SALUS SECURITY

JULY 2023

CODE SECURITY ASSESSMENT

SOLV PROTOCOL

Overview

Project Summary

- Name: Solv Protocol - Open-end Fund Business
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/solv-finance/solv-contracts-v3
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Solv Protocol - Open-end Fund Business
Version	v2
Type	Solidity
Dates	July 31 2023
Logs	July 26 2023; July 31 2023

Vulnerability Summary

Total High-Severity issues	2
Total Medium-Severity issues	3
Total Low-Severity issues	2
Total informational issues	9
Total	16

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	Attackers can DOS the fund pool by repaying	High	Denial of Service	Resolved
2	Incorrect slot handled in OpenFundMarket.setRedeemNav()	High	Business Logic	Resolved
3	redeemNavManager can not update the fundraising time	Medium	Access Control	Resolved
4	Incorrect calculation in claimableValue()	Medium	Business Logic	Resolved
5	Users are unable to claim all value	Medium	Business Logic	Resolved
6	Precision loss in OpenFundMarket.subscribe()	Low	Numerics	Resolved
7	Invalid time check in closeCurrentRedeemSlot()	Low	Data Validation	Resolved
8	Lack of validation for setting latestSetNavTime	Informational	Data Validation	Resolved
9	Inappropriate solidity version requirement	Informational	Configuration	Resolved
10	Incorrect gap size	Informational	Configuration	Acknowledged
11	Missing check for the equivalence of valueDecimals	Informational	Data Validation	Resolved
12	Missing explicit check for poolId existence	Informational	Data Validation	Resolved
13	Missing event parameters	Informational	Logging	Resolved
14	Redundant usage of boolean literal	Informational	Gas Optimization	Resolved
15	Improper error message	Informational	Logging	Resolved
16	Redundant code	Informational	Redundancy	Resolved

2.2 Notable Findings

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

1. Attackers can DOS the fund pool by repaying

Severity: High	Category: Denial of Service

Target:
- markets/open-fund-market/contracts/OpenFundMarket.sol
- sft/abilities/contracts/fcfs-multi-repayable/FCFSMultiRepayableDelegate.sol
- sft/abilities/contracts/fcfs-multi-repayable/FCFSMultiRepayableConcrete.sol

Description

sft/abilities/contracts/fcfs-multi-repayable/FCFSMultiRepayableConcrete.sol:L78-L83

```
function _beforeMint(uint256 /** tokenId_ */, uint256 slot_, uint256 mintValue_)
internal virtual {
        // skip repayment check when minting in the process of transferring from id to
address
        if (mintValue_ > 0) {
        require(_slotRepayInfo[slot_].repaidCurrencyAmount == 0, "FMR: already repaid");
        }
}
```

When minting an OpenFundRedemption SFT, the above _beforeMint() check is invoked via the following call path:

- OpenFundRedemptionDelegate._beforeValueTransfer()
- FCFSMultiRepayableDelegate._beforeValueTransfer()
- FCFSMultiRepayableConcrete.mintOnlyDelegate()
- FCFSMultiRepayableConcrete._beforeMint()

When mintValue is non-zero, _beforeMint() reverts if the repaidCurrencyAmount for a slot is non-zero.

sft/abilities/contracts/fcfs-multi-repayable/FCFSMultiRepayableDelegate.sol:L12-L22

```
function repay(uint256 slot_, address currency_, uint256 repayCurrencyAmount_) external
payable virtual override nonReentrant {
        IFCFSMultiRepayableConcrete(concrete()).repayOnlyDelegate(_msgSender(), slot_,
currency_, repayCurrencyAmount_);
        ...
}

function repayWithBalance(uint256 slot_, address currency_, uint256
repayCurrencyAmount_) external payable virtual override nonReentrant {
        require(allowRepayWithBalance(), "MultiRepayableDelegate: cannot repay with
balance");

IFCFSMultiRepayableConcrete(concrete()).repayWithBalanceOnlyDelegate(_msgSender(),
slot_, currency_, repayCurrencyAmount_);
        emit Repay(slot_, _msgSender(), repayCurrencyAmount_);
}
```

sft/abilities/contracts/fcfs-multi-repayable/FCFSMultiRepayableConcrete.sol:L19-L33

```
function repayOnlyDelegate(address txSender_, uint256 slot_, address currency_, uint256
repayCurrencyAmount_) external payable virtual override onlyDelegate {
        _beforeRepay(txSender_, slot_, currency_, repayCurrencyAmount_);
        _slotRepayInfo[slot_].repaidCurrencyAmount += repayCurrencyAmount_;
        ...
}

function repayWithBalanceOnlyDelegate(address txSender_, uint256 slot_, address
currency_, uint256 repayCurrencyAmount_) external payable virtual override onlyDelegate
{
        ...
        _slotRepayInfo[slot_].repaidCurrencyAmount += repayCurrencyAmount_;
        ...
}
```

However, the repaidCurrencyAmount can be increased by anyone through the use of repay() or repayWithBalance(), which can lead to a denial-of-service (DOS) attack on the minting of new OpenFundRedemption SFTs for a slot.

Here's a possible attack scenario:

1. The issuer creates an open-end fund pool
2. Buyers subscribe to the pool
3. One buyer initiates requestRedeem()
4. The attacker reads the redeemSlot from the transaction log
5. The attacker repays a non-zero amount of currency to the OpenFundRedemption SFT for this redeemSlot
6. As a result, subsequent requestRedeem() calls will fail, preventing other buyers from redeeming their funds.

sft/abilities/contracts/multi-repayable/MultiRepayableConcrete.sol:L67-L69

```
function repaidCurrencyAmount(uint256 slot_) public view virtual override returns
(uint256) {
        return _slotRepayInfo[slot_].repaidCurrencyAmount;
}
```

sft/abilities/contracts/multi-repayable/MultiRepayableConcrete.sol:L102-L107

```
function _beforeMint(uint256 /** tokenId_ */, uint256 slot_, uint256 mintValue_)
internal virtual {
        // skip repayment check when minting in the process of transferring from id to
address
        if (mintValue_ > 0) {
        require(repaidCurrencyAmount(slot_) == 0, "MultiRepayableConcrete: already
repaid");
        }
}
```

In the same vein, when minting an OpenFundShare SFT, a similar non-zero repaidCurrencyAmount check is performed. If repaidCurrencyAmount is non-zero, the minting will fail.

Attackers can use OpenFundShare SFT's repay() or repayWithBalance() to DOS the minting of new OpenFundShare SFT for a slot.

Here's a possible attack scenario:

1. The issuer creates a fund pool
2. The attacker reads the OpenFundShare slot
3. The attacker repays a non-zero amount of currency to this slot

4. As a result, subsequent subscribe() calls will fail, preventing buyers from subscribing to this pool.

This issue could affect functions in the OpenFundMarket contract that rely on the minting of OpenFundShare and OpenFundRedemption SFTs.

Recommendation

It is recommended to add proper access controls to repay() and repayWithBalance() to avoid unexpected repayments.

Status

This issue has been resolved by the team. The limit in the _beforeMint() has been removed.

2. Incorrect slot handled in OpenFundMarket.setRedeemNav()

Severity: High	Category: Business Logic

Target:
- markets/open-fund-market/contracts/OpenFundMarket.sol

Description

markets/open-fund-market/contracts/OpenFundMarket.sol:L217-L241

```
function setRedeemNav(bytes32 poolId_, uint256 redeemSlot_, uint256 nav_, uint256
currencyBalance_) external virtual override nonReentrant {
        ...


ISFTValueIssuableDelegate(poolInfo.poolSFTInfo.openFundShare).burnOnlyIssueMarket(_poolR
edeemTokenId[poolInfo.poolSFTInfo.latestRedeemSlot], 0);

OpenFundRedemptionDelegate(poolInfo.poolSFTInfo.openFundRedemption).setRedeemNavOnlyMark
et(poolInfo.poolSFTInfo.latestRedeemSlot, settledNav);
        ...
}
```

The burnOnlyIssueMarket and setRedeemNavOnlyMarket statements handle the latestRedeemSlot, but they should handle the passed-in redeemSlot_.

The redeem nav can only be set after the slot is closed. When closing a slot, the latestRedeemSlot will be updated with the newly created slot. If two closeCurrentRedeemSlot() functions are called without a setRedeemNav() call in between, the previously closed redeemSlot will not be able to be processed by setRedeemNav().

setRedeemNav() will set the redeem nav of a slot. And _redeemInfos[slot_].nav is used to calculate claimable value. Users can not claim if _redeemInfos[slot_].nav is zero.

sft/abilities/contracts/fcfs-multi-repayable/FCFSMultiRepayableConcrete.sol:L59

```
uint256 claimableValue_ = _slotRepayInfo[slot].currencyBalance *
Constants.FULL_PERCENTAGE * REPAY_RATE_SCALAR * (10 ** valueDecimals) / _repayRate(slot)
/ (10 ** currencyDecimals);
```

sft/payable/open-fund/contracts/open-fund-redemptions/OpenFundRedemptionConcrete.sol:L71-L73

```
function _repayRate( uint256 slot_) internal view virtual override returns (uint256) {
        return _redeemInfos[slot_].nav;
}
```

Recommendation

Consider changing the highlighted poolInfo.poolSFTInfo.latestRedeemSlot to redeemSlot_.

Status

This issue has been resolved by the team.

3. redeemNavManager can not update the fundraising time

Severity: Medium	Category: Access Control

Target:
- markets/open-fund-market/contracts/OpenFundMarket.sol

Description

The owner of the contract and the redeemNavManager should have permission to update the fundraising end time. However, the onlyOwner modifier will prevent the redeemNavManager from doing so.

markets/open-fund-market/contracts/OpenFundMarket.sol:L282-L287

```
function updateFundraisingEndTime(bytes32 poolId_, uint64 newEndTime_) external virtual
nonReentrant onlyOwner {
        PoolInfo storage poolInfo = poolInfos[poolId_];
        require(_msgSender() == owner || _msgSender() ==
poolInfo.managerInfo.redeemNavManager, "OFM: only owner or redeem nav manager");
        emit UpdateFundraisingEndTime(poolId_,
poolInfo.subscribeLimitInfo.fundraisingEndTime, newEndTime_);
        poolInfo.subscribeLimitInfo.fundraisingEndTime = newEndTime_;
}
```

Recommendation

Consider removing the onlyOwner modifier.

Status

This issue has been resolved by the team.

4. Incorrect calculation in claimableValue()

Severity: Medium	Category: Business Logic

Target:
- sft/abilities/contracts/fcfs-multi-repayable/FCFSMultiRepayableConcrete.sol
- sft/abilities/contracts/fcfs-multi-repayable/FCFSMultiRepayableDelegate.sol

Description

sft/payable/open-fund/contracts/open-fund-redemptions/OpenFundRedemptionConcrete.sol:L71-L73

```
function _repayRate( uint256 slot_) internal view virtual override returns (uint256) {
        return _redeemInfos[slot_].nav;
}
```

repayRate(slot) returns the nav set by the redeemNavManager through setRedeemNav().

markets/open-fund-market/contracts/OpenFundMarket.sol:L217-L241

```
function setRedeemNav(bytes32 poolId_, uint256 redeemSlot_, uint256 nav_, uint256
currencyBalance_) external virtual override nonReentrant {
        ...
        uint256 settledNav = nav_ * (currencyBalance_ - carryAmount) / currencyBalance_;

        ...

OpenFundRedemptionDelegate(poolInfo.poolSFTInfo.openFundRedemption).setRedeemNavOnlyMark
et(poolInfo.poolSFTInfo.latestRedeemSlot, settledNav);
        ...
}
```

According to the calculation logic for settledNav, the nav should be in line with the currency's decimal scale.

sft/abilities/contracts/fcfs-multi-repayable/FCFSMultiRepayableConcrete.sol:L54-L61

```
function claimableValue(uint256 tokenId_) public view virtual override returns (uint256)
{
        uint256 slot = ERC3525Upgradeable(delegate()).slotOf(tokenId_);
        uint256 balance = ERC3525Upgradeable(delegate()).balanceOf(tokenId_);
        uint8 valueDecimals = ERC3525Upgradeable(delegate()).valueDecimals();
        uint8 currencyDecimals = ERC20(_currency(slot)).decimals();
        uint256 claimableValue_ = _slotRepayInfo[slot].currencyBalance *
Constants.FULL_PERCENTAGE * REPAY_RATE_SCALAR * (10 ** valueDecimals) / _repayRate(slot)
/ (10 ** currencyDecimals);
        return claimableValue_ > balance ? balance : claimableValue_;
}
```

In the OpenFundRedemption SFT's claimableValue() function, nav is calculated using the scale of Constants.FULL_PERCENTAGE * REPAY_RATE_SCALAR, which equals 1e12. Therefore, if the currency's decimals are larger than 12, the calculated claimableValue will be lower than expected.

sft/abilities/contracts/fcfs-multi-repayable/FCFSMultiRepayableDelegate.sol:L24-L40

```
function claimTo(address to_, uint256 tokenId_, address currency_, uint256 claimValue_)
external virtual override nonReentrant {
        ...
        uint256 claimableValue =
IFCFSMultiRepayableConcrete(concrete()).claimableValue(tokenId_);
        require(claimValue_ <= claimableValue, "MultiRepayableDelegate: over claim");

        if (claimValue_ == ERC3525Upgradeable.balanceOf(tokenId_)) {
            ERC3525Upgradeable._burn(tokenId_);
        } else {
            ERC3525Upgradeable._burnValue(tokenId_, claimValue_);
        }
        ...
}
```

For example, if the currency is DAI, the redeem nav is set to 1.05 * 1e18. A user has 1 OpenFundRedemption SFT token with 1e18 value. Instead of 1e18, the maximum value he can claim is about 9.5 * 1e12.

Recommendation

Make sure the calculation in the claimableValue() function aligns with the calculation logic in the claimTo() function.

Status

This issue has been resolved by the team.

5. Users are unable to claim all value

Severity: Medium	Category: Business Logic
Target: - sft/abilities/contracts/fcfs-multi-repayable/FCFSMultiRepayableDelegate.sol	

Description

If the claimValue_ equals the value of a token, the token will be burned. However, the claimableValue() called in claimOnlyDelegate() needs a valid tokenId to get the slot and balance. If the token is burned, the transaction will be reverted with “ERC3525: invalid token ID”.

Meanwhile, burning the value before claimOnlyDelegate() will result in a smaller return value of claimableValue() in the claimOnlyDelegate() compared to claimTo(). This is also possible to cause the transaction to be reverted.

sft/abilities/contracts/fcfs-multi-repayable/FCFSMultiRepayableDelegate.sol:L24-L40

```
function claimTo(address to_, uint256 tokenId_, address currency_, uint256 claimValue_)
external virtual override nonReentrant {
        ...

        if (claimValue_ == ERC3525Upgradeable.balanceOf(tokenId_)) {
            ERC3525Upgradeable._burn(tokenId_);
        } else {
            ERC3525Upgradeable._burnValue(tokenId_, claimValue_);
        }

        uint256 claimCurrencyAmount =
IFCFSMultiRepayableConcrete(concrete()).claimOnlyDelegate(tokenId_, slot, currency_,
claimValue_);
        ...
}
```

sft/abilities/contracts/fcfs-multi-repayable/FCFSMultiRepayableConcrete.sol:L54-L61

```
function claimableValue(uint256 tokenId_) public view virtual override returns (uint256)
{
        uint256 slot = ERC3525Upgradeable(delegate()).slotOf(tokenId_);
        uint256 balance = ERC3525Upgradeable(delegate()).balanceOf(tokenId_);
        ...
}
```

Recommendation

Consider placing the burn logic after calculating the claimCurrencyAmount.

Status

This issue has been resolved by the team.

6. Precision loss in OpenFundMarket.subscribe()

Severity: Low	Category: Numerics

Target:
- markets/open-fund-market/contracts/OpenFundMarket.sol

Description

markets/open-fund-market/contracts/OpenFundMarket.sol:L96-L138

```
function subscribe(bytes32 poolId_, uint256 currencyAmount_, uint256 openFundShareId_,
uint64 expireTime_)
        external virtual override nonReentrant returns (uint256 value_)
{
        ...
        value_ = (currencyAmount_ * ( 10 **
IERC3525(poolInfo.poolSFTInfo.openFundShare).valueDecimals())) / nav;

        ...

        uint256 tokenId;
        if (openFundShareId_ == 0) {
        tokenId = ISFTValueIssuableDelegate(poolInfo.poolSFTInfo.openFundShare)
                .mintOnlyIssueMarket(_msgSender(), poolInfo.currency, _msgSender(),
poolInfo.poolSFTInfo.openFundShareSlot, value_);
        } else {
        require(IERC3525(poolInfo.poolSFTInfo.openFundShare).slotOf(openFundShareId_) ==
poolInfo.poolSFTInfo.openFundShareSlot, "OFM: slot not match");

ISFTValueIssuableDelegate(poolInfo.poolSFTInfo.openFundShare).mintValueOnlyIssueMarket(
                _msgSender(), poolInfo.currency, openFundShareId_, value_
        );
        tokenId = openFundShareId_;
        }
        ...
}
```

The value_ will be 0 when `(currencyAmount_ * (10 ** IERC3525(poolInfo.poolSFTInfo.openFundShare).valueDecimals()))` is less than `nav`. This is likely to occur when the valueDecimals of the OpenFundShare SFT is much less than the decimals of the currency.

In this case, the user sends currencyAmount_ currency to the contract without receiving OpenFundShare values in return, resulting in an unexpected loss for the user.

Recommendation

It is recommended to revert when value_ is zero.

Status

This issue has been resolved by the team.

7. Invalid time check in closeCurrentRedeemSlot()

Severity: Low	Category: Data Validation

Target:
- markets/open-fund-market/contracts/OpenFundMarket.sol

Description

markets/open-fund-market/contracts/OpenFundMarket.sol:L190-L208

```
function closeCurrentRedeemSlot(bytes32 poolId_) external virtual override nonReentrant
{
        ...
        require(block.timestamp -
poolRedeemSlotCloseTime[poolInfo.poolSFTInfo.latestRedeemSlot] >= 24 * 60 * 60, "OFM:
redeem period less than 24h");

        ...

        uint256 previousRedeemSlot = poolInfo.poolSFTInfo.latestRedeemSlot;
        poolRedeemSlotCloseTime[previousRedeemSlot] = block.timestamp;
        poolInfo.poolSFTInfo.latestRedeemSlot =
ISFTValueIssuableDelegate(poolInfo.poolSFTInfo.openFundRedemption).createSlotOnlyIssueMa
rket(_msgSender(), abi.encode(nextRedeemInfo));
        ...
}
```

`require(block.timestamp - poolRedeemSlotCloseTime[poolInfo.poolSFTInfo.latestRedeemSlot] >= 24 * 60 * 60, "OFM: redeem period less than 24h");` is invalid.

poolRedeemSlotCloseTime is only updated in the closeCurrentRedeemSlot() function. After updating the poolRedeemSlotCloseTime variable for the latestRedeemSlot, the poolInfo.poolSFTInfo.latestRedeemSlot variable is set to a new slot value.

As a result, `poolRedeemSlotCloseTime[poolInfo.poolSFTInfo.latestRedeemSlot]` is zero in the highlighted require statement, making this check useless.

Recommendation

The poolRedeemSlotCloseTime for the previous redeem slot should be tracked and used.

Status

This issue has been resolved by the team.

2.3 Informational Findings

8. Lack of validation for setting latestSetNavTime

Severity: Informational	Category: Data Validation
Target: - markets/open-fund-market/contracts/oracle/NavOracle.sol	

Description

The manager can update subscribe nav through the setSubscribeNav() function.

markets/open-fund-market/contracts/OpenFundMarket.sol:L210-L215

```
function setSubscribeNav(bytes32 poolId_, uint256 time_, uint256 nav_) external virtual
override {
        PoolInfo storage poolInfo = poolInfos[poolId_];
        require(_msgSender() == poolInfo.managerInfo.subscribeNavManager, "OFM: only
subscribe nav manager");
        INavOracle(poolInfo.navOracle).setSubscribeNavOnlyMarket(poolId_, time_, nav_);
        emit SetSubscribeNav(poolId_, time_, nav_);
}
```

Since the argument time_ is controlled by the manager, there is no guarantee that dayTime is greater than latestSetNavTime. This may affect the return value of the getSubscribeNav() function.

markets/open-fund-market/contracts/oracle/NavOracle.sol:L35-L42

```
function setSubscribeNavOnlyMarket(bytes32 poolId_, uint256 time_, uint256 nav_)
        external virtual override onlyMarket
{
        uint256 dayTime = time_ / 86400 * 86400;
        poolNavInfos[poolId_].navs[dayTime] = nav_;
        poolNavInfos[poolId_].latestSetNavTime = dayTime;
        emit SetSubscribeNav(poolId_, dayTime, nav_);
}
```

markets/open-fund-market/contracts/oracle/NavOracle.sol:L54-L66

```
function getSubscribeNav(bytes32 poolId_, uint256 time_)
    external view virtual override returns (uint256 nav_, uint256 navTime_)
{
    ...
    // if nav of the day is not set, return the latest nav info
    if (nav_ == 0) {
        navTime_ = poolNavInfo.latestSetNavTime;
        nav_ = poolNavInfo.navs[navTime_];
    }
}
```

Recommendation

Consider adding a check before updating the latestSetNavTime.

Status

This issue has been resolved by the team.

9. Inappropriate solidity version requirement

Severity: Informational	Category: Configuration

Target:
- sft/abilities/contracts/value-issuable/SFTValueIssuableDelegate.sol
- sft/payable/open-fund/contracts/open-fund-shares/OpenFundShareConcrete.sol

Description

Custom error is introduced since Solidity 0.8.4. Hence the pragma should be ^0.8.4.

sft/abilities/contracts/value-issuable/SFTValueIssuableDelegate.sol:L2,L11

```
pragma solidity ^0.8.0;
...
error OnlyMarket();
```

sft/payable/open-fund/contracts/open-fund-shares/OpenFundShareConcrete.sol:L3,L9

```
pragma solidity ^0.8.0;
...
error BurnNotAllowed();
```

Recommendation

Consider making the changes suggested above.

Status

This issue has been resolved by the team.

10. Incorrect gap size

Severity: Informational	Category: Configuration
Target: - sft/abilities/contracts/fcfs-multi-repayable/FCFSMultiRepayableConcrete.sol - markets/open-fund-market/contracts/OpenFundMarketStorage.sol	

Description

In the FCFSMultRepayableConcrete contract, the number of used storage slots is 2:

- _slotRepayInfo
- allocatedCurrencyBalance

However, the gap size is 47.

In the OpenFundMarketStorage contract, the number of used storage slots is 8, while the gap size is 43.

Recommendation

Consider changing the gap size of the FCFSMultRepayableConcrete contract to 48 and the OpenFundMarketStorage contract to 42.

Status

This issue has been acknowledged by the team. The code is left unchanged to prevent storage collision during contract upgrades.

11. Missing check for the equivalence of valueDecimals

Severity: Informational	Category: Data Validation
Target: - markets/open-fund-market/contracts/OpenFundMarket.sol	

Description

OpenFundMarket._validateInputPoolInfo() can add the following check to make sure the OpenFundShare and OpenFundRedemption SFTs for a pool have the same valueDecimals.

```
require(IERC3525(inputPoolInfo.openFundShare).valueDecimals() ==
IERC3525(inputPoolInfo.openFundRedemption).valueDecimals(), "OFM: share and redemption
SFTs have different valueDecimals");
```

Recommendation

Consider making the changes suggested above.

Status

This issue has been resolved by the team.

12. Missing explicit check for poolId existence

Severity: Informational	Category: Data Validation

Target:
- markets/open-fund-market/contracts/OpenFundMarket.sol

Description

An explicit check for poolId existence could be added to the subscribe(), requestRedeem(), revokeRedeem(), closeCurrentRedeemSlot(), setSubscribeNav(), and setRedeemNav() functions in the OpenFundMarket contract so that when a user inputs an invalid poolId, the function will fail fast and throw appropriate error messages.

Recommendation

An example modification:

```
PoolInfo storage poolInfo = poolInfos[poolId_];
require(poolInfos[poolId_].poolSFTInfo.openFundShareSlot != 0, "OFM: pool does not
exist");
```

Status

This issue has been resolved by the team.

13. Missing event parameters

Severity: Informational	Category: Logging
Target: - sft/abilities/contracts/fcfs-multi-repayable/IFCFSMultiRepayableDelegate.sol	

Description

Since repayments and claims are currency dependent, it is recommended to add the currency parameter to the Repay and Claim events.

sft/abilities/contracts/fcfs-multi-repayable/IFCFSMultiRepayableDelegate.sol:L6-L7

```
event Repay(uint256 indexed slot, address indexed payer, uint256 repayCurrencyAmount);
event Claim(address indexed to, uint256 indexed tokenId, uint256 claimValue, uint256
claimCurrencyAmount);
```

Recommendation

Consider making the changes suggested above.

Status

This issue has been resolved by the team.

14. Redundant usage of boolean literal

Severity: Informational	Category: Gas Optimization

Target:
- markets/open-fund-market/contracts/OpenFundMarket.sol

Description

markets/open-fund-market/contracts/OpenFundMarket.sol:L301

```
poolInfo.permissionless = whitelist_.length == 0 ? true : false;
```

Assigning true or false to poolInfo.permissionless explicitly is not required since whitelist_.length == 0 uses `==` operator which will evaluate to boolean values.

Recommendation

Consider changing the above line to the following which doesn't explicitly use boolean literals.

```
poolInfo.permissionless = whitelist_.length == 0;
```

Status

This issue has been resolved by the team.

15. Improper error message

Severity: Informational	Category: Logging
Target: - markets/open-fund-market/contracts/OpenFundMarket.sol	

Description

The following statement is checking if the currencyAmount_ reaches the minimum subscription value. The error message "exceed subscribe min limit" is not proper.

markets/open-fund-market/contracts/OpenFundMarket.sol:L120

```
require(currencyAmount_ >= poolInfo.subscribeLimitInfo.subscribeMin, "OFM: exceed
subscribe min limit");
```

Recommendation

Consider updating to a suitable error message.

Status

This issue has been resolved by the team.

16. Redundant code

Severity: Informational	Category: Redundancy

Target:
- markets/open-fund-market/contracts/OpenFundMarket.sol

Description

The following code is commented out.

markets/open-fund-market/contracts/OpenFundMarket.sol:L162

```
// ERC3525TransferHelper.doTransfer(poolInfo.poolSFTInfo.openFundShare,
openFundShareId_, _poolRedeemTokenId[poolInfo.poolSFTInfo.latestRedeemSlot],
redeemValue_);
```

Recommendation

Consider removing the redundant code.

Status

This issue has been resolved by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit a3d1238 of repository https://github.com/solv-finance-dev/solv-contracts-v3:

File	SHA-1 hash
commons/solidity-utils/contracts/helpers/ERC3525TransferHelper.sol	b64a572a767a08148a630a959fd8a4f981a56e65
commons/solidity-utils/contracts/helpers/ERC20TransferHelper.sol	70c745743b15d484d7df2e63e58f6f7aa00a196e
sft/abilities/contracts/fcfs-multi-repayable/FCFSMultiRepayableConcrete.sol	71b286d2538845a6341fe573c83955f0aca8d6f6
sft/abilities/contracts/fcfs-multi-repayable/FCFSMultiRepayableDelegate.sol	4bec394de0259b6b1a67327c9fc552adfddb29ab
sft/abilities/contracts/value-issuable/SFTValueIssuableConcrete.sol	16ab15b4e5aea9f419a57e43eb6d9e4703ea126b
sft/abilities/contracts/value-issuable/SFTValueIssuableDelegate.sol	ff167ec1eb09bb9162d7cbfb113b26754f8ac2ec
sft/payable/open-fund/contracts/open-fund-shares/OpenFundShareDelegate.sol	aab46cffd9d62e3a5ece70f35698f0eac7eefcc8
sft/payable/open-fund/contracts/open-fund-shares/OpenFundShareConcrete.sol	71e9417dc0e5efddc43015a39bfca352cd009620
sft/payable/open-fund/contracts/open-fund-redemptions/OpenFundRedemptionDelegate.sol	35609a2349e49508a2bfddba7b46baa310af5ed9
sft/payable/open-fund/contracts/open-fund-redemptions/OpenFundRedemptionConcrete.sol	c08e5ea8f864ac304f7d77f71c7c2d0a3f4408bb
markets/open-fund-market/contracts/whitelist/OFMWhitelistStrategyManager.sol	9bad0da1bba2864213e3d539b88000a24974212d
markets/open-fund-market/contracts/OpenFundMarket.sol	f1c4c8ef075bc66ef92e5bd9a3d460639d117daf
markets/open-fund-market/contracts/OpenFundMarketStorage.sol	b8ad604530f47d86885b8afc4fdfa318393a55f7
markets/open-fund-market/contracts/OFMConstants.sol	4bcc3bdbed1747cc962ad27ddcd1dd3ac02d5d55

The files, updated post issue resolution, are located in commit 7ad12e8 of the repository: https://github.com/solv-finance/solv-contracts-v3.