# SECURITY REVIEW OF FIJA FINANCE

## Summary

**Auditor:** 0xWeiss (Marc Weiss)

**Client:** Fija Finance

**Report Delivered:** September ,2023

## Protocol Summary

| | |
|---|---|
| Protocol Name | Fija Finance |
| Language | Solidity |
| Codebase | |
| Commit | |
| Previous Audits | Yes, 2 previous audits from Chainsulting |

## About 0xWeiss

Marc Weiss, or **0xWeiss**, is an independent smart contract security researcher. Having found numerous security vulnerabilities in various protocols, he does his best to contribute to the blockchain ecosystem and its protocols by putting time and effort into security research & reviews. Reach out on Twitter @0xWeisss or on Telegram @0xWeiss.

## Audit Summary

Fija Finance engaged 0xWeiss to review the security of its defi protocol. From the 12th of August to the 19th of August, 0xWeiss reviewed the source code in scope. At the end, there were 12 issues identified. All findings have been recorded in the following report. Notice that the examined smart contracts are not resistant to internal exploit. For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

## Vulnerability Summary

| Severity | Total | Pending | Acknowledged | Par. resolved | Resolved |
|---|---|---|---|---|---|
| CRITICAL | 0 | 0 | 0 | 0 | 0 |
| HIGH | 1 | 0 | 0 | 0 | 0 |
| MEDIUM | 5 | 0 | 0 | 0 | 0 |
| LOW | 1 | 0 | 0 | 0 | 0 |
| INF | 5 | 0 | 0 | 0 | 0 |

## Audit Scope

| ID | File Path |
|---|---|
| VAULT | contracts\base\FijaVault.sol |
| STRA | contracts\base\FijaStrategy.sol |
| ACL | contracts\base\FijaACL.sol |
| ERR | contracts\base\errors.sol |

## Severity Classification

| Severity | Classification |
|---|---|
| CRITICAL | Easily exploitable by anyone, causing loss/manipulation of assets or data. |
| HIGH | Arduously exploitable, causing loss/manipulation of assets or data. |
| MEDIUM | Risk of future exploits that may or may not impact the smart contract execution. |
| LOW | Minor code errors that may or may not impact the smart contract execution. |
| INF | No impact issues. Code improvement |

## Methodology

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

## Findings and Resolutions

| ID | Category | Severity | Status |
|---|---|---|---|
| **VAULT-1** | Logical Error | **HIGH** | Resolved |
| **VAULT-2** | Permissioning | **MEDIUM** | Resolved |
| **VAULT-3** | Locked Funds | **MEDIUM** | Resolved |
| **VAULT-4** | Logical Error | **MEDIUM** | Resolved |
| **VAULT-5** | Locked Funds | **MEDIUM** | Resolved |
| **VAULT-6** | Typo | **INF** | Resolved |
| **STRA-1** | Logical Error | **MEDIUM** | Resolved |
| **STRA-2** | Typo | **INF** | Resolved |
| **ACL-1** | Centralization /Privilege | **LOW** | Resolved |
| **GLOBAL-1** | Centralization /Privilege | **INF** | Resolved |
| **GLOBAL-2** | Design Error | **INF** | Resolved |
| **GLOBAL-3** | Typo | **INF** | Resolved |

# VAULT-1 | Unwhitelisted users will lose their funds.

| Severity | Category | Status |
|---|---|---|
| HIGH | Logical Error | Resolved |

## Description of the issue

The current permissions of FIJA allows resellers to whitelist and remove from the whitelist any address they want. This is most likely to be a separate fetaure in the reseller's application. As an example, If a user enters the "PRO" mode of their app, they will unlock the ability to invest in Fija's strategies. But before they should get their wallet whitelisted by said reseller.

Accounts can be unwhitelisted for a set of reasons. This can be external to FIJA, like breaking the terms of service from the reseller's platform.

The whitelist is enforced thourgh all the codebase, including the `withdraw` and `redeem` functions from the strategy vault.

```
function withdraw(
        uint256 assets,
        address receiver,
        address owner
    )
        public
        virtual
        override(ERC4626, IERC4626)
        onlyWhitelisted    <---- It requires the user to still be
whitelisted
        onlyReceiverOwnerWhitelisted(receiver, owner)
        returns (uint256)
    {}
```

Any user that is removed from the whitelist in the future will not be able to withdraw his funds, losing their entire investment.

## Recommendation

Remove the whitelist requirement for withdrawals and redeems, as users already had the first layer of whitelist check while depositing.

## Resolution

## VAULT-2 | Un-whitelisted user for a specific strategy would still be able to deposit due to an error in the whitelist validation.

| Severity | Category | Status |
|---|---|---|
| MEDIUM | Permissioning | Resolved |

### Description of the issue

When depositing tokens to the vault, a user first deposits the funds in the vault

```
uint256 tokens = super.deposit(assets, receiver);
```

and then the deposited funds are sent to the strategy contract.

```
_strategy.deposit{value: allAssets}(allAssets, address(this));
```

As you can see, the first deposit function in the FijaVault enforces msg.sender and the receiver to be whitelisted.

```
function deposit(
        uint256 assets,
        address receiver
    )
        public
        payable
        virtual
        override(ERC4626, IERC4626)
        onlyWhitelisted
        onlyReceiverWhitelisted(receiver)
        returns (uint256);
```

Meanwhile in the second deposit, this time to the strategy contract, even though they are using the same base contract `FijaERC4626Base`. The `onlyWhitelisted` modifier checks that msg.sender is whitelisted. In this case is the `fijaVault` contract is the actual msg.sender and not the end user.

As both whitelists are separate, this could allow a whitelisted user in `fijaVault` to deposit funds into a strategy where he is not whitelisted.

### Recommendation

Validate that the end user is also included in the whitelist of the strategy contract.

### Resolution

# VAULT-3 | Funds can be permanently stuck in the vault when updating the strategy.

| Severity | Category | Status |
|---|---|---|
| MEDIUM | Locked Funds | Resolved |

## Description of the issue

When the `fijaVault` contract is deployed, the strategy contract address has to be provided in the constructor. Consecuently, then it checkes that the asset from the strategy is the same as the asset from the fija vault. This has to be checked because there are several spots on the vault contract were this is assumed, so it has to hold true on the future.

When updating the strategy from the updateStrategy() function, a strategy that `strategy_.asset() != asset()` can be accepted because this check is not enforced.

The following scenario has to happen for this to be given:

- Fija Vault is deployed and initialized
- The first strategy gets updated through `updateStrategy()`
- As there is not check for the new strategy asset, it will not revert if a new strategy with a different asset than the vault is introduced through governance
- If the second strategy, also referred as the new one, is updated again for a third strategy, the assets of the second strategy will be forever locked in the vault as they will be redeemed: ` _strategy.redeem(redeemAmount, address(this), address(this));` and then, theoretically should be sent to the new strategy. But, they will never be sent because it approves the asset() from the vault, not the one form the strategy:

```
if (asset() != ETH) {
            SafeERC20.forceApprove(
                IERC20(asset()),
                address(_strategy),
                totalAssetsInVault); }
```

## Recommendation

Do enforce the same check of:

```
if (strategy_.asset() != asset()) {
            revert VaultNoAssetMatching();
        }
```

when updating the strategy.

## Resolution

# VAULT-4 | Rewards will be left unclaimed when updating the strategy.

| Severity | Category | Status |
|---|---|---|
| MEDIUM | Logical Error | Resolved |

## Description of the issue

When updating the strategy, the `fijaVault` contract will redeem the funds from the old strategy and send them to the new one. This is done through the `updateStrategy()` function. The problem is that the strategy has not taken out profits before being updated to the new one. This means that the profits will be left in the old strategy and will not be claimed by the new one.

## Recommendation

Call the harvest function on the strategy before updating it.

```
_strategy.harvest();
// get assets back from strategy in batches
uint256 remainingTokens = _strategy.balanceOf(address(this));
```

## Resolution

# VAULT-5 | If an updated strategy redeems more than twice the MAX_TICKET_SIZE, funds will be stuck in the vault.

| Severity | Category | Status |
|---|---|---|
| MEDIUM | Locked Funds | Resolved |

## Description of the issue

Currently you can update a strategy from the main Vault, in the function ` updateStrategy()`.

This function redeems all the funds from the old strategy and sends them to the newly deployed one.
Currently on the vaults, it exists the enforcement of `MAX_TICKET_SIZE` as the maximum deposit amount that you can deposit at once, only in the deposit function, not on the redeem function.
The following scenario can be given.

- `MAX_TICKET_SIZE` = 200 * 10e26;  //sample value
- ` updateStrategy()` is called
- 401 * 10e26 is redeemed from the old strategy to the vault
- Vault tries to send all the funds to the new strategy, but ` MAX_TICKET_SIZE` is enforced, so only 200 * 10e26 are sent.
- There is 201 * 10e26 of the token/asset remaining on the vault.
- There is no way of claiming these 201 * 10e26 tokens back. Because they can't be deposited again to the new strategy as the vault pulls the whole contract balance to deposit to the strategy. As the requirement of `MAX_TICKET_SIZE` exists, funds will be stuck in the vault.

## Recommendation

Either allow ` MAX_TICKET_SIZE` to not be immutable and add a setter function for it to be adjusted if this happens or add a permissioned function to sweep assets from the main vault to the strategy.

## Resolution

# VAULT-6 | Confusing assignement to proposedTime

| Severity | Category | Status |
|---|---|---|
| INFORMATIONAL | Typo | Resolved |

## Description of the issue

When resetting the proposedTime while updating the startegy from the vault contracts, this `proposedTime` is being set to 6000000000, which makes no sense (it also has no impact).

```
_strategyCandidate.proposedTime = 6000000000;
```

## Recommendation

Consider just making it 0 not 6000000000.

## Resolution

## STRA-1 | Emergency mode is not enforced correctly, allowing users to withdraw, redeem and update the strategy.

| Severity | Category | Status |
|---|---|---|
| MEDIUM | Logical Error | Resolved |

### Description of the issue

Currently, strategy contracts do have an emergyMode feature called `emergencyModeRestriction`. This serves the purpose, as an example, of pausing the strategy contract if a security issue is discovered or if the protocol wants to stop an ongoing attack.

While `emergencyModeRestriction` is enforced in some functions, it lacks to be enforced in the most important ones, where you would mostly like the functionality to be paused. This functions are `withdraw` and `redeem`. If there is an ongoing attack, attackers would still be able to withdraw the stolen funds due to the lack of enforcement of `emergencyModeRestriction` in both functions.

Apart from it, the strategy should not be able to be upgraded while on emergency mode. Currently, this feature exists on the `fijaVault` `updateStrategy()` function, where a strategy can be updated even on emergency mode.

### Recommendation

Do add the `emergencyModeRestriction` modifier in `withdraw` and `redeem`

### Resolution

## STRA-2 | Typographical Issues

| Severity | Category | Status |
|---|---|---|
| INFORMATIONAL | Typo | Resolved |

### Description of the issue

There is no need to initialize booleans to false as their default values are already false.

```
bool internal _isEmergencyMode = false;
```

### Recommendation

Do not assign false to an uninitialized boolean.

### Resolution

## ACL-1 | Use a 2-step ownership transfer.

| Severity | Category | Status |
|---|---|---|
| LOW | Centralization /Privilege | Resolved |

### Description of the issue

### Recommendation

Fija uses a single-step access control transfer pattern. This means that if the current owner account transfers ownership with an incorrect address, then this owner role will be lost forever along with all the functionality that depends on it.

### Resolution

Follow the pattern from OpenZeppelin's Ownable2Step and implement a two-step transfer pattern for the action.

## GLOBAL-1 | FIJA is a highly permissioned protocol.

| Severity | Category | Status |
|---|---|---|
| Informational | Centralization /Privilege | Resolved |

### Description of the issue

The whole codebase is very restrictive in what users can do and cannot do with the contracts. There are also several roles inside the codebase that oversee key state changes that could affect users.

### Recommendation

All the roles should be managed by multisig wallets with KYC'ed participants.

### Resolution

## GLOBAL-2 | Project does not work with transfer-tax tokens.

| Severity | Category | Status |
|---|---|---|
| Informational | Design Error | Resolved |

### Description of the issue

The whole architecture has several spots which will not work with transfer-tax tokens, this will break the whole system.

### Recommendation

Consider not using such tokens.

### Resolution

## GLOBAL-3 | `uint` and `uint256` is used across contracts.

| Severity | Category | Status |
|---|---|---|
| Informational | Typo | Resolved |

### Description of the issue

Throughout the contracts, the `uint` and `uint256` types are used inconsistently.

### Recommendation

Consider using only `uint256` for a better readability of the codebase.

### Resolution

## DISCLAIMER

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Marc Weiss to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk.

My position is that each company and individual are responsible for their own due diligence and continuous security. My goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze. Therefore, I do not guarantee the explicit security of the audited smart contract, regardless of the verdict.