SALUS SECURITY

MAR 2023

CODE SECURITY ASSESSMENT

SOLV PROTOCOL V3

Overview

Project Summary

- Name: Solv Protocol
- Version: V3
- 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 V3
Version	v2
Type	Solidity
Dates	Mar 10 2023
Logs	Mar 1 2023; Mar 10 2023

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	1
Total Low-Severity issues	3
Total informational issues	6
Total	11

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	User can subscribe for free if price * value < 10**decimals	High	Business Logic	Resolved
2	Centralization risk	Med	Centralization	Acknowledged
3	Missing storage gap in upgradeable base contracts	Low	Business Logic	Resolved
4	Upgradeable dependency not initialized	Low	Business Logic	Resolved
5	doApprove() can fail silently if the underlying parameter is an EOA account	Low	Business Logic	Resolved
6	Function visibility can be pure	Informational	Code Quality	Resolved
7	SFTConcreteControl could be declared abstract	Informational	Code Quality	Resolved
8	Incorrect error message	Informational	Auditing and Logging	Resolved
9	Use calldata instead of memory for function parameters	Informational	Gas Optimization	Resolved
10	Can use mapping(address=>bool) instead of address[] to store the whitelisted addresses for an issueKey	Informational	Gas Optimization	Resolved
11	Redundant code	Informational	Redundancy	Resolved

2.2 Notable Findings

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

1. User can subscribe for free if price * value < 10**decimals

Severity: High	Category: Business Logic

Target:
- markets/issue-market/contracts/IssueMarket.sol

Description

When a user subscribes to an SFT token from the issue market by using subscribe(), the vars.payment amount of currency is transferred from the user to the IssueMarket contract, while the corresponding SFT token is minted to the user.

markets/issue-market/contracts/IssueMarket.sol:L144

```
vars.payment = (vars.price * value_)/(10**vars.sftInfo.decimals);
```

However, if price * value < 10**decimals, the vars.payment will be 0 due to rounding to zero. In this situation, the user can subscribe for free.

Proof of Concept

Here, we can do a PoC based on the existing test file
solv-contracts-v3/markets/issue-market/test/index.ts

First, we add some print statements to the test. We change the code from
solv-contracts-v3/markets/issue-market/test/index.ts:L115-L119

```
it("subscribe should be successful", async() => {
    await currency.approve(market.address, payment);
    const expireTime = await current() + 1000;
    await market.subscribe(sft.address, slot, underwriterName, subscribeValue,
expireTime);
});
```

to

```
it("subscribe should be successful", async () => {
  await currency.approve(market.address, payment);
  const expireTime = (await current()) + 1000;
  console.log(
    "currency balance before subscribe:",
    (await currency.balanceOf(deployer.address)).toNumber()
  );
  console.log(
    "sft balance before subscribe",
    (await sft["balanceOf(address)"](deployer.address)).toNumber()
  );
```

```
  await market.subscribe(sft.address, slot, underwriterName, subscribeValue,
expireTime);
  console.log(
    "currency balance after subscribe:",
    (await currency.balanceOf(deployer.address)).toNumber()
  );
  console.log(
    "sft balance after subscribe",
    (await sft["balanceOf(address)"](deployer.address)).toNumber()
  );
});
```

When we run the test, the printed message is:

```
currency balance before subscribe: 100000000
sft balance before subscribe 0
currency balance after subscribe: 99999990
sft balance after subscribe 1
```

This means that the buyer has subscribed one sft token at a cost of 10 (100000000 - 99999990) amount of currency.

Then, we change the subscribeValue variable from
solv-contracts-v3/markets/issue-market/test/index.ts:L19

```
const subscribeValue = 100000000000;
```

to

```
const subscribeValue = 100000000;
```

So that price * subscribeValue < 10**decimals, and subscribeValue >= min.

When we rerun the test again, the printed message is:

```
currency balance before subscribe: 100000000
sft balance before subscribe 0
currency balance after subscribe: 100000000
sft balance after subscribe 1
```

This means that the buyer has subscribed to an SFT token for free!

Recommendation

Consider adding a `require(vars.payment > 0);` check to avoid free subscribing.

Status

This issue has been resolved by the team in commit f0b5483. The team has added the following check:

```
require(vars.price == 0 || vars.payment > 0, "IssueMarket: payment must be greater than
0");
```

2. Centralization risk

Severity: Medium	Category: Centralization
Target: - all	

Description

The upgradeable proxy pattern is used throughout the Solv Protocol V3 codebase. The proxy admin controls the upgrade mechanism to upgradeable proxies, they can change the respective implementations. Should the admin's private key be compromised, an attacker could upgrade the logic contract to execute their own malicious logic on the proxy state.

There is also a privileged Owner role in the IssueMarket contract, the Owner of the IssueMarket contract:

- can withdraw fees from the contract to any address
- can add or remove currencies from the whitelist
- can add or remove SFT tokens
- can add underwriters and add currencies for underwriters

Should the Owner's private key be compromised, an attacker could drain all the funds from the contract.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the users.

Recommendation

Consider transferring the privileged roles to multi-sig accounts.

Status

This issue has been acknowledged by the team. The team has promised to transfer the Owner of IssueMarket to an address with a timelock or voting mechanism in the future, and to transfer the ProxyAdmin of IssueMarket and the ProxyAdmin of Payable to multisig addresses after contract deployment.

3. Missing storage gap in upgradeable base contracts

Severity: Low	Category: Business Logic

Target:
- common/solidity-utils/contracts/access/SFTConcreteControl.sol
- common/solidity-utils/contracts/access/SFTDelegateControl.so
- common/solidity-utils/contracts/access/AdminControl.sol
- common/solidity-utils/contracts/access/OwnControl.sol
- sft/core/contracts/BaseSFTDelegateUpgradeable.sol
- sft/core/contracts/BaseSFTConcreteUpgradeable.sol
- sft/abilities/contracts/issuable/SFTIssuableConcrete.sol
- sft/abilities/contracts/issuable/SFTIssuableDelegate.sol
- sft/abilities/contracts/multi-rechargeable/MultiRechargeableConcrete.sol
- sft/abilities/contracts/multi-rechargeable/MultiRechargeableDelegate.sol
- sft/abilities/contracts/multi-repayable/MultiRepayableConcrete.sol
- sft/abilities/contracts/multi-repayable/MultiRepayableDelegate.sol
- sft/abilities/contracts/mintable/SFTMintableConcrete.sol
- sft/abilities/contracts/mintable/SFTMintableDelegate.sol

Description

When using the proxy pattern for upgrades, it is common practice to include storage gaps in parent contracts to reserve space for potential future variables.

According to the OpenZeppelin document, adding a storage gap to the upgradeable base contract would allow the developers to freely add new state variables in the future without compromising the storage compatibility with existing deployments.

However, the upgradeable base contracts in Solv Protocol V3 lack the storage gaps. Without the storage gap, the variable in the child contract might be overwritten by the upgraded base contract if new variables are added to the base contract. This could have unintended and serious consequences for the child contracts, potentially resulting in the loss of user funds or complete failure of the contract.

Recommendation

Consider adding appropriate storage gaps at the end of upgradeable base contracts. Please refer to the OpenZeppelin upgradeable contract document.

Status

This issue has been resolved by the team in commit f0b5483.

4. Upgradeable dependency not initialized

Severity: Low	Category: Business Logic

Target:
- markets/issue-market/contracts/IssueMarket.sol
- sft/core/contracts/BaseSFTDelegateUpgradeable.sol

Description

Upgradeable dependencies should be initialized.

sft/core/contracts/BaseSFTDelegateUpgradeable.sol:L14-L15

```
abstract contract BaseSFTDelegateUpgradeable is IBaseSFTDelegate,
ERC3525SlotEnumerableUpgradeable,
        OwnControl, SFTDelegateControl, ReentrancyGuardUpgradeable {
```

markets/issue-market/contracts/IssueMarket.sol:L22

```
contract IssueMarket is IIssueMarket, IssueMarketStorage, ReentrancyGuardUpgradeable,
ResolverCache {
```

The ReentryancyGuardUpgradeable contract is inherited by

- BaseSFTDelegateUpgradeable
- IssueMarket

However, the ReentryancyGuardUpgradeable contract is not initialized in these contracts.

Although this is not a major issue since an uninitialized _status variable in the ReentryancyGuardUpgradeable contract does not affect the operation of the nonReentrant() modifier. However, it is recommended that you initialize all the upgradeable dependencies, including ReentryancyGuardUpgradeable, to improve code consistency.

Recommendation

Consider initializing the ReentryancyGuardUpgradeable contract by using __ReentrancyGuard_init() function in the initialization logic of the BaseSFTDelegateUpgradeable contract and the IssueMarket contract.

Status

This issue has been resolved by the team in commit f0b5483.

5. doApprove() can fail silently if the underlying parameter is an EOA account

Severity: Low	Category: Business Logic

Target:
- commons/solidity-utils/contracts/helpers/ERC20TransferHelper.sol

Description

commons/solidity-utils/contracts/helpers/ERC20TransferHelper.sol:L24-L41

```
function doApprove(
    address underlying,
    address spender,
    uint256 amount) internal {
        require(underlying != Constants.ETH_ADDRESS
            && underlying != Constants.ZERO_ADDRESS, "invalid underlying");
         (bool success, bytes memory data) = underlying.call(
            abi.encodeWithSelector(
                ERC20Interface.approve.selector,
                spender,
                amount
            )
        );
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "SAF"
        );
}
```

The doApprove() function in the ERC20TransferHelper library uses a low-level function call to do the ERC-20 approval.

According to the Solidity docs:
“The low-level functions call, delegatecall and staticcall return true as their first return value if the account called is non-existent, as part of the design of the EVM. Account existence must be checked prior to calling if needed.”

Therefore, if **underlying** is an EOA account, the low-level call, **underlying.call(),** will return true and empty bytes, which will bypass the subsequent require() statement. The doApprove() function will fail silently.

In terms of the Solv Protocol V3 codebase, the doApprove() function is only used by the IssueMarket.subscribe() function. Since the underlying currency in the IssueMarket contract can only be set by the Owner role, it is highly unlikely that an EOA address will be passed as the underlying parameter to the doApprove() function, so the above issue is unlikely to occur.

Nevertheless, the potential risk should be documented in the doApprove() docstring in case the doApprove() function is misused in future development. Better yet, a check for the account's existence of the underlying parameter can be added in the doApprove() function.

Recommendation

Consider checking whether the underlying parameter is an EOA address in the doApprove() function.

Status

This issue has been resolved by the team in commit f0b5483.

2.3 Informational Findings

6. Function visibility can be pure

Severity: Informational	Category: Code Quality

Target:
- markets/issue-market/contracts/price/FixedPriceStrategy.sol

Description

markets/issue-market/contracts/price/FixedPriceStrategy.sol:L9

```
function getPrice(bytes memory priceInfo_) public view returns (uint256)
```

markets/issue-market/contracts/price/FixedPriceStrategy.sol:L14

```
function checkPrice(bytes memory priceInfo_) public view returns (bool)
```

The getPrice() and checkPrice() functions do not change or access the state variables within the contract, the function visibility of them could be changed from view to pure.

The pure functions do not read or modify the state variables, which return the values only using the parameters passed to the function or local variables present in it.

Recommendation

Consider changing the function visibility of the getPrice() and checkPrice() functions to pure.

Status

This issue has been resolved by the team in commit f0b5483.

7. SFTConcreteControl could be declared abstract

Severity: Informational	Category: Code Quality

Target:
- common/solidity-utils/contracts/access/SFTConcreteControl.sol
- sft/core/contracts/BaseSFTConcreteUpgradeable.sol

Description

commons/solidity-utils/contracts/access/SFTDelegateControl.sol:L7

```
abstract contract SFTDelegateControl is ISFTDelegateControl, AdminControl {
```

common/solidity-utils/contracts/access/SFTConcreteControl.sol:L8

```
contract SFTConcreteControl is ISFTConcreteControl, AdminControl {
```

The SFTDelegateControl contract is declared abstract, while the corresponding SFTConcreteControl contract is not.

sft/core/contracts/BaseSFTConcreteUpgradeable.sol:L10

```
abstract contract BaseSFTConcreteUpgradeable is IBaseSFTConcrete, SFTConcreteControl {
```

In addition, the BaseSFTConcreteUpgradeable contract, which inherits from the SFTConcreteControl contract, is also declared abstract.

To improve code consistency, the SFTConcreteControl can be declared abstract.

Recommendation

Consider declaring the SFTConcreteControl contract as an abstract class.

Status

This issue has been resolved by the team in commit f0b5483.

8. Incorrect error messages

Severity: Informational	Category: Auditing and Logging

Target:
- markets/issue-market/contracts/IssueMarket.sol

Description

markets/issue-market/contracts/IssueMarket.sol:L170

```
require(input_.min <= input_.max, "IssueMarket: min must be less than max");
```

markets/issue-market/contracts/IssueMarket.sol:L172

```
require(input_.max <= input_.issueQuota, "IssueMarket: max must be less than
totalIssuance");
```

The above the error messages do not match the check, the check uses <=, while the error message indicates < (less than).

Recommendation

Consider changing the description from “be less than” to “be no greater than” in the error message.

Status

This issue has been resolved by the team in commit f0b5483.

9. Use calldata instead of memory for function parameters

Severity: Informational	Category: Gas Optimization

Target:
- markets/issue-market/contracts/whitelist/WhitelistStrategyManager.sol
- markets/issue-market/contracts/whitelist/IWhitelistStrategyManager.sol
- markets/issue-market/contracts/IssueMarket.sol
- sft/abilities/contracts/issuable/SFTIssuableConcrete.sol
- sft/abilities/contracts/issuable/SFTIssuableDelegate.sol
- sft/abilities/contracts/mintable/SFTMintableConcrete.sol
- sft/abilities/contracts/mintable/SFTMintableDelegate.sol
- sft/future/underwriter-profit/contracts/UnderwriterProfitConcrete.sol

Description

A common gas-saving practice is to use calldata instead of memory when the function argument is read only.

1. markets/issue-market/contracts/whitelist/WhitelistStrategyManager.sol:L19

```
function setWhitelist(bytes32 issueKey_, address[] memory whitelist_) external virtual
override {
```

2. markets/issue-market/contracts/whitelist/IWhitelistStrategyManager.sol:L6

```
function setWhitelist(bytes32 issueKey_, address[] memory whitelist_) external;
```

3. markets/issue-market/contracts/IssueMarket.sol:L36

```
function issue(address sft_, address currency_, bytes memory inputSlotInfo_, bytes
memory inputIssueInfo_,
        bytes memory inputPriceInfo_) external payable override nonReentrant returns
(uint256 slot_) {
```

4. markets/issue-market/contracts/IssueMarket.sol:L229

```
function addUnderwriterOnlyOwner(string memory underwriter_, uint16 defaultFeeRate_,
address[] calldata currencies_) public onlyOwner {
```

5. markets/issue-market/contracts/IssueMarket.sol:L239

```
function addUnderwriterCurrenciesOnlyOwner(string memory underwriter_, address[]
calldata currencies_) public onlyOwner {
```

6. sft/abilities/contracts/issuable/SFTIssuableConcrete.sol:L17

```
function createSlotOnlyDelegate(address txSender_, bytes memory inputSlotInfo_) public
virtual override onlyDelegate returns (uint256 slot_)  {
```

7. sft/abilities/contracts/issuable/SFTIssuableDelegate.sol:L20

```
function createSlotOnlyIssueMarket(address txSender_, bytes memory inputSlotInfo_)
public virtual override nonReentrant returns(uint256 slot_) {
```

8. sft/abilities/contracts/mintable/SFTMintableConcrete.sol:L14

```
function createSlotOnlyDelegate(address txSender_, bytes memory inputSlotInfo_) external
virtual override onlyDelegate returns (uint256 slot_) {
```

9. sft/abilities/contracts/mintable/SFTMintableDelegate.sol:L20

```
function createSlot(bytes memory inputSlotInfo_) public virtual override nonReentrant
returns (uint256 slot_) {
```

For the above functions, having function arguments in calldata instead of memory is more optimal.

Recommendation

Consider using calldata instead of memory for read only function parameters to save gas.

Status

This issue has been resolved by the team in commit f0b5483.

10. Can use mapping(address=>bool) instead of address[] to store the whitelisted addresses for an issueKey

Severity: Informational	Category: Gas Optimization
Target: - markets/issue-market/contracts/whitelist/WhitelistStrategyManager.sol	

Description

markets/issue-market/contracts/whitelist/WhitelistStrategyManager.sol:L12-L32

```
mapping(bytes32 => address[]) private _whitelists;

...

function setWhitelist(bytes32 issueKey_, address[] memory whitelist_) external virtual
override {
    require(_msgSender() == _issueMarket(), "only issue market");
    _whitelists[issueKey_] = whitelist_;
}

function isWhitelisted(bytes32 issueKey_, address buyer_) external view virtual override
returns (bool) {
    address[] memory whitelist = _whitelists[issueKey_];
    for (uint256 i = 0; i < whitelist.length; i++) {
        if (whitelist[i] == buyer_) {
            return true;
        }
    }
    return false;
}

...
```

In the **WhitelistStrategyManager** contract, all the whitelisted addresses for an issueKey are stored in _whitelists[issueKey], which is an address[].

The _whitelists[issueKey] is only used in the setWhitelist() function and the isWhitelisted() functions.

Therefore, the address[] structure can be replaced with a mapping(address=>bool) structure. For example, we can define _whitelists as

```
mapping(bytes32 => mapping(address => bool)) _whitelists
```

And

- use `_whitelists[issueKey][buyer] = true` to add a buyer to the whitelist for issueKey;
- use `_whitelists[issueKey][buyer]` to indicate whether the buyer's address is whitelisted for the issue key.

Since the isWhitelisted() function iterates over the address array to check whether a buyer address is whitelisted, using a mapping() instead of an array can save gas for the isWhitelisted() function.

Recommendation

Consider declaring _whitelists as a nested mapping instead of a mapping to an address array.

Status

This issue has been resolved by the team by declaring _whitelists as a nested mapping in commit f0b5483.

11. Redundant code

Severity: Informational	Category: Redundancy

Target:
- markets/issue-market/contracts/IssueMarket.sol
- markets/issue-market/contracts/IssueMarket.sol
- markets/issue-market/contracts/IssueMarket.sol
- markets/issue-market/contracts/IssueMarketStorage.sol
- markets/issue-market/contracts/price/PriceStrategyManager.sol
- commons/address-resolver/contracts/ResolverCache.sol
- commons/solidity-utils/contracts/helpers/ERC20TransferHelper.sol
- commons/solidity-utils/contracts/helpers/ERC20TransferHelper.sol
- sft/future/payable/contracts/PayableConcrete.sol

Description

1. markets/issue-market/contracts/IssueMarket.sol:L71-L83

```
UnderwriterIssueInfo[] memory underwriterIssues = new
UnderwriterIssueInfo[](vars.inputIssueInfo.underwriters.length);
for (uint256 i = 0; i < vars.inputIssueInfo.underwriters.length; i++) {
   string memory underwriterName = vars.inputIssueInfo.underwriters[i];
   vars.underwriterKey = _getUnderwriterKey(underwriterName);
   require(underwriterInfos[vars.underwriterKey].isValid, "IssueMarket: underwriters not
supported");
   UnderwriterIssueInfo memory underwriterIssueInfo = UnderwriterIssueInfo({
            name: underwriterName,
            quota: vars.inputIssueInfo.quotas[i],
            value: vars.inputIssueInfo.quotas[i]
        });
   underwriterIssueInfos[vars.underwriterKey][vars.issueKey] = underwriterIssueInfo;
   underwriterIssues[i] = underwriterIssueInfo;
}
```

The underwriterIssues variable is redundant; thus the highlighted lines can be removed.

2. markets/issue-market/contracts/IssueMarket.sol:L12,L15

```
import
"@solvprotocol/contracts-v3-solidity-utils/contracts/helpers/ERC3525TransferHelper.sol";
...
import "./IIssueMarketStorage.sol";
```

ERC3525TransferHelper and IIssueMarketStorage are imported but not used, so they can be removed.

3. markets/issue-market/contracts/IssueMarket.sol:L20

```
import "hardhat/console.sol";
```

console.sol is an unused import and can be removed.

4. markets/issue-market/contracts/IssueMarketStorage.sol:L34

```
mapping(bytes32 => EnumerableSet.Bytes32Set) issueKeys;
```

The state variable issueKeys is declared but never used; thus, it can be removed.

5. markets/issue-market/contracts/price/PriceStrategyManager.sol:L12-L14

```
struct FixedPrice {
    uint256 price;
}
```

The struct FixedPrice is defined but not used in the PriceStrategyManager contract.

6. commons/address-resolver/contracts/ResolverCache.sol:L5

```
import "hardhat/console.sol";
```

console.sol is an unused import and can be removed.

7. commons/solidity-utils/contracts/helpers/ERC20TransferHelper.sol:L117-L119

```
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b <= a, "SafeMath: subtraction overflow");
    return a - b;
}
```

Overflow is checked by default since Solidity v0.8.0, therefore, the require() check in the sub() function is unnecessary.

8. commons/solidity-utils/contracts/helpers/ERC20TransferHelper.sol:L43-L82

```
function doTransferIn(
    address underlying,
    address from,
    uint256 amount
) internal returns (uint256) {
    if (underlying == Constants.ETH_ADDRESS) {
        // Sanity checks
        require(tx.origin == from || msg.sender == from, "sender mismatch");
        require(msg.value >= amount, "value mismatch");

        return amount;
    } else {
        require(msg.value == 0, "don't support msg.value");
        uint256 balanceBefore = ERC20Interface(underlying).balanceOf(
            address(this)
        );
        (bool success, bytes memory data) = underlying.call(
            abi.encodeWithSelector(
                ERC20Interface.transferFrom.selector,
                from,
                address(this),
                amount
            )
        );
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "STF"
        );

        // Calculate the amount that was *actually* transferred
        uint256 balanceAfter = ERC20Interface(underlying).balanceOf(
            address(this)
        );
        require(
            balanceAfter >= balanceBefore,
            "TOKEN_TRANSFER_IN_OVERFLOW"
        );
        return balanceAfter - balanceBefore; // underflow already checked above, just
```

```
subtract
    }
}
```

Overflow is checked by default since Solidity v0.8.0, therefore, the overflow check in the doTransferIn() function is unnecessary and the highlighted lines can be removed.

9. sft/future/payable/contracts/PayableConcrete.sol:L12

```
import "hardhat/console.sol";
```

console.sol is an unused import and can be removed.

Recommendation

Consider removing the redundant code.

Status

This issue has been resolved by the team in commit f0b5483.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in https://github.com/solv-finance-dev/solv-contracts-v3 (Commit a8b61eb):

File	SHA-1 hash
commons/address-resolver/contracts/AddressResolver.sol	98913117604d626d482f9bb31f7cae3b1a1d233f
commons/address-resolver/contracts/ResolverCache.sol	dae69a2f641a645754e0c6db213faff485bfaedd
commons/solidity-utils/contracts/access/AdminControl.sol	57220b4fd53a60f552f61c9253b17881270b0319
commons/solidity-utils/contracts/access/OwnControl.sol	157634e86885d6db06ca49fac8d0bbb595ac47fa
commons/solidity-utils/contracts/access/SFTConcreteControl.sol	23730955f4488c25f2ab2aa6f81fea8290eea51c
commons/solidity-utils/contracts/access/SFTDelegateControl.sol	7e28e7156250f3e487b75be7c2707cfc8979a87b
commons/solidity-utils/contracts/helpers/ERC20TransferHelper.sol	4e5425914eba9e36697bde718b87728d19fe574f
commons/solidity-utils/contracts/helpers/ERC3525TransferHelper.sol	4ee02f76c63604a64fd1560803e1e9fb38418c52
markets/issue-market/contracts/whitelist/WhitelistStrategyManager.sol	daf1a6e29a72c3ae016e556171309285eb14d9f4
markets/issue-market/contracts/IssueMarket.sol	3cb63ce8222eb209122e135ef95c36abd27c8a3d
markets/issue-market/contracts/price/FixedPriceStrategy.sol	2a9e960c78eb5b0d33258aa325dd7c5668ace708
markets/issue-market/contracts/IssueMarketStorage.sol	f1991a3706ce4b61077be06fb2167651202316c2
sft/core/contracts/BaseSFTDelegateUpgradeable.sol	31e2b0d75b4f6122b1043240c0d83b7d997aaebf

sft/core/contracts/BaseSFTConcreteUpgradeable.sol	b0f8ba6bf96b935d6f9861c88d74cf3dc7bdc05e
sft/abilities/contracts/issuable/SFTIssuableConcrete.sol	80ab4a2d947254ed962b31c371589c2e02bf6f64
sft/abilities/contracts/issuable/SFTIssuableDelegate.sol	b2e97c2012894e338bfb593cfee1a31214115625
sft/abilities/contracts/issuable/MultiRechargeableConcrete.sol	c2c0f850f79d8487b70953b663b614540e91c2a9
sft/abilities/contracts/multi-rechargeable/MultiRechargeableDelegate.sol	5d8dee3d2cdf4b45e9328f0f0c170e564ca06c53
sft/abilities/contracts/multi-rechargeable/MultiRepayableConcrete.sol	48ea5b35d55e70d5fcab040e8fb138161767bbc1
sft/abilities/contracts/multi-rechargeable/MultiRepayableDelegate.sol	d9977a7975b4f0d3e68acb505dd5cf2c5110c6a4
sft/abilities/contracts/mintable/SFTMintableConcrete.sol	cd759eab8138f7229161b2c3f241834e8df32af7
sft/abilities/contracts/mintable/SFTMintableDelegate.sol	c9fb1e1a99e24d34762b3a23b72b816d4cd1fa51
sft/future/payable/contracts/PayableDelegate.sol	060626e21eb21530fe443d23d21a5d86ccbb6b3b
sft/future/payable/contracts/PayableConcrete.sol	fe31fd310182e0547cf896e7944ad9fa842d9d9f
sft/future/payable/contracts/factories/FactoryCore.sol	172f44d9fd28626eb83afb0fa68e5cab00e15c67
sft/future/payable/contracts/factories/PayableConcreteFactory.sol	1dfaf8f080c6f61ab30d340957409a7a463db344
sft/future/payable/contracts/factories/PayableDelegateFactory.sol	bc00bae71371b8b1536a62f8f594ddf7306b03b1
sft/future/underwriter-profit/contracts/UnderwriterProfitConcrete.sol	0ccff7558420c81c5e3cc1228f4e86210eadaba5
sft/future/underwriter-profit/contracts/UnderwriterProfitDelegate.sol	7aa9bf62f9617d275fe15a38332071bc5e07bad4

Appendix 2 - Git repository changed

The original Git repository and the commit ID of reviewed files is:

https://github.com/solv-finance-dev/solv-contracts-v3 (Commit a8b61eb)

After the team fixed all the issues, they decided to move the code to a public Git repository. The following is the Git repository and the commit ID after all fixes of the issues:

https://github.com/solv-finance/solv-contracts-v3 (Commit 3f3ce5c)

During the process, the team has renamed some paths and contracts:

Before	**After**
markets/issue-market	markets/prime
sft/future	sft/payable
sft/future/payable	sft/payable/earn
sft/future/payable/IPayableConcrete.sol	sft/payable/earn/IEarnConcrete.sol
sft/future/payable/PayableConcrete.sol	sft/payable/earn/EarnConcrete.sol
sft/future/payable/IPayableDelegate.sol	sft/payable/earn/IEarnDelegate.sol
sft/future/payable/PayableDelegate.sol	sft/payable/earn/EarnDelegate.sol
sft/future/payable/factories/PayableDelegateFactory.sol	sft/payable/earn/factories/EarnDelegateFactory.sol
sft/future/payable/factories/PayableConcreteFactory.sol	sft/payable/earn/factories/EarnConcreteFactory.sol