totalBondedAmount += amount;

    emit Bonded(sender, amount);
}
```

Recommendation

Please refer to Openzepplin governance implementation and refactor this part.

Status

This issue has been resolved by the team with commit c0ea7b5.

5. Centralization risk

Severity: Medium	Category: Centralization

Target:
- contracts/FILLiquid.sol

Description

There are some privileged roles in the FILLiquid codebase.

The owner of the FILLiquid contract can update key smart contract addresses.

Should the owner's private key be compromised, an attacker could update the related contracts to some malicious smart contract.

contracts/FILLiquid.sol:L823-L839

```
function setAdministrativeFactors(
    address new_tokenFILTrust,
    address new_validation,
    address new_calculation,
    address new_filecoinAPI,
    address new_fitStake,
    address new_governance,
    address payable new_foundation
) onlyOwner external {
    _tokenFILTrust= FILTrust(new_tokenFILTrust);
    _validation = Validation(new_validation);
    _calculation = Calculation(new_calculation);
    _filecoinAPI = FilecoinAPI(new_filecoinAPI);
    _fitStake = FITStake(new_fitStake);
    _governance = new_governance;
    _foundation = new_foundation;
}
```

Similar centralization issue exists in the Governance contract. The owner of the Governance contract can update key configuration parameters.

contracts/Governance.sol:L356-L388

```
function setFactors(
    uint new_rateBase,
    uint new_minYes,
    uint new_maxNo,
    uint new_maxNoWithVeto,
    uint new_quorum,
    uint new_liquidate,
    uint new_depositRatioThreshold,
    uint new_depositAmountThreshold,
    uint new_voteThreshold,
    uint new_votingPeriod,
    uint new_executionPeriod,
    uint new_maxActiveProposals
    ) onlyOwner external {
    ...
}
```

If the privileged account is a plain EOA account, this can be worrisome and pose a risk to the other users.

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 resolved by the team with commit 658a102. The project team will renounce the ownership after deployment.

6. Potential cross-contract signature replay

Severity: Low	Category: Cryptography

Target:
- contracts/Utils/Validation.sol

Description

The validateOwner() function is used to verify if a sender has been approved by the miner's owner to collateralize the miner in the FILLiquid contract. However, the message to sign does not contain the address of the FILLiquid contract, making it vulnerable to cross-contract signature replay attacks.

contracts/Utils/Validation.sol:L13-L33

```
function validateOwner(
        uint64 minerID,
        bytes calldata signature,
        address sender
) external {
        ...
        bytes memory digest = _getDigest(
            ownerAddr.data,
            minerID,
            sender
        );
        int256 exitCode = AccountAPI.authenticateMessage(
        CommonTypes.FilActorId.wrap(PrecompilesAPI.resolveAddress(ownerAddr)),
            AccountTypes.AuthenticateMessageParams({
               signature: signature,
               message: digest
            })
        );
```

contracts/Utils/Validation.sol:L46-L60

```
function _getDigest(
        bytes memory ownerAddr,
        uint64 minerID,
        address sender
) private view returns (bytes memory) {
        bytes32 digest = keccak256(abi.encode(
            keccak256("validateOwner"),
            ownerAddr,
            minerID,
            sender,
            _nonces[ownerAddr],
            _getChainId()
        ));
        return bytes.concat(digest);
}
```

Recommendation

Consider including the Validation contract address in the message to avoid cross-contract signature replay attacks.

Status

This issue has been resolved by the team with commit c0ea7b5.

2.3 Informational Findings

7. Inconsistency between interface and implementation

Severity: Informational	Category: Inconsistency
Target: - contracts/FILLiquid.sol	

Description

In the FILLiquidInterface, the last parameter of the deposit() and redeem() functions is exchangeRate, while the last parameter in the implementation is expectAmountFILTrust and expectAmountFIL.

contracts/FILLiquid.sol:L96-L106

```
/// @dev deposit FIL to the contract, mint FILTrust
/// @param exchangeRate approximated exchange rate at the point of request
/// @return amount actual FILTrust minted
function deposit(uint exchangeRate) external payable returns (uint amount);

/// @dev redeem FILTrust to the contract, withdraw FIL
/// @param amountFILTrust the amount of FILTrust user would like to redeem
/// @param exchangeRate approximated exchange rate at the point of request
/// @return amount actual FIL withdrawal
/// @return fee fee deducted
function redeem(uint amountFILTrust, uint exchangeRate) external returns (uint amount,
uint fee);
```

contracts/FILLiquid.sol:L375,L391

```
function deposit(uint expectAmountFILTrust) external payable returns (uint)
function redeem(uint amountFILTrust, uint expectAmountFIL) external returns (uint, uint)
```

Recommendation

Consider updating the parameter in the interface to match the implementation.

Status

This issue has been resolved by the team with commit c0ea7b5.

8. Inconsistency between comments and documentation

Severity: Informational	Category: Inconsistency
Target: - contracts/Governance.sol	

Description

According to the implementation and the whitepaper, to proceed to the voting stage, proposers should deposit at least the higher of 0.001% of the FIG circulating supply or 10,000 FIG.

contracts/Governance.sol:L292-L293

```
///     Proposals that have been deposited with at least the higher of 0.01% of FIG
circulating supply or
///     500 FIG, will proceed to voting stage.
```

Recommendation

Consider updating the numerical value in the comments.

Status

This issue has been resolved by the team with commit c0ea7b5.

9. Typos

Severity: Informational	Category: Code Quality
Target: - contracts/Governance.sol - contracts/FILLiquid.sol	

Description

There are typos in the codes below.

contracts/Governance.sol:L11

```
enum proposolCategory
```

proposolCategory should be proposalCategory.

contracts/FILLiquid.sol:L184

```
/// @dev return borrowing interest rate: a mathematical function of utilizatonRate
```

utilizatonRate should be utilizationRate.

Recommendation

Consider fixing the typos.

Status

This issue has been resolved by the team with commit c0ea7b5.

10. Gas optimization suggestions

Severity: Informational	Category: Gas Optimization

Target:
- contracts/MultiSignFactory.sol
- contracts/Deployer1.sol
- contracts/Deployer2.sol
- contracts/Deployer3.sol

Description

When updating the _isSigner mapping in the renewSigners() function, it is recommended to use the signers array instead of the _signers array stored in the storage to save gas.

contracts/MultiSignFactory.sol:L90-L100

```
function renewSigners(address[] calldata signers, uint approvalThreshold)
isValidSignerCount(approvalThreshold, signers.length) senderIsSelf external {
        ...
        _signers = signers;
        for (uint i = 0; i < _signers.length; i++) {
            _isSigner[_signers[i]] = true;
        }
        ...
}
```

The state variables in the Deployer1, Deployer2, and Deployer3 contracts are set in the constructor and can not be updated after deployment. Thus, they can be changed to immutable to save gas.

contracts/Deployer1.sol:L11-L16

```
Validation private _validation;
Calculation private _calculation;
FilecoinAPI private _filecoinAPI;
FILTrust private _filTrust;
FILGovernance private _filGovernance;
address private _owner;
```

Recommendation

Consider following the above suggestions to save gas.

Status

This issue has been resolved by the team with commit c0ea7b5.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 2abda0c:

File	SHA-1 hash
fea5f1031c9bde79bf399a54c0fb1c2a28deb0cf	contracts/DataFetcher.sol
baa8dc436c340f5d34944e7f455b135f23fce9c1	contracts/Deployer1.sol
1fb952d063574f69a46852d177fd942ca9091204	contracts/Deployer2.sol
1ab117e0db3edf1529b17b1759d25cf3f01d540d	contracts/Deployer3.sol
3aefe070830511ab26de78be7af7dd75a51b7d31	contracts/ERC20Pot.sol
9d420c15ed225ff1f9a7373a9f701fe0347dc504	contracts/FILGovernance.sol
52e68a990ce44fee0bfaf152ab628a409981a2db	contracts/FILLiquid.sol
df0b82ba422268f2db0c258905a3d39f516a39fd	contracts/FILTrust.sol
ac3ce7fbf175ead83b0e8b5c7abe7f1d1289f903	contracts/FITStake.sol
6ad505986dea6f3152c092eb7ad529458d2fd249	contracts/Governance.sol
da3627b242dbc2596404de4c89c2de7d0df9d3af	contracts/MultiSignFactory.sol
3158ff97b60bb9a883e2379e07a9d9751272a0eb	contracts/Utils/Calculation.sol
848fde09c22a6164cf3a8629d00979e80fdeef21	contracts/Utils/Conversion.sol
51f4193b9ea47675ebd34be4d413babaaca87a00	contracts/Utils/FilecoinAPI.sol
84148d49d26a87ef5ed1f5e3583642accb9a892e	contracts/Utils/Validation.sol