SALUS SECURITY

JAN 2024

CODE SECURITY ASSESSMENT

SOLV PROTOCOL

Overview

Project Summary

- Name: Solv Protocol - stUSD
- Platform: EVM-compatible Chains
- Language: Solidity
- Repository: https://github.com/solv-finance/stUSD
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	Solv Protocol - stUSD
Version	v3
Type	Solidity
Dates	Jan 01 2024
Logs	Dec 14 2023; Dec 19 2023; Jan 01 2024

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	0
Total Low-Severity issues	1
Total informational issues	1
Total	3

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	DOS attack via deployment back-run	High	Denial of Service	Resolved
2	Incorrect deployment of upgradeable contract	Low	Upgradeability	Resolved
3	sftWrappedTokens data might be overridden	Informational	Business Logic	Resolved

2.2 Notable Findings

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

1. DOS attack via deployment back-run	
Severity: High	Category: Denial of Service
Target: - contracts/SftWrappedToken.sol	

Description

Attackers can perform a denial-of-service (DOS) attack on the SftWrappedToken contract by sending SFT value to the contract right after its deployment.

contracts/SftWrappedToken.sol:L68-L81

```
function mint(uint256 sftId_, uint256 amount_) external virtual override nonReentrant {
    ...
    if (IERC3525(wrappedSftAddress).balanceOf(address(this)) == 0) {
        holdingSftId = ERC3525TransferHelper.doTransferIn(wrappedSftAddress, sftId_,
amount_);
    } else {
        ERC3525TransferHelper.doTransfer(wrappedSftAddress, sftId_, holdingSftId,
amount_);
    }
}
```

The *holdingSftId* state variable in the SftWrappedToken contract is the tokenId that holds value from wrapped SFT. It is initialized during the first mint() call when the contract doesn't hold any wrapped SFT. However, if the contract already has wrapped SFTs before the first mint() call, the *holdingSftId* variable would be uninitialized and remain 0, resulting in the mint() function cannot be used.
Attackers who have SFT value can DoS attack the SftWrappedToken contract by transferring some SFT value to the contract immediately after it is deployed. This would prevent the *holdingSftId* variable from being initialized.

Recommendation

It is recommended to set the correct initialization condition for holdingSftId.
For example, can change to if condition to the following.

```
if (holdingSftId == address(0)) {
    holdingSftId = ERC3525TransferHelper.doTransferIn(wrappedSftAddress, sftId_,
amount_);
} else ...
```

Status

The team has resolved this issue in commit 77f7593.

2. Incorrect deployment of upgradeable contract	
Severity: Low	Category: Upgradeability
Target: - contracts/SftWrappedTokenFactory.sol	

Description

Based on the deployment script, SftWrappedTokenFactory's logic contract is initialized and used as the user-facing contract.

As a result, the SftWrappedTokenFactory contract cannot be upgraded by anyone. If the project team wants to upgrade the SftWrappedTokenFactory's logic, they need to deploy and use a new contract to replace the existing one.

Recommendation

It is recommended to follow the Proxy Upgrade Pattern when using upgradeable contracts.

Alternatively, if the SftWrappedTokenFactory does not need to be upgradeable, consider using the constructor() to replace the initialize() function and using OpenZeppelin's Role-Based Access Control system to replace the upgradeable AdminControl and GovernorControl contracts.

Status

The team has resolved this issue in commit 77f7593.

2.3 Informational Findings

3. sftWrappedTokens data might be overridden	
Severity: Informational	Category: Business Logic
Target: - contracts/SftWrappedTokenFactory.sol	

Description

contracts/SftWrappedTokenFactory.sol:L82-112

```
function deployProductProxy(
    string memory productType_, string memory productName_,
    string memory tokenName_, string memory tokenSymbol_,
    address wrappedSft_, uint256 wrappedSftSlot_, address navOracle_
) external virtual onlyGovernor returns (address proxy_) {
    ...
    sftWrappedTokens[wrappedSft_][wrappedSftSlot_] = proxy_;
    ...
}
```

When the Governor deploys a wrapped token using the deployProductProxy() function, it writes the token's address to sftWrappedTokens[wrappedSft_][wrappedSftSlot_].

If the Governor then deploys another wrapped token with the same wrappedSft_ address and wrappedSftSlot_ value, the data in sftWrappedTokens[wrappedSft_][wrappedSftSlot_] would be overwritten.

Recommendation

Consider checking whether the wrapped token is already deployed in the deployProductProxy() function.

For example:

```
require(sftWrappedTokens[wrappedSft_][wrappedSftSlot_] == address(0));
```

Status

The team has resolved this issue in commit 77f7593.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 03d1ee2:

File	**SHA-1 hash**
SftWrappedToken.sol	e88551cb6b7fda8ac0a9b5c9e9cd8ee75994731f
SftWrappedTokenFactory.sol	42dea23a3e8c9819bc67ac959da54910fc8c092a
AdminControl.sol	8ac6c0c03cc61f63fee493c41cd22e959f02c4fc
GovernorControl.sol	04ee21e6f0b90fdd0527c1291d525ae3ef72944f
ERC3525TransferHelper.sol	b64a572a767a08148a630a959fd8a4f981a56e65

The second round of audit covered the following files in commit 7f71bbb:

File	**SHA-1 hash**
SftWrappedToken.sol	e5e0f0f85d5b95102ad50defb04b7af84af13268
SftWrappedTokenFactory.sol	aef780acd16339859122368c00d1a25358ce0759
AdminControl.sol	5ff9dcded924f1951e8a29f4ac8fe51f2d0b1104
GovernorControl.sol	798e30bdbda12d66030a69b1ffb2c5d319489ea9
ERC3525TransferHelper.sol	26b928d8c861ad51951f83140c0f0cd9d69fdeb8