SALUS SECURITY

APR 2023

CODE SECURITY ASSESSMENT

SATORI

Overview

Project Summary

- Name: Satori
- Platform: EVM-compatible chains
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Satori
Version	v2
Type	Solidity
Dates	Apr 13 2023
Logs	Apr 02 2023; Apr 13 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	5
Total Low-Severity issues	1
Total informational issues	3
Total	9

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 **4**

1.1 About SALUS 4

1.2 Audit Breakdown 4

1.3 Disclaimer 4

Findings **5**

2.1 Summary of Findings 5

2.2 Notable Findings 6

1. Flawed logic for selecting vault in depositCoin() 6
2. Lack of sanity check for _coinAddr in depositCoin() and authorizedWithdrawCoin() 8
3. Deleting an EnumerableSet may corrupt its internal data 9
4. Malicious actors can call authorizedWithdrawCoin() without providing signatures when requiredSignatures == 0 11
5. Centralization risk 12
6. Return value of 0 from ecrecover not checked 13

2.3 Informational Findings 14

7. Use of floating compiler version 14
8. Redundant code 15
9. Could add the vault address to the LogDepositSuccess() event 16

Appendix **17**

Appendix 1 - Files in Scope 17

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	Flawed logic for selecting vault in depositCoin()	Medium	Business Logic	Resolved
2	Lack of sanity check for _coinAddr in depositCoin() and authorizedWithdrawCoin()	Medium	Validation	Resolved
3	Deleting an EnumerableSet may corrupt its internal data	Medium	Business Logic	Resolved
4	Malicious actors can call authorizedWithdrawCoin() without providing signatures when requiredSignatures == 0	Medium	Validation	Resolved
5	Centralization risk	Medium	Centralization	Acknowledged
6	Return value of 0 from ecrecover not checked	Low	Validation	Resolved
7	Use of floating compiler version	Informational	Configuration	Resolved
8	Redundant code	Informational	Redundancy	Resolved
9	Could add the vault address to the LogDepositSuccess() event	Informational	Logging	Acknowledged

2.2 Notable Findings

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

1. Flawed logic for selecting vault in depositCoin()

Severity: Medium	Category: Business logic
Target: - contracts/DwBiz.sol	

Description

contracts/DwBiz.sol:L36

```
mapping (address => uint256) public maxReserve; // The maximum reserve amount of assets
in the direct vault.
```

The maxReserve mapping stores the maximum reserve amount in the direct vault for a given asset.

contracts/DwBiz.sol:L68-L89

```
function depositCoin(
    uint256 _bizNo,
    uint256 _bizTime,
    address _coinAddr,
    uint256 _amount
) external payable ensure(_bizTime) whenNotPaused nonReentrant {
    require(_amount > 0 && _amount >= minDepositAmount[_coinAddr], "Deposit amount is
too low");
    address defaultVault;
    if (_coinAddr == address(0)) {
        defaultVault = directVault.balance >= maxReserve[_coinAddr] ? signVault :
directVault;
        require(_amount == msg.value, "Insufficient balance");
        (bool success, ) = defaultVault.call{value: _amount}(new bytes(0));
        require(success, "ETH transfer failed");
    } else {
        uint256 _balance = IERC20Upgradeable(_coinAddr).balanceOf(directVault);
        defaultVault = _balance >= maxReserve[_coinAddr] ? signVault : directVault;
        require(IERC20Upgradeable(_coinAddr).balanceOf(msg.sender) >= _amount,
"Insufficient balance");
        require(IERC20Upgradeable(_coinAddr).allowance(msg.sender, address(this)) >=
_amount, "Insufficient allowance");
        IERC20Upgradeable(_coinAddr).safeTransferFrom(msg.sender, defaultVault,
_amount);
    }
    emit LogDepositSuccess(_bizNo, _bizTime, _coinAddr, msg.sender, _amount);
}
```

However, the checks in depositCoin() are not sufficient to ensure the asset in directVault does not exceed maxReserve[asset]. The _balance >= maxReserve[_coinAddr] condition check compares the current balance with maxReserve, instead of the balance after deposit.

Let's assume the maxReserve[USDT] is 1 and the current balance of USDT in directVault is zero, if the user calls depositCoin() with _amount == 100, the 100 wei USDT will be sent to

directVault instead of signVault, resulting in the balance in directVault exceeds the maxReserve.

Recommendation

Consider changing `defaultVault = directVault.balance >= maxReserve[_coinAddr] ? signVault : directVault;` to `defaultVault = directVault.balance + _amount >= maxReserve[_coinAddr] ? signVault : directVault;`, and `defaultVault = _balance >= maxReserve[_coinAddr] ? signVault : directVault;` to `defaultVault = _balance + _amount >= maxReserve[_coinAddr] ? signVault : directVault;`.

Status

The team has resolved this issue by changing the condition from `directVault.balance >= maxReserve[_coinAddr]` to `directVault.balance + _amount >= maxReserve[_coinAddr]`.

2. Lack of sanity check for _coinAddr in depositCoin() and authorizedWithdrawCoin()

Severity: Medium	Category: Validation
Target: - contracts/DwBiz.sol	

Description

contracts/DwBiz.sol:L32

```
mapping (address => bool) public depositCoins;
```

The depositCoins indicates whether a coin is whitelisted for deposit. It can be set by the owner using setDepositCoins(). However, the depositCoin() function does not verify that depositCoins[_coinAddr] is true.

contracts/DwBiz.sol:L34

```
mapping (address => bool) public withdrawCoins;
```

The withdrawCoins indicates whether a coin is whitelisted for withdrawal. The owner can set this using setWithdrawCoins(). However, the authorizedWithdrawCoin() function does not check if withdrawCoins[_coinAddr] is true.

It's safer for the project to only interact with a predetermined set of coin addresses, rather than allowing users to input arbitrary ones.

Recommendation

Consider checking whether _coinAddr is already set for deposit in depositCoin() and for withdrawal in authorizedWithdrawCoin().

Status

This issue has been resolved by the team. The team added a `require(depositCoins[_coinAddr], "Coin address not support");` check in depositCoin() and a `require(withdrawCoins[_coinAddr], "Coin address not support");` check in authorizedWithdrawCoin().

3. Deleting an EnumerableSet may corrupt its internal data

Severity: Medium	Category: Business logic

Target:
- contracts/DwMultiSignVault.sol

Description

contracts/DwMultiSignVault.sol:L47-L69

```
function withdraw(
    uint256 _bizNo,
    address _srcAsset,
    uint256 _assetAmt,
    address _toUser
) public whenNotPaused nonReentrant onlyAuthorized {
    require(isRoleMemberExist(WHITELIST_ROLE, _toUser), "Address not whitelist");
    require(_assetAmt > 0, "Invalid amount");
    bytes32 _hash = keccak256(abi.encodePacked(_bizNo, _srcAsset, _assetAmt, _toUser));
    grantRole(_hash, msg.sender);
    if (getRoleMemberCount(_hash) >= requiredSignatures) {
        delete roleMembers[_hash];
        if (_srcAsset == address(0)) {
            require(address(this).balance >= _assetAmt, "Insufficient balance");
            (bool success, ) = _toUser.call{value: _assetAmt}(new bytes(0));
            require(success, "ETH transfer failed");
        } else {
            require(IERC20(_srcAsset).balanceOf(address(this)) >= _assetAmt,
"Insufficient balance");
            IERC20(_srcAsset).safeTransfer(_toUser, _assetAmt);
        }
        emit LogWithdraw(_srcAsset, _assetAmt, msg.sender, _toUser);
    }
}
```

The withdraw() function in the DwMultiSignVault contract uses the delete operator to clean a EnumerableSet struct. However, since the EnumerableSet struct contains an inner mapping, the delete operation cannot clean the inner mapping and would instead corrupt the struct's data (see the warning in the code comment of EnumerableSet).

To illustrate, assuming Alice and Bob are two AUTHORIZED_ROLEs and the requiredSignatures == 2. After they both call withdraw(), the getRoleMemberCount(_hash) would return 0. However isRoleMemberExist(_hash, Alice) and isRoleMemberExist(_hash, Bob) both return true, indicating that the underlying EnumerableSet is in a corrupted state.

Recommendation

We recommend that you do not delete the EnumerableSet, but instead use the usedHashes mechanism that you have used in DwBiz.authorizedWithdrawCoin() to invalidate a used hash. Specifically, you could define a usedHashes mapping variable, check if the hash is used in withdraw(), revert if it is, and set it to true when the funds for that hash are withdrawn.

Status

This issue has been resolved by the team. The team employed a `usedBizNos` mapping to check if a BizNo had already been used.

4. Malicious actors can call authorizedWithdrawCoin() without providing signatures when requiredSignatures == 0

Severity: Medium	Category: Validation
Target: - contracts/DwBiz.sol	

Description

The requiredSignatures state variable in the DwBiz contract represents the number of signatures needed to call authorizedWithdrawCoin(). If requiredSignatures is zero, no signatures are required to withdraw coins.

However, the variable requiredSignatures is not initialized in the initialize() function and defaults to zero. This could create a possible attack scenario:

1. The owner deploys the contracts.
2. The owner sets maxReserve and deposits funds into the directVault.
3. Then the owner wants to add signers and set requiredSignatures using adminUpdateSigners().
4. An attacker could front-run the adminUpdateSigners() transaction, and withdraw funds using authorizedWithdrawCoin() since requiredSignatures is 0.

Furthermore, there is no validation for the _signThreshold parameter in adminUpdateSigners() and revokeAuthorization(). If the owner mistakenly sets _signThreshold to zero, the requiredSignatures state variable will be set to zero, allowing malicious actors to withdraw funds without providing signatures.

Recommendation

Consider adding a `require(requiredSignatures != 0);` check in authorizedWithdrawCoin().

Status

The team has resolved this issue by making the minimum number of required signatures to two.

5. Centralization risk

Severity: Medium	Category: Centralization
Target: - all	

Description

The project has privileged accounts.

The owner of the DwBiz contract:
- can pause the contract
- can add and revoke authorized signers
- can set the number of required signatures
- can set minimal deposit amount, minimal withdraw amount and max reserves for coins

The owner of the DwDirectVault contract:
- can pause the contract
- can add and revoke authorized accounts

The owner of the DwMultiSignVault contract:
- can pause the contract
- can add and revoke authorized signers
- can set the number of required signatures
- can grant and revoke whitelisted receiver accounts

If these privileged owner accounts are plain EOA accounts, this poses a risk to the users. If the owner's private key is compromised, an attacker could use the above privileged operation to attack the project.

Moreover, the upgradeable proxy pattern is used in the DwBiz contract. The proxy admin controls the upgrade mechanism to upgradeable proxies, and 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.

Recommendation

Consider transferring the privileged roles to multi-sig accounts.

Status

This issue has been acknowledged by the team.

6. Return value of 0 from ecrecover not checked

Severity: Low	Category: Validation
Target: - contracts/DwBiz.sol	

Description

contracts/DwBiz.sol:L91-L117

```
function authorizedWithdrawCoin(
    uint256 _bizNo,
    uint256 _bizTime,
    address _coinAddr,
    address _toAddr,
    uint256 _amount,
    uint8[] memory _vArr,
    bytes32[] memory _rArr,
    bytes32[] memory _sArr
) external ensure(_bizTime) whenNotPaused nonReentrant {
    ...
    for (uint256 i = 0; i < _vArr.length; i++) {
        address signer = ecrecover(msgHash, _vArr[i], _rArr[i], _sArr[i]);
        require(isRoleMemberExist(AUTHORIZED_ROLE, signer), "Invalid signer");
        ...
    }
    IDwDirectVault(directVault).withdraw(_bizNo, _coinAddr, _amount, _toAddr);
    emit LogWithdrawSuccess(_bizNo, _bizTime, _coinAddr, directVault, msg.sender,
_toAddr, _amount);
}
```

The **ecrecover()** function returns zero for an invalid signature (see Solidity documentation). The authorizedWithdrawCoin() function does not check if the returned signer from ecrecover() is not equal to address(0).

If the owner accidentally assigns address(0) as the AUTHORIZED_ROLE using adminUpdateSigners(), then the actual number of required signatures in authorizedWithdrawCoin() will be one less than the requiredSignatures. This is because a malicious actor can use an invalid signature to skip one iteration in the signature check loop.

Recommendation

We recommend adding `require(signer != address(0), "...");` after the ecrecover() call. Alternatively, you can use the `ECDSA.recover(hash, v, r, s)` function from OpenZeppelin, which reverts for invalid signatures, to replace ecrecover().

Status

The team has resolved this issue by adding the `signer != address(0)` check.

2.3 Informational Findings

7. Use of floating compiler version

Severity: Informational	Category: Configuration
Target: - all	

Description

```
pragma solidity ^0.8.0;
```

The project uses a floating compiler version ^0.8.0.

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Recommendation

Consider locking the pragma version.

Status

This issue has been resolved by the team.

8. Redundant code

Severity: Informational	Category: Redundancy
Target: - contracts/DwBiz.sol - contracts/interfaces/IDwMultiSignVault.sol	

Description

1. contracts/DwBiz.sol:L10

```
import "@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol";
```

The AddressUpgradeable contract is imported but not used in the DwBiz contract, so it can be removed.

2. contracts/DwBiz.sol:L21-23

```
bytes32 public constant PERMIT_TYPEHASH = keccak256(
    abi.encodePacked("Permit(address asset,address to,uint256 amount,uint256
deadline,uint256 salt)")
);
```

The keccak256 hash of an ABI-encoded string bytes is the same as the hash of the string itself, i.e. keccak256(abi.encodePacked(aString)) == keccak256(aString). Therefore, the highlighted code can be removed.

3. The IDwMultiSignVault interface is not used in the project. So it can be removed.

Recommendation

Consider removing the redundant codes.

Status

This team has resolved this issue by removing the redundant codes.

9. Could add the vault address to the LogDepositSuccess() event

Severity: Informational	Category: Logging

Target:
- contracts/DwBiz.sol
- contracts/interfaces/IDwBiz.sol

Description

contracts/interfaces/IDwBiz.sol:L6-L22

```
event LogDepositSuccess(
    uint256 _bizNo,
    uint256 _bizTime,
    address _coinAddr,
    address _fromAddr,
    uint256 _amount
);

event LogWithdrawSuccess(
    uint256 _bizNo,
    uint256 _bizTime,
    address _coinAddr,
    address _fromAddr,
    address _callerAddr,
    address _toAddr,
    uint256 _amount
);
```

The LogWithdrawSuccess() event has a parameter (_fromAddr) that indicates the vault address from which funds are withdrawn. However, the LogDepositSuccess() event does not have a parameter to show where the funds are deposited.

Recommendation

Consider adding a parameter in the LogDepositSuccess() event to indicate the vault address where the funds are deposited.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
contracts/DwBiz.sol	8ad638ca910d323d327032b6815802e26480809f
contracts/DwDirectVault.sol	88dba83d5d87f71630022e6a0bcee00ee84b8d02
contracts/DwMultiSignVault.sol	534b32b13a0007a7d4c73f16ef3fe8c01687dcf5
contracts/interfaces/IDwBiz.sol	c34cda4b5bdb8291c4d8e744b5f0bb618c1289ab
contracts/interfaces/IDwDirectVault.sol	d55a98d92d38da9e39685cbba5a71940dfd4e8e3
contracts/interfaces/IDwMultiSignVault.sol	e7b7271614f7061e13e5e34fcc60f7344022cda5
contracts/interfaces/IDwVault.sol	1539923c1f4abe65d6c5e9e98587169a7a56711c