SALUS SECURITY

DEC 2023

CODE SECURITY ASSESSMENT

SHUI LST

Overview

Project Summary

- Name: Shui LST
- Platform: Conflux Network
- Language: Solidity
- Repository: https://github.com/Shui-LST/Shui-LST-Design
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	Shui LST
Version	v3
Type	Solidity
Dates	Dec 14 2023
Logs	Dec 06 2023; Dec 13 2023; Dec 14 2023

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	5
Total Low-Severity issues	1
Total informational issues	3
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	Incorrect balance verification in increaseLock() leads to DOS	High	Denial of Service	Resolved
2	Address zero becomes staker after burning	Medium	Business Logic	Resolved
3	Minter can mint sCFX without making a deposit	Medium	Centralization	Acknowledged
4	The owner of the bridge has the ability to withdraw all CFX from the deposits in sCFX	Medium	Centralization	Resolved
5	Owner can raise the price of sCFX at will	Medium	Centralization	Acknowledged
6	Use of dangerous proxy pattern	Medium	Upgradeability	Resolved
7	Modifications exist only for local variables	Low	Redundancy	Resolved
8	Shortening the lock period may result in incomplete withdrawal of unlocked funds	Informational	Business Logic	Acknowledged
9	Missing zero check	Informational	Data validation	Resolved
10	Use sendValue instead of transfer	Informational	Business Logic	Acknowledged

2.2 Notable Findings

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

1. Incorrect balance verification in increaseLock() leads to DOS

Severity: High	Category: Denial of Service
Target: - contracts/espace/VotingEscrow.sol	

Description

There is a invalid amount check in the increaseLock() function.

contracts/espace/VotingEscrow.sol:L108

```
function increaseLock(uint256 amount) public {
    ...
    require(
        _userLockInfo[msg.sender].amount + amount <= userStakeAmount(msg.sender),
"Governance: insufficient balance"
    );
    ...
}
```

The above check is intended to make sure the user's locked amount does not exceed the user's staked amount.

contracts/espace/VotingEscrow.sol:L79-99

```
function createLock(uint256 amount, uint256 unlockBlock) public {
    ...
    bool success = sCFX.transferFrom(msg.sender, address(this), amount);
    require(success, "sCFX transfer failed");
    ...

    uint256 _lockAmount = _scfxToCfx(amount);
    _userLockInfo[msg.sender] = LockInfo(_lockAmount, unlockBlock);
    ...
}
```

However, during the createLock() call, some of the user's sCFX is transferred to the VotingEscrow contract, resulting in a decrease in the return value from userStakeAmount(user).

As a result, when the user then calls the increaseLock() function, it's possible for _userLockInfo[msg.sender].amount + amount to be larger than userStakeAmount(msg.sender), resulting in the transaction being reverted.

Recommendation

It is recommended to change to mentioned check from

`_userLockInfo[msg.sender].amount + amount <= userStakeAmount(msg.sender)`
to
`amount <= userStakeAmount(msg.sender)`

Status

The team has resolved this issue in commit [3ed2bb1](3ed2bb1).

2. Address zero becomes staker after burning

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

Description

contracts/espace/sCFX.sol:L149-L154

```
function _beforeTokenTransfer(address from, address to, uint256 amount) internal
override {
    if (amount > 0) {
        _stakers.add(to);
    }
}
```

The code does not consider token burning. When sCFX is burned, the contract adds address(0) to the _stakers set. This causes the length of the _stakers set to increase by one, which affects the return values of the sCFX.stakerNumber() and sCFXBridge.eSpacePoolStakerNumber() functions.

Recommendation

It is recommended to handle the situation where to == address(0) in _beforeTokenTransfer().

Status

The team has resolved this issue in commit 3ed2bb1.

3. Minter can mint sCFX without making a deposit

Severity: Medium	Category: Centralization
Target: - contracts/espace/sCFX.sol	

Description

The sCFX token inherits the ERC20PresetMinterPauserUpgradeable contract.

ERC20PresetMinterPauserUpgradeable.sol:L64-L67

```
function mint(address to, uint256 amount) public virtual {
    require(hasRole(MINTER_ROLE, _msgSender()), "ERC20PresetMinterPauser: must have
minter role to mint");
    _mint(to, amount);
}
```

The MINTER_ROLE can call the mint() function to mint sCFX tokens without making a deposit. Since the sCFX token represents users' share, it is not advisable to configure a MINTER_ROLE who can mints sCFX without making a deposit.

Recommendation

It is recommended to remove the MINTER_ROLE and relevant minting functionality from the sCFX contract.

Status

This issue has been acknowledged by the team.

4. The owner of the bridge has the ability to withdraw all CFX from the deposits in sCFX

Severity: Medium	Category: Centralization

Target:
- contracts/espace/sCFX.sol
- contracts/core/sCFXBridge.sol

Description

contracts/espace/sCFX.sol:L43-L58

```
function deposit() public payable {
    ...
    uint256 amount = msg.value;
    _transferToBridge(amount);
    ...
}
```

The sCFX contract transfers all of the user's deposit to the bridge.

But the owner of the bridge has the ability to transfer all deposits through transferToEspacePool to the sCFXAddr address, which is an address that can be reset by the bridge owner by using setESpacePool.

contracts/core/sCFXBridge.sol:L313-321,L57-59,L352-354

```
function transferToEspacePool(uint256 amount) public onlyOwner {
    require(amount <= _balance(), "Not enough balance");
    CROSS_SPACE_CALL.transferEVM{value: amount}(_ePoolAddrB20());
}

function transferToEspacePool() public onlyOwner {
    uint256 _amount = _balance();
    CROSS_SPACE_CALL.transferEVM{value: _amount}(_ePoolAddrB20());
}
//L57-L59
function setESpacePool(address sCFXAddress) public onlyOwner {
    sCFXAddr = sCFXAddress;
}
function _ePoolAddrB20() internal view returns (bytes20) {
    return bytes20(sCFXAddr);
}
```

Recommendation

It is recommended to remove the setESpacePool function, a bridge should correspond to an sCFX contract. If you need to reset the sCFX please redeploy a bridge to correspond to it. We also recommend that the owner account use a multi-sig wallet to enhance its security.

Status

The team has resolved this issue in commit 7d71bcc.

5. Owner can raise the price of sCFX at will

Severity: Medium	Category: Centralization

Target:
- contracts/core/sCFXBridge.sol
- contracts/espace/sCFX.sol

Description

contracts/core/sCFXBridge.sol:L182-L184

```
function eSpaceAddAssets(uint256 amount) public onlyOwner {
    CROSS_SPACE_CALL.callEVM(_ePoolAddrB20(),
abi.encodeWithSignature("addAssets(uint256)", amount));
}
```

contracts/espace/sCFX.sol:L130-L132

```
function addAssets(uint256 delta) public onlyBridge {
    totalDeposited += delta;
}
```

The owner can increase the totalDeposited variable in sCFX by calling the eSpaceAddAssets() function in the sCFXBridge contract.

Should the owner's private key being compromised, the attacker can transfer CFX to sCFX with sCFX.deposit(). They can then exploit sCFXBridge.eSpaceAddAssets -> sCFX.addAdssets to increase the price of sCFX against CFX. Finally, they can transfer sCFX back to CFX to make a profit.

Recommendation

It is recommended to transfer 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 acknowledged by the team.

6. Use of dangerous proxy pattern

Severity: Medium	Category: Upgradeability
Target: - contracts/utils/Proxy1967.sol	

Description

Instead of using OpenZepplin's Transparent Proxy, the project uses its custom Proxy1967 contract as the proxy.

The Proxy1967 contract inherits the Ownable contract, resulting in the slot 0 storage to be used by the _owner variable of the proxy contract.

Therefore, a call to the transferOwnership() would override the data in slot 0.

The custom Proxy1967 contract is compatible with the PoSOracle, PoSPool, sCFXBridge, and EVotingEscrow contract. These contracts inherit the Ownable contract and reuse the slot 0 set by the proxy contract as the _owner in the implementation logic.

However, the Proxy1967 contract is NOT compatible with the SHUI contract and the sCFX contract. The slot 0 for these contracts is used by the _initialized and _initializing variables in implementation's initializable contract, and would be overridden by the Proxy1967 contract.

Recommendation

It is recommended to use OpenZeppelin's Transparent Proxy instead of Proxy1967 as the proxy contract. When using the Transparent Proxy, ensure to initialize the _owner variable in the implementation contract during initialization.

Status

The team has resolved this issue in commit b50cad2, 5ace8a7, 13d1db1.

7. Modifications exist only for local variables

Severity: Low	Category: Redundancy
Target: - contracts/espace/VotingEscrow.sol	

Description

contracts/espace/VotingEscrow.sol:L253-L266

```
function _lockStake(uint256 unlockBlock) internal {
    if (unlockBlock > lastUnlockBlock) {
        lastUnlockBlock = unlockBlock;
    }

    uint256 accAmount = 0;
    uint256 blockNumber = lastUnlockBlock;

    while (blockNumber >= block.number) {
        accAmount += globalLockAmount[blockNumber];

        blockNumber -= QUARTER_BLOCK_NUMBER;
    }
}
```

All operations after the if statement only affect local variables, but this function is used in the createLock, increaseLock, extendLockTime contracts.

Recommendation

We recommend that the project owner check if there are other features of _lockStake that are not implemented.

Status

The team has resolved this issue in commit [b50cad2](b50cad2).

2.3 Informational Findings

8. Shortening the lock period may result in incomplete withdrawal of unlocked funds

Severity: Information	Category: Business Logic
Target: - contracts/utils/VotePowerQueue.sol	

Description

contracts/utils/VotePowerQueue.sol:L53-L63

```
function collectEndedVotes(InOutQueue storage q) internal returns (uint256) {
    uint256 total = 0;
    for (uint256 i = q.start; i < q.end; i++) {
        if (q.items[i].endBlock > block.number) {
            break;
        }
        total += q.items[i].votePower;
        dequeue(q);
    }
    return total;
}
```

In the calculation logic of collectEndedVotes() and sumEndedVotes(), the project side assumes that the endBlock of the queue is incremental. So once there is an item.endBlock > block.number, the loop is stopped.

But if the owner calls setLockPeriod() to shorten the lock period, this will break the incremental relationship.

This will result in funds that should have been unlocked remaining locked in the contract.

Attach Scenario

1. block.number = 1, lockPeriod = 100: Alice stake 3 ETH, endBlock1 = 101

2. block.number = 2, Owner adjusts lockPeriod to 50

3. block.number = 10, Alice adds 1 ETH to the stake, endBlock2 = 60

4. block.number = 70, Alice's unlocked amount is 0 (should actually be 1 ETH).

Recommendation

We expect the project team to be aware of this potential issue of shortening the lock period.

Status

This issue has been acknowledged by the team.

9. Missing zero check

Severity: Informational	Category: Data validation
Target: - contracts/core/PoSPool.sol	

Description

In PoSPool contracts, many external functions lack checks against zero value.

If decreaseStake()/withdrawStake() is called with 0 as the input votePower, or claimInterest() is called with 0 as the input amount, an useless Event would be emitted, which might waste resources during off-chain logging. Moreover, the _updateAPY() internal call is triggered, resulting in trivial data being added to the apyNodes, which might increase the gas cost for iterating the apyNodes.

contracts/core/PoSPool.sol:L181

```
function _updateAPY() private {
    ...
    apyNodes.enqueueAndClearOutdated(node, outdatedBlock);
}
```

Recommendation

It is recommended to add zero value checks for the mentioned functions.

Status

The team has resolved this issue in commit 3ed2bb1.

10. Use sendValue instead of transfer

Severity: Informational	Category: Business Logic
Target: - contracts/espace/sCFX.sol	

Description

The use of transfer() for native token transfer will inevitably make the transaction fail when:

a) The claimer smart contract does not implement a payable function.

b) The claimer smart contract does implement a payable fallback which uses more than 2300 gas unit.

c) The claimer smart contract implements a payable fallback function that needs less than 2300 gas units but is called through proxy, raising the call's gas usage above 2300.

d) Additionally, using higher than 2300 gas might be mandatory for some multisig wallets.

Recommendation

It is recommended to use the sendValue() function provided by OpenZeppelin to transfer native tokens.

When using sendValue() you should follow the checks-effects-interactions (CEI) pattern or consider using nonReentrant to prevent reentrancy attacks.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 8107d5d:

File	SHA-1 hash
PoolContext.sol	062b8399891a9e51a6da9340d6bbae09aa8d8b1b
PoSOracle.sol	8a3cd1d2b3d89da0ef19823328d14963989288dc
PoSPool.sol	f0f5d1e14971a8fff792f85a636f68cee1190420
sCFXBridge.sol	7a039bdc9a69b1cae2268b7f21380727052774c0
sCFX.sol	e52dc6c938707c9f5b598881af451024db346238
SHUI.sol	9e9ec25e43ec74cf51f2d702376eb2a921892df6
VotingEscrow.sol	64710cbd5af67f30e182e40c884345c54dcd6ee3
PoolAPY.sol	ed5b459042a1d7886c5e8bd66741406dd3d2994a
Proxy1967.sol	ec17e7ab41393c769e3b4fe550cd5870e20d08e0
ProxyAdmin.sol	c02b01e0ca7c2a138ab60cd9ac22e6bb773df31f
RedeemQueue.sol	7ef876d7632885d1fc3831d4abc58d4c1bf7b4b7
TransparentProxy.sol	3e2a49f331d0dad3d8e0db67c3b45e1fe80060aa
VotePowerQueue.sol	13701b6900b053b00a5b102d2ff8ceb9b43c9130