Veridise Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Satori DwBiz

Veridise Inc.
October 1, 2024

► **Prepared For:**

Satori Finance
https://satori.finance/

► **Prepared By:**

Benjamin Mariano
Xiangan He
Benjamin Gaska

► **Contact Us:** contact@veridise.com

► **Version History:**

May 08, 2023	Initial Draft
May 15, 2023	V1

Contents

1 Executive Summary

From Apr. 12, 2023 to May 8, 2023, Satori Finance engaged Veridise to review the security of the DwBiz, a set of smart contracts which implement a basic vault mechanism that allows users to deposit and withdraw funds. Veridise conducted the assessment over 3 person-weeks, with 3 engineers reviewing code over 1 weeks. The auditing strategy involved an extensive manual review performed by Veridise engineers.

Code assessment. The Satori Finance developers provided the source code of the DwBiz contracts for review. Documentation of the code was quite limited. A general whitepaper was shared, however the whitepaper did not describe the intended behavior of the smart contracts under review in the audit. There was also very little documentation/comments within the code. No tests were provided to the Veridise auditors.

Summary of issues detected. The audit uncovered 12 issues, 2 of which are assessed to be of high or critical severity by the Veridise auditors. Some of the errors found include a centralized risk around a single trusted user being able to drain all contract funds (V-VUL-SAT-001), an incorrect check that can allow unintended users to withdraw funds (V-VUL-SAT-002), as well as possible DOS of user withdrawals when contracts are paused (V-VUL-SAT-005).

Recommendations. After auditing the protocol, we have a two suggestions to improve the DwBiz project. First, we strongly suggest improving documentation so that intended behavior is more clear. Second, if not already implemented, we suggest testing the code (no tests were provided to the auditing team).

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
DwBiz	NA	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Apr. 12 - May 8, 2023	Manual	3	3 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	0	0
High-Severity Issues	2	2
Medium-Severity Issues	3	3
Low-Severity Issues	3	3
Warning-Severity Issues	2	2
Informational-Severity Issues	2	2
TOTAL	12	12

Table 2.4: Category Breakdown.

Name	Number
Logic Error	6
Denial of Service	2
Maintainability	2
Data Validation	1
Gas Optimization	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Satori Finance's smart contracts.

In our audit, we sought to answer the following questions:

- Is there any way that malicious users can steal funds from the vault?
- Are signature requirements on withdrawals from the multi-sig vault followed?
- Can users lose funds when interacting with the vault?
- Can funds be locked in the vault?
- Can a single user get past multi-sig protections or drain a contract?

3.2 Audit Scope

Scope The scope of this audit consisted of all contracts provided in the `/contracts` folder. This includes the following contracts:

- interfaces/IDwBiz.sol
- interfaces/IDwDirectVault.sol
- interfaces/IDwMultiSignVault.sol
- interfaces/IDwVault.sol
- interfaces/IOwnerManager.sol
- interfaces/IUpgradeManager.sol
- manager/CommonManger.sol
- manager/PrivilegeManager.sol
- manager/UpgradeManager.sol
- CommonRoleManager.sol
- DwBiz.sol
- DwDirectVault.sol
- DwMultiSignVault.sol

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowleged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-VUL-SAT-001	Authorized user can drain contract	High	Acknowledged
V-VUL-SAT-002	isContract check not guaranteed to work	High	Acknowledged
V-VUL-SAT-003	Unchecked signature requirement setting	Medium	Fixed
V-VUL-SAT-004	DoS of user withdrawals when vault is paused	Medium	Intended Behavior
V-VUL-SAT-005	Large arrays and for loops may cause DoS	Medium	Acknowledged
V-VUL-SAT-006	Possible lost funds	Low	Acknowledged
V-VUL-SAT-007	Possible withdrawal reentrancy	Low	Fixed
V-VUL-SAT-008	Withdraw reverts with more signatures	Low	Fixed
V-VUL-SAT-009	Repeated code	Warning	Acknowledged
V-VUL-SAT-010	Possible lost funds on deposit	Warning	Acknowledged
V-VUL-SAT-011	Bad error reporting text	Info	Fixed
V-VUL-SAT-012	Redundant check	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-VUL-SAT-001: Authorized user can drain contract

Severity	High	**Commit**	n/a
Type	Logic Error	**Status**	Acknowledged
File(s)	DwDirectVault.sol,DwBiz.sol		
Location(s)	withdraw, authorizedWithdrawCoin		

The function `withdraw` in `DwDirectVault` can be used to entirely drain the contract of funds by any authorized user.

```
function withdraw(
        uint256 _bizNo,
        address _srcAsset,
        uint256 _assetAmt,
        address _toUser
    ) external whenNotPaused nonReentrant onlyAuthorized {
        require(_assetAmt > 0, "Invalid amount");
        require(!_toUser.isContract(), "Only EOA");
        require(withdrawHistory[_bizNo] == 0, "Already exists");
        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);
        }
        withdrawHistory[_bizNo] = _assetAmt;
        emit LogWithdraw(_srcAsset, _assetAmt, msg.sender, _toUser);
    }
```

A similar issue is present for the function `authorizedWithdrawCoin` in `DwBiz`, although this does at least require the required number of signatures to approve.

Impact The contract's funds can be entirely drained if any authorized user is malicious.

Recommendation It seems like certain `bizNos` should be associated with amounts that can be withdrawn and users who can withdraw them. It may be advisable to, at the very least, limit the amount that can be withdraw by any individual authorized user.

Developer Response Developers are okay with this because only trusted users will be allowed to invoke these functions.

4.1.2 V-VUL-SAT-002: isContract check not guaranteed to work

Severity	High	**Commit**	n/a
Type	Logic Error	**Status**	Acknowledged
File(s)	`DwDirectVault.sol`		
Location(s)	withdraw		

The function `withdraw` tries to rule out recipients which are not contracts through the following check:

```
1 require(!_toUser.isContract(), "Only EOA");
```

However, `isContract()` can return `true` if the address corresponds to the caller which is called from a constructor.

Impact `_toUser` could be a contract.

Recommendation Use caution when trying to restrict callers to be EOA. Depending on the desired behavior of `withdraw`, it may be advisable to whitelist recipients.

Developer Response The developers recognize the issue but say the only point of this check is to avoid reentrancy, which they handle with the `nonReentrant` modifier. We have suggested they remove the check if it is unnecessary, which developers plan to do.

4.1.3 V-VUL-SAT-003: Unchecked signature requirement setting

Severity	Medium	**Commit**	n/a
Type	Logic Error	**Status**	Fixed
File(s)	DwBiz.sol,DwMultiSignVault.sol		
Location(s)	revokeAuthorization		

The function `revokeAuthorization` allows a user to set `requiredSignatures` to anything, despite checks everywhere else that it must be set to something `>= 2` and must be `<= signers.length`.

```
function revokeAuthorization(address[] memory _signers, uint256 _signThreshold)
    external onlyOwner {
   for (uint256 i = 0; i < _signers.length; i++) {
       revokeRole(AUTHORIZED_ROLE, _signers[i]);
   }
   requiredSignatures = _signThreshold;
   emit LogAdminSignerRevoked(msg.sender, _signThreshold, _signers);
 }
```

Impact If `requiredSigners` is set to either `1` or `0`, any signer can successfully call withdraw to withdraw any amount they want from the `DwMultiSignVault`.

Recommendation Check to make sure `requiredSigners` cannot be set to an arbitrary value on this call.

Developer Response The developers addressed through a refactor that separated the threshold setting and revocation and adding appropriate checks as follows in `DwBiz.sol` and `DwMultiSignVault.sol`:

```
function removeSigners(address[] memory _signers) external onlyOwner {
    require(getRoleMemberCount(AUTHORIZED_ROLE) - _signers.length >=
    requiredSignatures, "params error");
    for (uint256 i = 0; i < _signers.length; i++) {
        revokeRole(AUTHORIZED_ROLE, _signers[i]);
    }
}

function setSignThreshold(uint256 _signThreshold) external onlyOwner {
    require(_signThreshold>=2, "params error");
    requiredSignatures = _signThreshold;
}
```

4.1.4 V-VUL-SAT-004: DoS of user withdrawals when vault is paused

Severity	Medium	**Commit**	n/a
Type	Denial of Service	**Status**	Intended Behavior
File(s)	`DwBiz.sol`,`DwDirectVault.sol`,`DwMultiSignVault.sol`		
Location(s)	authorizedWithdrawCoin, withdraw		

Currently, `authorizedWithdrawCoin` and other vault withdrawal mechanisms rely on `whenNotPaused`, which causes functions to revert if the market is `paused`. Funds are irretrievable when markets are paused as a result.

```
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 {
```

Snippet 4.1: The implementation of `DwBiz`'s withdraw function

Impact The revert causes funds to be stuck inside vaults when the markets are paused, with no way to retrieve the funds under the desired paused conditions.

Recommendation Funds should still be able to be withdrawn even when markets are paused.

Developer Response Under some extreme circumstances, the developers would like the ability to cutoff withdraws, even with the potential risk to users.

4.1.5 V-VUL-SAT-005: Large arrays and for loops may cause DoS

Severity	Medium	Commit	n/a
Type	Denial of Service	Status	Acknowledged
File(s)	DwBiz.sol,DwDirectVault.sol,DwMultiSignVault.sol		
Location(s)	authorizedWithdrawCoin, revokeAuthorization, etc.		

For-loops are widely utilized in the Satori contracts. Iterating operations through an array, however, can potentially result in a very gas-expensive function call, to the point where gas costs may cause a DoS on various important functionalities (e.g. updating signers, revoking authorization, etc.)

The following snippet in `authorizedWithdrawCoin`, for example, is likely going to revert due to out of gas issues with larger quorums. See a few other example snippets as follows

```
        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"); //
    @audit-info checks auth of every signer
            for (uint256 j = 0; j < i; j++) {
                require(signatures[j] != signer, "Duplicate signer"); // @audit-info
    ensures no signer is duplicated
            }
            signatures[i] = signer;
        }
```

Snippet 4.2: Iterating two potentially large arrays, for example, may cause a gas-related revert.

```
function adminUpdateSigners(address[] memory _signers, uint256 _signThreshold)
    external onlyOwner {
        require(_signers.length >= _signThreshold && _signThreshold >= 2, "Params
    error");
        for (uint256 i = 0; i < _signers.length; i++) {
            grantRole(AUTHORIZED_ROLE, _signers[i]);
        }
        requiredSignatures = _signThreshold;
        emit LogAdminSignerUpdated(msg.sender, _signThreshold, _signers);
    }
```

Snippet 4.3: A list of too many signers may cause this and `revokeAuthorization` to be DoS due to gas-related issues, disabling authority updates.

Impact If the quorum of signers and authorized users get too large, it may become impossible to remove users from various groups or perform key actions such as `withdraw` from the protocol.

Recommendation Generally more gas-efficient methods of operation instead of for-loops through an array. Only use for-loops if it is confirmed that the quorum would be relatively small.

```
function setDepositCoins(address[] memory _depositCoins, uint256[] memory
    _depositBalances) external onlyOwner {
        require(_depositCoins.length == _depositBalances.length, "Invalid deposit
    coins");
        for (uint256 i = 0; i < _depositCoins.length; i++) {
            address coin = _depositCoins[i];
            depositCoins[coin] = true;
            minDepositAmount[coin] = _depositBalances[i];
        }
    }
```

Snippet 4.4: Too long of a list of assets, as another example, can disable setting deposit and withdraw coins

Developer Response The developers acknowledge that this could be an issue, but because the functions affected are either admin-only functions or the iterations of loops can be controlled by admins (through the number of signers which should be `<= 10`) they are not concerned about this issue and have chosen to keep the loops as they are.

4.1.6 V-VUL-SAT-006: Possible lost funds

Severity	Low	Commit	n/a
Type	Logic Error	Status	Acknowledged
File(s)	DwDirectVault.sol,DwMultiSignVault.sol		
Location(s)	receive		

The function `receive` can receive funds from a user, but does not update any state. Instead, it simply emits an event. This function is intended to be invoked by calls from `DwBiz`, however, if another user accidentally sends funds to the contract, they cannot be retrieved.

```
receive() external payable {
    if (msg.value > 0) {
        emit Received(msg.sender, msg.value);
    }
}
```

Impact Users could unintentionally lose funds.

Recommendation As the only caller of this contract seems like it should be `DwBiz`, perform controls that only allow whitelisted contracts to contribute funds to the vault.

Developer Response Developers consider such a mistake on users and thus are not concerned with addressing the issue at this time.

4.1.7 V-VUL-SAT-007: Possible withdrawal reentrancy

Severity	Low	**Commit**	n/a
Type	Logic Error	**Status**	Fixed
File(s)	`DwDirectVault.sol`		
Location(s)	withdraw		

The function `withdraw` can have enable reentrancy through the following call:

```
(bool success, ) = _toUser.call{value: _assetAmt}(new bytes(0));
```

After this call, the `withdrawHistory` is updated as follows:

```
withdrawHistory[_bizNo] = _assetAmt;
```

Because `withdraw` is guarded with a reentrancy guard, the consequences of this are not serious. However, any external contract which relies on the `public` value of `withdrawHistory` could be manipulated via the external call.

Impact External contracts relying on `withdrawHistory` could be manipulated via a reentrant call.

Recommendation Move write of `withdrawHistory` to before external call.

4.1.8 V-VUL-SAT-008: Withdraw reverts with more signatures

Severity	Low	**Commit**	n/a
Type	Logic Error	**Status**	Fixed
File(s)	DwBiz.sol		
Location(s)	authorizedWithdrawCoin		

The function `authorizedWithdrawCoin` checks that the length of `_vArr` is greater than or equal to the required signatures:

```
require(_vArr.length >= requiredSignatures && _vArr.length > 0, "Invalid number of
    signatures provided");
```

However, when verifying the signatures, the results are saved in the address array `signatures` which is initialized as follows:

```
address[] memory signatures = new address[](requiredSignatures);
```

This array is only length of `requiredSignatures`, so assigning any signatures to it beyond the required number of signatures will cause the function to revert.

Impact This could cause unexpected reverts of the function.

Recommendation If it is intended that only the exact required signatures should be sent, then check that in the require. Otherwise, make the signatures array dynamic length.

It may also be worthwhile to include requires that the length of `_rArr` and `_sArr` are the same length as `_vArr` to avoid unexpected reverts due to this difference.

4.1.9 V-VUL-SAT-009: Repeated code

Severity	Warning	**Commit**	n/a
Type	Maintainability	**Status**	Acknowledged
File(s)	`DwBiz.sol`,`DwMultiSignVault.sol`,`CommonManger.sol`,`etc.`		
Location(s)	withdraw		

A number of functions are defined identically multiple times involving the handling of roles and authorization: `pause`, `revokeAuthorization`, `adminUpdateSigners`, `grantRole`, `revokeRole`, `getRoleMembers`, `getRoleMember`, `getRoleMemberCount`, and `isRoleMemberExist`.

Impact Multiple definitions could become out-of-date over time.

Recommendation Define one common interface/contract with intended functions and use inheritance to share these functionalities.

Developer Response To avoid significant contract changes, developers have elected not to make this change at this point.

4.1.10 V-VUL-SAT-010: Possible lost funds on deposit

Severity	Warning	**Commit**	n/a
Type	Data Validation	**Status**	Acknowledged
File(s)	`DwBiz.sol`		
Location(s)	depositCoin		

The function `depositCoin` allows users to deposit both ETH and ERC20 tokens. The function is `payable`, which means that ETH can be sent along with the transaction. In the case that the user is depositing ERC20, if some ETH is attached to the transaction, it will just be lost to the contract.

Impact Naive users could lose some of their funds

Recommendation Add a requirement to the else branch within `depositCoin` which ensures that the `msg.value` is equal to `0`.

Developer Response Developers consider such a mistake on users and thus are not concerned with addressing the issue at this time.

4.1.11 V-VUL-SAT-011: Bad error reporting text

Severity	Info	**Commit**	n/a
Type	Maintainability	**Status**	Fixed
File(s)	`DwBiz.sol`		
Location(s)	authorizedWithdrawCoin		

The following require statement has a misleading error text as it references “deposit amount” despite amounts are actually being withdrawn:

```
require(_amount > 0 && _amount >= minWithdrawAmount[_coinAddr], "Deposit amount is
    too low");
```

Recommendation Change to “Withdraw amount is too low”.

4.1.12 V-VUL-SAT-012: Redundant check

Severity	Info	**Commit**	n/a
Type	Gas Optimization	**Status**	Fixed
File(s)	CommonManger.sol		
Location(s)	_init		

The following two checks occur in `_init`:

```
require(_signers.length >= _signThreshold && _signThreshold >= 2, "Params error");
require(_signThreshold!=0, "SignThreshold error");
```

The second check is redundant because of the first.

Recommendation Remove the second check.