SALUS SECURITY

JULY 2023

CODE SECURITY ASSESSMENT

SOLV PROTOCOL

Overview

Project Summary

- Name: Solv Protocol - Time-locked ERC20 Container
- Version: commit c9c0576
- Platform: EVM-compatible Chains
- Language: Solidity
- Repository: https://github.com/solv-finance/solv-contracts-v3
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	Solv Protocol - Time-locked ERC20 Container
Version	v2
Type	Solidity
Dates	July 5 2023
Logs	May 22 2023; July 5 2023

Vulnerability Summary

Total High-Severity issues	3
Total Medium-Severity issues	0
Total Low-Severity issues	1
Total informational issues	2
Total	6

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	Arguments being passed in the wrong order can lead to malfunctioning	High	Business logic	Resolved
2	Users are charged twice when they mint	High	Business logic	Resolved
3	Incorrect decimals converting logic	Hign	Business logic	Resolved
4	doTransferIn() can fail silently if the underlying parameter is an EOA account	Low	Data validation	Resolved
5	Implementation contracts initialization allowed	Informational	Configuration	Resolved
6	Redundant TimelockType	Informational	Redundancy	Resolved

2.2 Notable Findings

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

1. Arguments being passed in the wrong order can lead to malfunctioning

Severity: High	Category: Business logic

Target:
- sft/abilities/contracts/time-locked-erc20/TimelockedERC20Delegate.sol
- sft/payable/time-locked-erc20-container/contracts/TimelockedERC20ContainerDelegate.sol

Description

We have identified two places where arguments are passed in the wrong order:

1. sft/abilities/contracts/time-locked-erc20/TimelockedERC20Delegate.sol:L66

```
function _afterMint(address mintTo_, uint256 tokenId_, uint256 slot_, uint256 value_)
internal virtual {}
```

The TimelockedERC20Delegate contract defines the _afterMint() parameters in the order of mintTo_, tokenId_, slot_, and value_.

sft/payable/time-locked-erc20-container/contracts/TimelockedERC20ContainerDelegate.sol:L22-L25

```
function _afterMint(address /* mintTo_ */, uint256 slot_, uint256 /* tokenId_ */,
uint256 value_) internal virtual override {
    address erc20 = ITimelockedERC20Concrete(concrete()).erc20(slot_);
    ERC20TransferHelper.doTransferIn(erc20, _msgSender(), value_);
}
```

However, when the TimelockedERC20ContainerDelegate overrides the _afterMint() function, the tokenId_ and slot_ parameters are in the wrong order, resulting in a malfunctioning for the _afterMint() call in the mint() function.

2. sft/abilities/contracts/time-locked-erc20/TimelockedERC20Delegate.sol:L79

```
function _timelock_doTransferOut(address erc20_, address to_, uint256 amount_) internal
virtual;
```

In the TimelockedERC20Delegate contract, the _timelock_doTransferOut() function takes the parameters erc20_, to_, amount_ in that order.

sft/abilities/contracts/time-locked-erc20/TimelockedERC20Delegate.sol:L44

```
_timelock_doTransferOut(_msgSender(), erc20_, tokenOutAmount);
```

However, in the claim() function, the erc20_ and to_ arguments passed to _timelock_doTransferOut() are in the wrong order, leading to a malfunctioning for claiming.

Recommendation

Ensure that the passed-in arguments are in the correct order.

Status

This issue has been resolved by the team in commit e8a4e37.

2. Users are charged twice when they mint

Severity: High	Category: Business logic

Target:
- sft/abilities/contracts/time-locked-erc20/TimelockedERC20Delegate.sol
- sft/payable/time-locked-erc20-container/contracts/TimelockedERC20ContainerDelegate.sol

Description

sft/abilities/contracts/time-locked-erc20/TimelockedERC20Delegate.sol:L20-L32

```
function mint(address mintTo_, address erc20_, bytes calldata inputSlotInfo_, uint256
tokenInAmount_)
    external payable virtual override nonReentrant returns (uint256 slot_, uint256
tokenId_)
{
    slot_ = _createSlotIfNotExist(erc20_, inputSlotInfo_);

    _timelock_doTransferIn(erc20_, _msgSender(), tokenInAmount_);
    uint256 mintValue = tokenInAmount_ * valueDecimals() / ERC20(erc20_).decimals();
    _beforeMint(mintTo_, slot_, mintValue);
    tokenId_ = _mint(mintTo_, slot_, mintValue);
    ITimelockedERC20Concrete(concrete()).mintOnlyDelegate(_msgSender(), mintTo_, slot_,
tokenId_, mintValue);
    _afterMint(mintTo_, tokenId_, slot_, mintValue);
    emit MintValue(slot_, tokenId_, mintValue);
}
```

When users mint Time-locked ERC20 containers, they are charged twice when there should only be one charge.

This is because the _timelock_doTransferIn() and _afterMint() functions in the TimelockedERC20ContainerDelegate contract both transfer funds from the msg.sender to the contract, causing users to pay more than necessary.

Recommendation

We recommend removing the _afterMint() function in the TimelockedERC20ContainerDelegate contract.

Status

This issue has been resolved by the team in commit e8a4e37.

3. Incorrect decimals converting logic

Severity: High	Category: Business Logic
Target: - sft/abilities/contracts/time-locked-erc20/TimelockedERC20Delegate.sol	

Description

sft/abilities/contracts/time-locked-erc20/TimelockedERC20Delegate.sol:L26

```
uint256 mintValue = tokenInAmount_ * valueDecimals() / ERC20(erc20_).decimals();
```

In the mint() function, the tokenInAmount is normalized to the ERC3525's valueDecimals by multiplying it with (valueDecimals / ERC20's decimals), but it should be multiplied by (10 ** valueDecimals / 10 ** ERC20's decimals).

sft/abilities/contracts/time-locked-erc20/TimelockedERC20Delegate.sol:L43

```
uint256 tokenOutAmount = claimValue_ * ERC20(erc20_).decimals() / valueDecimals();
```

The same issue also applies to the decimals converting logic in the claim() function.

Recommendation

We recommended converting the value by multiplying it with the ratio of 10 to the power of decimals, instead of the ratio of the decimals.

Status

This issue has been resolved by the team in commit e8a4e37.

4. doTransferIn() can fail silently if the underlying parameter is an EOA account

Severity: Low	Category: Data validation
Target: - commons/solidity-utils/contracts/helpers/ERC20TransferHelper.sol	

Description

The doTransferIn() function in the ERC20TransferHelper library uses a low-level function call to perform an ERC-20 token transfer. However, it does not check if the passed-in **underlying** argument is an EOA account in the branch where underlying != Constants.ETH_ADDRESS.

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.”

If **underlying** is an EOA account, the low-level call, **underlying.call()**, will return true, which will bypass the subsequent require() statement and continue executing the subsequent logic in the caller function when there is no ERC-20 transfer happening.

Recommendation

Consider checking whether underlying is an EOA address in the doTransferIn() function, similar to what’s done in the doTransferOut() function.

Status

This issue has been resolved by the team in commit e8a4e37.

2.3 Informational Findings

5. Implementation contracts initialization allowed

Severity: Informational	Category: Configuration

Target:
- sft/payable/time-locked-erc20-container/contracts/TimelockedERC20ContainerDelegate.sol
- sft/payable/time-locked-erc20-container/contracts/TimelockedERC20ContainerConcrete.sol

Description

The TimelockedERC20ContainerDelegate and TimelockedERC20ContainerConcrete contracts have initializable functions that are meant to be called by the proxies. However, nothing prevents users from directly calling the initialize function on the implementation contracts.

According to the OpenZeppelin guideline, the _disableInitializers function call should be added to the constructor to lock implementation contracts that are designed to be called through proxies.

Recommendation

Consider adding a constructor with the _disableInitializers function call.

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
    _disableInitializers();
}
```

Status

This issue has been resolved by the team in commit e8a4e37.

6. Redundant TimelockType

Severity: Informational	Category: Redundancy

Target:
- sft/abilities/contracts/time-locked-erc20/ITimelockedERC20Concrete.sol
- sft/abilities/contracts/time-locked-erc20/TimelockedERC20Concrete.sol

Description

sft/abilities/contracts/time-locked-erc20/TimelockedERC20Concrete.sol:L89

```
if (info.timelockType == TimelockType.LINEAR || info.timelockType ==
TimelockType.ONE_TIME) {
```

sft/abilities/contracts/time-locked-erc20/TimelockedERC20Concrete.sol:L123

```
if (slotInfo_.timelockType == TimelockType.LINEAR || slotInfo_.timelockType ==
TimelockType.ONE_TIME) {
```

TimelockType.LINEAR and TimelockType.ONE_TIME are two types defined in the TimelockType enum. However, they are handled the same way in relevant functions. Since LINEAR is a more general concept than ONE_TIME, it is recommended to remove the latter from the TimelockType enum.

Recommendation

Consider removing the redundant ONE_TIME type and all its uses.

Status

The team has resolved this issue in commit e8a4e37 by implementing the logic to handle TimelockType.ONE_TIME.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit c9c0576:

File	SHA-1 hash
sft/abilities/contracts/slot-ownable/SlotOwnable.sol	875ed3e144ee8e6f1c2a412962c7b1152e3f67d3
sft/abilities/contracts/time-locked-erc20/ITimelockedERC20Concrete.sol	b66dcef883035819ecd1989c83246e699c02f246
sft/abilities/contracts/time-locked-erc20/ITimelockedERC20Delegate.sol	c8848a3cd026e733a6a47610a688cb37ac9c6d1e
sft/abilities/contracts/time-locked-erc20/TimelockedERC20Concrete.sol	86146b2f0fe14e515e492ab54c009661dfbbd14b
sft/abilities/contracts/time-locked-erc20/TimelockedERC20Delegate.sol	cfeffdfc245d332671e317c19daaecda04788e3b
sft/payable/time-locked-erc20-container/contracts/ITimelockedERC20ContainerConcrete.sol	d3c58ab8da98290030f7c242eb05f909efd69e44
sft/payable/time-locked-erc20-container/contracts/ITimelockedERC20ContainerDelegate.sol	d9d666335829532cf47a44272e346068a87e75fc
sft/payable/time-locked-erc20-container/contracts/TimelockedERC20ContainerConcrete.sol	fb0da1bbcae362597a2240d9a875a5d75b3f07ad
sft/payable/time-locked-erc20-container/contracts/TimelockedERC20ContainerDelegate.sol	2d5fb3137ca264dddc4aaa8368db00a6b7ed5eff