Security Audit Report for Fiat24 Contracts

Date: August 6, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Mantle
Target	Fiat24 Contracts

Version History

Version	Date	Description
1.0	August 6, 2025	First release

Signature

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the code repository [1] of Fiat24 Contracts of Mantle.

Fiat24 is a digital banking platform built on blockchain technology that bridges banking services with the crypto ecosystem. The platform provides NFT-based digital accounts as unique identifiers for users, with each account represented as an ERC-721 token featuring customizable features and status management. Fiat24 supports multiple fiat currencies through tokenized representations, including USD24, EUR24, CHF24, GBP24, and CNH24, with real-time exchange rates and seamless cross-currency transactions. The platform features crypto deposit functionality that enables users to deposit USDC and other cryptocurrencies, automatically converting them to fiat tokens at current market rates.

Note this audit only focuses on the smart contracts in the following directories/files:

- fiat24contracts/src/Fiat24CryptoDeposit.sol
- fiat24contracts/src/Fiat24CryptoDeposit2.sol
- fiat24contracts/src/Fiat24CryptoDeposit_Base.sol
- fiat24contracts/src/Fiat24CardAuthorizationMargeta.sol
- fiat24contracts/src/Fiat24CryptoRelay.sol
- fiat24contracts/src/FiatTokenBeacon.sol
- fiat24contracts/src/FiatTokenFactory.sol

Other files are not within the scope of the audit. Additionally, all dependencies of the smart contracts within the audit scope are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
Fiat24 Contracts	Version 1	32b66f10a42b9ba39de279312754160d20d2100d
	Version 2	8fa9f76352a27f901c293552ed1c03b06c9bb3f4

[1] https://github.com/mantle-xyz/fiat24contracts

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section **??**. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

- Access control
- Permission management
- Whitelist and blacklist mechanisms
- Initialization consistency
- Improper use of the proxy system
- Reentrancy
- Denial of Service (DoS)
- Untrusted external call and control flow
- Exception handling
- Data handling and flow
- Events operation

- Error-prone randomness
- Oracle security
- Business logic correctness
- Semantic and functional consistency
- Emergency mechanism
- Economic and incentive impact

1.3.2 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table **??**.

Table 1.1: Vulnerability Severity Classification

Impact \ **Likelihood**	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we found **nine** potential security issues. Besides, we have **six** recommendations and **seven** notes.

- High Risk: 1
- Medium Risk: 2
- Low Risk: 6
- Recommendation: 6
- Note: 7

ID	Severity	Description	Category	Status
1	High	Incorrect permission check in function `updateExchangeRate()`	Security Issue	Fixed
2	Medium	Potential front-running attacks when updating exchange rates	Security Issue	Confirmed
3	Medium	Fixed exchange rates during initialization creates front-running risk	Security Issue	Confirmed
4	Low	Potential DoS risk in the function `_removeFailedKey()`	Security Issue	Fixed
5	Low	Inconsistent mechanism of updating exchange rates	Security Issue	Confirmed
6	Low	Incorrect rounding direction in the functions `authorize()` and `increment()`	Security Issue	Confirmed
7	Low	Lack of checks for the parameters `cardCurrency_` and `originalPaidCurrency_`	Security Issue	Confirmed
8	Low	`Fiat24` tokens received by a `fiat24account` with specific `status` will be locked	Security Issue	Confirmed
9	Low	Inconsistent access control	Security Issue	Fixed
10	-	Inconsistency between the comment and the codes	Recommendation	Fixed
11	-	Lack of duplication check on the `fiatName` in the function `addFiatToken()`	Recommendation	Fixed
12	-	Add zero address checks	Recommendation	Confirmed
13	-	Lack of duplication check in the function `addTokenAddress()`	Recommendation	Confirmed
14	-	Confusing naming for the variable `_amountOutMinimum`	Recommendation	Confirmed
15	-	Lack of non zero value check in the function `updateExchangeRates()`	Recommendation	Confirmed

16	-	Atomicity in Fiat24 card authorization process	Note	-
17	-	Lack of fiat tokens removal mechanism	Note	-
18	-	The parameter _amountOutMinimum should be validated in the backend	Note	-
19	-	Upgrade the implementation of Fiat24Token properly	Note	-
20	-	Ensure that the exchangeRates and validXXX24Tokens are set properly	Note	-
21	-	Initialize the implementation contracts immediately after deployments	Note	-
22	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Incorrect permission check in function updateExchangeRate()

Severity High

Status Fixed in Version 2

Introduced by Version 1

Description Since the permission check for msg.sender is implemented in the function _updateExchangeRate() in the contract Fiat24CardAuthorizationMarqeta, a malicious attacker could exploit this by passing empty arrays (fiatTokens and rates with length 0) to the function updateExchangeRates(). This circumvents the authorization logic in the function _updateExchangeRate() while still allowing the attacker to modify the marketClosed value arbitrarily. Since the variable marketClosed affects spread calculations, this could ultimately lead to potential financial loss. The same problem exists in the contract Fiat24CryptoRelay.

```
function updateExchangeRates(
    address[] calldata fiatTokens,
    uint256[] calldata rates,
    bool isMarketClosed
) external {
    require(fiatTokens.length == rates.length, "Arrays length mismatch");
    marketClosed = isMarketClosed;
    for (uint256 i = 0; i < fiatTokens.length; i++) {
        address token = fiatTokens[i];
        uint256 rate = rates[i];
        require(validXXX24Tokens[token], "Invalid token");
        require(rate > 0, "Rate must be >0");
        _updateExchangeRate(token, rate, isMarketClosed);
    }
}
```

```

/// @notice Updating the exchange rate between USD and individual fiat currencies
function _updateExchangeRate(address _fiatToken, uint256 _rateUsdcToFiat, bool _isMarketClosed) internal {

    uint256 oldRate = exchangeRates[usd24Address][_fiatToken];

    if (hasRole(RATES_UPDATER_OPERATOR_ROLE, _msgSender())) {
        exchangeRates[usd24Address][_fiatToken] = _rateUsdcToFiat;
        emit ExchangeRateUpdatedByOperator(_fiatToken, oldRate, _rateUsdcToFiat, _isMarketClosed);
    } else if (hasRole(RATES_UPDATER_ROBOT_ROLE, _msgSender())) {

        uint256 rateDiff = oldRate > _rateUsdcToFiat ? (oldRate - _rateUsdcToFiat) : (_rateUsdcToFiat - oldRate);
        rateDiff = rateDiff * 10000 / oldRate;
        require(rateDiff < 300, "Rate Update Robot: change too large");
        exchangeRates[usd24Address][_fiatToken] = _rateUsdcToFiat;
        emit ExchangeRateUpdatedByRobot(_fiatToken, oldRate, _rateUsdcToFiat, _isMarketClosed);
    } else {
        revert Fiat24CardAuthorizationMarqeta__NotRateUpdater((_msgSender()));
    }
}
```

Listing 2.1: src/Fiat24CardAuthorizationMarqeta.sol

Impact This could ultimately lead to potential financial loss.

Suggestion Add a permission check in the function updateExchangeRates().

2.1.2 Potential front-running attacks when updating exchange rates

Severity Medium

Status Confirmed

Introduced by Version 1

Description In both the contracts Fiat24CardAuthorizationMarqeta and Fiat24CryptoRelay, exchange rate updates performed by the RATES_UPDATER_OPERATOR_ROLE and RATES_UPDATER_ROBOT_ROLE are vulnerable to front-running attacks. A malicious user could:

1. Monitor pending rate update transactions in the mempool.

2. Front-run the update by executing advantageous trades. For example, when the exchange rate rises, the user could front-run to exchange for cheaper fiatTokens.

```
function updateExchangeRates(
    address[] calldata fiatTokens,
    uint256[] calldata rates,
    bool isMarketClosed
) external {
    require(fiatTokens.length == rates.length, "Arrays length mismatch");
    marketClosed = isMarketClosed;
    for (uint256 i = 0; i < fiatTokens.length; i++) {
        address token = fiatTokens[i];
```

```
            uint256 rate = rates[i];
            require(validXXX24Tokens[token], "Invalid token");
            require(rate > 0, "Rate must be >0");
            _updateExchangeRate(token, rate, isMarketClosed);
        }
    }
```

Listing 2.2: src/Fiat24CardAuthorizationMarqeta.sol

Impact A malicious user could front-run to exchange for cheaper `fiatTokens`.

Suggestion Revise the logic accordingly.

Feedback from the project The project states that an exchange fee is charged during the exchange process, and there is a slight possibility of benefiting from front-running. Furthermore, the project states that the exchanges or spending operations for users are limited.

2.1.3 Fixed exchange rates during initialization creates front-running risk

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description In the contracts `Fiat24CardAuthorizationMarqeta`, the `initialize()` function sets hardcoded exchange rates for `Fiat24` tokens (e.g., `EUR24`, `USD24`, `CHF24`). These rates are applied immediately upon deployment and initialization, before any dynamic updates can occur. This introduces a front-running risk:

1.A malicious actor could monitor the contract deployment and initialization, and immediately execute trades at the fixed rates before the protocol updates them.

2.Since the initial rates may not reflect real-time market prices, attackers could arbitrarily profit by exploiting mispriced conversions (e.g., buying undervalued tokens or selling overvalued ones).

This could lead to protocol losses if the initial rates are significantly off-market. The contracts `Fiat24CryptoDeposit` and `Fiat24CryptoRelay` have the same problem.

```
        exchangeRates[usd24Address][usd24Address] = 10000;
        exchangeRates[usd24Address][eur24Address] = 9168;
        exchangeRates[usd24Address][chf24Address] = 8632;
        exchangeRates[usd24Address][gbp24Address] = 7674;
        exchangeRates[usd24Address][cnh24Address] = 70885;
```

Listing 2.3: src/Fiat24CardAuthorizationMarqeta.sol

Impact This could cause a loss to the protocol.

Suggestion Revise the code logic accordingly.

Feedback from the project The project states that the rate used in the deployment is up to date at that time.

2.1.4 Potential DoS risk in the function `_removeFailedKey()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `Fiat24CryptoRelay`, the function `_lzReceive()` attempts to process messages from `LayerZero` and append the failed `messageId` to the state variable `failedKeys`. The only way to remove a `messageId` is by invoking the function `_removeFailedKey()`, which will attempt to iterate through the entire `failedKeys` array. A malicious actor could create numerous failing messages, potentially causing the array iteration to exceed the block gas limit and eventually resulting in DoS in the functions `retryFailedMessage()` and `adminProcessFailedMessage()`.

```
function _removeFailedKey(bytes32 messageId) internal {
    uint256 len = failedKeys.length;
    for (uint256 i = 0; i < len; i++) {
        if (failedKeys[i] == messageId) {
            if (i < len - 1) {
                failedKeys[i] = failedKeys[len - 1];
            }
            failedKeys.pop();
            break;
        }
    }
}
```

Listing 2.4: src/Fiat24CryptoRelay.sol

```
function adminProcessFailedMessage(bytes32 messageId) external {
    if (!hasRole(OPERATOR_ROLE, _msgSender())) revert Fiat24CryptoDeposit__NotOperator(
        _msgSender());
    bytes memory payload = _failedPayloads[messageId];
    require(payload.length > 0, "No failed message to retry");
    delete _failedPayloads[messageId];
    _removeFailedKey(messageId);
    emit FailedMessageProcessed(messageId);
}
```

Listing 2.5: src/Fiat24CryptoRelay.sol

```
function retryFailedMessage(bytes32 messageId) external {
    bytes memory payload = _failedPayloads[messageId];
    require(payload.length > 0, "No failed message to retry");

    delete _failedPayloads[messageId];

    try this.processMessage(payload) {
        _removeFailedKey(messageId);
        emit MessageRetried(messageId, true, "");
    } catch Error(string memory reason) {
        _failedPayloads[messageId] = payload;
        emit MessageRetried(messageId, false, reason);
    } catch {
        _failedPayloads[messageId] = payload;
        emit MessageRetried(messageId, false, "Unknown failure");
    }
}
```

Listing 2.6: src/Fiat24CryptoRelay.sol

Impact This may cause a potential DoS risk.

Suggestion Implement a mechanism that allows partial processing of `failedKeys` instead of requiring a full traversal.

2.1.5 Inconsistent mechanism of updating exchange rates

Severity Low

Status Confirmed

Introduced by `Version 1`

Description The contract `Fiat24CardAuthorizationMarqeta` currently implements two different mechanisms for batch updating exchange rates. The first approach updates individual rates that meet the `rateDiff < 300` requirement while skipping non-compliant ones, and always updates the variable `marketClosed`. The second approach rejects the entire batch update if any single rate fails the `rateDiff < 300` check, including preventing the `marketClosed` update. The contract `Fiat24CryptoRelay` has the same problem. This could create unpredictable system behavior depending on which update mechanism is triggered.

```
    } else if ((hasRole(RATES_UPDATER_ROBOT_ROLE, _msgSender()))) {
        uint256 rateDiff_usd_eur = (exchangeRates[usd24Address][eur24Address] > _usd_eur)
            ? (exchangeRates[usd24Address][eur24Address] - _usd_eur)
            : (_usd_eur - exchangeRates[usd24Address][eur24Address]);
        rateDiff_usd_eur = (rateDiff_usd_eur * 10000) / exchangeRates[usd24Address][eur24Address
            ];
        uint256 rateDiff_usd_chf = (exchangeRates[usd24Address][chf24Address] > _usd_chf)
            ? (exchangeRates[usd24Address][chf24Address] - _usd_chf)
            : (_usd_chf - exchangeRates[usd24Address][chf24Address]);
        rateDiff_usd_chf = (rateDiff_usd_chf * 10000) / exchangeRates[usd24Address][chf24Address
            ];
        uint256 rateDiff_usd_gbp = (exchangeRates[usd24Address][gbp24Address] > _usd_gbp)
            ? (exchangeRates[usd24Address][gbp24Address] - _usd_gbp)
            : (_usd_gbp - exchangeRates[usd24Address][gbp24Address]);
        rateDiff_usd_gbp = (rateDiff_usd_gbp * 10000) / exchangeRates[usd24Address][gbp24Address
            ];
        uint256 rateDiff_usd_cnh = (exchangeRates[usd24Address][cnh24Address] > _usd_cnh)
            ? (exchangeRates[usd24Address][cnh24Address] - _usd_cnh)
            : (_usd_cnh - exchangeRates[usd24Address][cnh24Address]);
        rateDiff_usd_cnh = (rateDiff_usd_cnh * 10000) / exchangeRates[usd24Address][cnh24Address
            ];
        if (rateDiff_usd_eur < 300) exchangeRates[usd24Address][eur24Address] = _usd_eur;
        if (rateDiff_usd_chf < 300) exchangeRates[usd24Address][chf24Address] = _usd_chf;
        if (rateDiff_usd_gbp < 300) exchangeRates[usd24Address][gbp24Address] = _usd_gbp;
        if (rateDiff_usd_cnh < 300) exchangeRates[usd24Address][cnh24Address] = _usd_cnh;
        marketClosed = _isMarketClosed;
        emit ExchangeRatesUpdatedByRobot(
            _msgSender(),
            exchangeRates[usd24Address][eur24Address],
            exchangeRates[usd24Address][chf24Address],
            exchangeRates[usd24Address][gbp24Address],
            exchangeRates[usd24Address][cnh24Address],
            marketClosed
        );
    } else {
```

Listing 2.7: src/Fiat24CardAuthorizationMarqeta.sol

```
    } else if (hasRole(RATES_UPDATER_ROBOT_ROLE, _msgSender())) {

        uint256 rateDiff = oldRate > _rateUsdcToFiat ? (oldRate - _rateUsdcToFiat) : (
            _rateUsdcToFiat - oldRate);
        rateDiff = rateDiff * 10000 / oldRate;
        require(rateDiff < 300, "Rate Update Robot: change too large");
        exchangeRates[usd24Address][_fiatToken] = _rateUsdcToFiat;
        emit ExchangeRateUpdatedByRobot(_fiatToken, oldRate, _rateUsdcToFiat, _isMarketClosed);
    } else {
```

Listing 2.8: src/Fiat24CardAuthorizationMarqeta.sol

Impact This could create unpredictable system behavior depending on which update mechanism is triggered.

Suggestion Uniform the two mechanisms.

Feedback from the project The project states that they will delete one of the two functions `updateExchangeRates()` in the future.

2.1.6 Incorrect rounding direction in the functions `authorize()` and `increment()`

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In the contract `Fiat24CardAuthorizationMarqeta`, the functions `authorize()` and `increment()` round down when calculating the value of `paidAmount`, which should be transferred from the user account to the `booked`. This could cause a loss to the protocol.

```
    if (validXXX24Tokens[XXX24Tokens[transactionCurrency_]]) {
        if (
            IERC20Upgradeable(XXX24Tokens[transactionCurrency_]).balanceOf(sender) >=
                transactionAmount_
                && IERC20Upgradeable(XXX24Tokens[transactionCurrency_]).allowance(sender, address
                    (this)) >= transactionAmount_
        ) {
            paidCurrency = XXX24Tokens[transactionCurrency_];
            paidAmount = transactionAmount_;
        } else {
            paidAmount = transactionAmount_ * getRate(XXX24Tokens[transactionCurrency_], cardCurrency_)
                * getSpread(XXX24Tokens[transactionCurrency_], cardCurrency_, false) / 100000000;
        }
    } else {
        if (settlementCurrency_ != eur24Address) revert Fiat24CardAuthorizationMarqeta__DefaultSettlementCurrencyIsNotEUR(settlementCurrency_);
        paidAmount =
            settlementAmount_ * (100 + interchange) * getRate(eur24Address, cardCurrency_) * getSpread(eur24Address, cardCurrency_, false) / 10000000000;
    }
```

Listing 2.9: src/Fiat24CardAuthorizationMarqeta.sol

```
        if (validXXX24Tokens[XXX24Tokens[transactionCurrency_]]) {
            if (
                IERC20Upgradeable(XXX24Tokens[transactionCurrency_]).balanceOf(sender) >= transactionAmount_
                    && IERC20Upgradeable(XXX24Tokens[transactionCurrency_]).allowance(sender, address(this)) >= transactionAmount_
            ) {
                paidCurrency = XXX24Tokens[transactionCurrency_];
                paidAmount = transactionAmount_;
            } else {
                paidCurrency = cardCurrency_;
                paidAmount = transactionAmount_ * getRate(XXX24Tokens[transactionCurrency_], cardCurrency_)
                    * getSpread(XXX24Tokens[transactionCurrency_], cardCurrency_, false) / 100000000;
            }
        } else {
            if (settlementCurrency_ != eur24Address) revert Fiat24CardAuthorizationMarqeta__DefaultSettlementCurrencyIsNotEUR(settlementCurrency_);
            paidCurrency = cardCurrency_;
            paidAmount =
                settlementAmount_ * (100 + interchange) * getRate(eur24Address, cardCurrency_) * getSpread(eur24Address, cardCurrency_, false) / 10000000000;
        }
```

Listing 2.10: src/Fiat24CardAuthorizationMarqeta.sol

Impact This could cause a loss to the protocol.

Suggestion Round up when calculating the value of the variable `paidAmount` in the functions `authorize()` and `increment()`.

2.1.7 Lack of checks for the parameters `cardCurrency_` and `originalPaidCurrency_`

Severity Low

Status Confirmed

Introduced by Version 1

Description The contract Fiat24CardAuthorizationMarqeta does not verify whether the input parameters cardCurrency_ and originalPaidCurrency_ have corresponding validXXX24Tokens values set to true. This oversight could allow processing of invalid currencies that are not registered in validXXX24Tokens, ultimately leading to exchange rate lookups returning zero values, which may cause a loss to the protocol.

```
function authorize(
    string memory authorizationToken_,
    string memory cardId_,
    uint256 tokenId_,
    address cardCurrency_,
    string memory transactionCurrency_,
    address settlementCurrency_,
    uint256 transactionAmount_,
    uint256 settlementAmount_
) public {
    if (!(hasRole(AUTHORIZER_ROLE, _msgSender()))) revert
         Fiat24CardAuthorizationMarqeta__NotAuthorizer(_msgSender());
    if (paused()) revert Fiat24CardAuthorizationMarqeta__Suspended();
    if (!validXXX24Tokens[settlementCurrency_]) revert
         Fiat24CardAuthorizationMarqeta__NotValidSettlementCurrency(settlementCurrency_);
    address sender = IFiat24Account(fiat24AccountAddress).ownerOf(tokenId_);
    address booked = IFiat24Account(fiat24AccountAddress).ownerOf(CARD_BOOKED);
    address paidCurrency = cardCurrency_;
```

Listing 2.11: src/Fiat24CardAuthorizationMarqeta.sol

```
function advice(
    string memory authorizationToken_,
    string memory originalAuthorizationToken_,
    string memory cardId_,
    uint256 tokenId_,
    string memory transactionCurrency_,
    address settlementCurrency_,
    uint256 transactionAmount_,
    uint256 settlementAmount_,
    address originalPaidCurrency_
) public {
    if (!(hasRole(AUTHORIZER_ROLE, _msgSender()))) revert
         Fiat24CardAuthorizationMarqeta__NotAuthorizer(_msgSender());
    if (paused()) revert Fiat24CardAuthorizationMarqeta__Suspended();
    if (!validXXX24Tokens[settlementCurrency_]) revert
         Fiat24CardAuthorizationMarqeta__NotValidSettlementCurrency(settlementCurrency_);
    address sender = IFiat24Account(fiat24AccountAddress).ownerOf(tokenId_);
    address booked = IFiat24Account(fiat24AccountAddress).ownerOf(CARD_BOOKED);
    address paidCurrency = originalPaidCurrency_; // Always pay back to the same currency
```

Listing 2.12: src/Fiat24CardAuthorizationMarqeta.sol

Impact This could cause a loss to the protocol.

Suggestion Add validations for the parameters cardCurrency_ and originalPaidCurrency_.

Feedback from the project The project states that they will fix this issue in a future version.

2.1.8 Fiat24 tokens received by a fiat24account with specific status will be locked

Severity Low

Status Confirmed

Introduced by Version 1

Description In the contract Fiat24Token, the function tokenTransferAllowed() allows a fiat24account whose status is either Na or Tourist to receive Fiat24 tokens. However, due to the tokenTransferAllowed() check, these accounts are unable to use the received tokens. For example, if the users want to pay out by invoking the function clientPayout(), they will fail, since only the accounts with the status.Live can transfer Fiat24 tokens to other addresses. As a result, the Fiat24 tokens held by the accounts whose status is either Na or Tourist become locked.

```
function tokenTransferAllowed(address from, address to, uint256 amount) public view returns (
    bool) {
    require(!fiat24account.paused(), "Fiat24Token: All account transfers are paused");
    require(!paused(), "Fiat24Token: All account transfers of this currency are paused");
    if (sanctionCheck) {
        SanctionsList sanctionsList = SanctionsList(sanctionContract);
        bool toIsSanctioned = sanctionsList.isSanctioned(to);
        require(!toIsSanctioned, "Fiat24Token: Transfer to sanctioned address");
        bool fromIsSanctioned = sanctionsList.isSanctioned(from);
        require(!fromIsSanctioned, "Fiat24Token: Transfer from sanctioned address");
    }
    if (from != address(0) && to != address(0)) {
        if (balanceOf(from) < amount) {
            return false;
        }
        uint256 toAmount = amount + balanceOf(to);
        Fiat24Account.Status fromClientStatus;
        uint256 accountIdFrom = fiat24account.historicOwnership(from);
        if (accountIdFrom != 0) {
            fromClientStatus = fiat24account.status(accountIdFrom);
        } else if (from != address(0) && fiat24account.balanceOf(from) > 0) {
            fromClientStatus = Fiat24Account.Status.Tourist;
            accountIdFrom = fiat24account.tokenOfOwnerByIndex(from, 0);
        } else {
            fromClientStatus = Fiat24Account.Status.Na;
        }
        Fiat24Account.Status toClientStatus;
        uint256 accountIdTo = fiat24account.historicOwnership(to);
        if (accountIdTo != 0) {
            toClientStatus = fiat24account.status(accountIdTo);
        } else if (to != address(0) && fiat24account.balanceOf(to) > 0) {
            toClientStatus = Fiat24Account.Status.Tourist;
            accountIdTo = fiat24account.tokenOfOwnerByIndex(to, 0);
        } else {
            toClientStatus = Fiat24Account.Status.Na;
        }
        uint256 amountInChf = convertToChf(amount);
        bool fromLimitCheck = fiat24account.checkLimit(accountIdFrom, amountInChf);
        bool toLimitCheck = fiat24account.checkLimit(accountIdTo, amountInChf);
        // When the money from 91xx, we don't consider the client limit
        if (accountIdFrom >= 9100 && accountIdFrom <= 9199) {
            toLimitCheck = true;
        }
        return (
            fromClientStatus == Fiat24Account.Status.Live
                && (toClientStatus == Fiat24Account.Status.Live || toClientStatus == Fiat24Account.Status.SoftBlocked) && fromLimitCheck && toLimitCheck
        )
            || (
                fromClientStatus == Fiat24Account.Status.Live && fromLimitCheck
                    && ((toClientStatus == Fiat24Account.Status.Na || toClientStatus == Fiat24Account.Status.Tourist) && toAmount <= LimitWalkin)
            );
    }
    return false;
}
```

Listing 2.13: src/Fiat24Token.sol

```
function clientPayout(uint256 amount, string memory contactId) external {
    require(amount >= minimalPayoutAmount, "Fiat24Token: amount < minimal payout amount");
    uint256 tokenId = fiat24account.tokenOfOwnerByIndex(msg.sender, 0);
    // string memory txid = string(abi.encodePacked(uintToString(tokenId), "-", uintToString(ArbSys(address(100)).arbBlockNumber())));
    string memory txid = string(abi.encodePacked(uintToString(tokenId), "-", uintToString(block.number)));
    transferByAccountId(9102, amount);
    emit ClientPayout(tokenId, msg.sender, 9102, amount, contactId, txid);
}
```

Listing 2.14: src/Fiat24Token.sol

Impact Fiat tokens held by the accounts whose `status` is either `Na` or `Tourist` are locked.

Suggestion Revise the code logic accordingly.

Feedback from the project The project states that FiatToken will only be unlocked for users who have successfully passed the KYC verification process.

2.1.9 Inconsistent access control

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `Fiat24CryptoDeposit_Base`, the functions `changeUsdcAddress()` and `changeUsdcDepositAddress()` are executed by the role `OPERATOR_ADMIN_ROLE`. However, in the contract Fiat24CryptoDeposit2, the same functions are used by the role DEFAULT_ADMIN_ROLE. The difference in access control for the same function across contracts may lead to misoperations.

```
function changeUsdcAddress(address _usdcAddress) external {
    if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
        Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
    require(_usdcAddress != address(0), "Invalid usdc address");
    usdc = _usdcAddress;
}

function changeUsdcDepositAddress(address _usdcDepositAddress) external {
    if (!hasRole(DEFAULT_ADMIN_ROLE, _msgSender())) revert
        Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
    address oldUsdcDepositAddress = usdcDepositAddress;
    usdcDepositAddress = _usdcDepositAddress;
    emit UsdcDepositAddressChanged(oldUsdcDepositAddress, usdcDepositAddress);
}
```

Listing 2.15: src/Fiat24CryptoDeposit2.sol

```
function changeUsdcAddress(address _usdcAddress) external {
    if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
        Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
    require(_usdcAddress != address(0), "Invalid usdc address");
    usdc = _usdcAddress;
}

function changeUsdcDepositAddress(address _usdcDepositAddress) external {
    if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
        Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
    address oldUsdcDepositAddress = usdcDepositAddress;
    usdcDepositAddress = _usdcDepositAddress;
    emit UsdcDepositAddressChanged(oldUsdcDepositAddress, usdcDepositAddress);
}
```

Listing 2.16: src/Fiat24CryptoDeposit_Base.sol

Impact Potential misoperations due to the inconsistent access control.

Suggestion Revise the logic accordingly.

2.2 Recommendation

2.2.1 Inconsistency between the comment and the codes

Status Fixed in Version 2

Introduced by Version 1

Description In the contract Fiat24CryptoDeposit_Base, the comment for the function quoteLayerzeroFee() states, "Quotes the gas needed to pay for the full omnichain transaction in native gas or ZRO token." However, in the code implementation, when calling the function _quote(), the parameter _payInLzToken is set to false, indicating that the transaction fee is paid in only native gas, which is inconsistent with the description in the comment.

```
/**
 * @notice Quotes the gas needed to pay for the full omnichain transaction in native gas or ZRO token.
 */
function quoteLayerzeroFee(
    uint32 _dstEid,
    address _userAddress,
    address _inputToken,
    uint256 _inputAmount,
    uint256 _usdcAmount,
    address _outputToken
) public view returns (MessagingFee memory fee) {
    bytes memory payload = abi.encode(
        _userAddress,
        _inputToken,
        _inputAmount,
        _usdcAmount,
        _outputToken
    );

    bytes memory defaultWorkerOptions = OptionsBuilder
        .newOptions()
        .addExecutorLzReceiveOption(relay_gas_limit, 0);

    fee = _quote(_dstEid, payload, defaultWorkerOptions, false);
}
```

Listing 2.17: src/Fiat24CryptoDeposit_Base.sol

Suggestion Revise the code logic accordingly.

2.2.2 Lack of duplication check on the fiatName in the function addFiatToken()

Status Fixed in Version 2

Introduced by Version 1

Description In the contract Fiat24CardAuthorizationMarqeta, the function addFiatToken() does not verify whether the _fiatName string parameter has already been used in the XXX24Tokens mapping. This oversight could lead to accidental overwriting of existing token entries in the mapping. The vulnerability affects the paidCurrency = XXX24Tokens[transactionCurrency_] logic, potentially returning incorrect token addresses when looking up currencies by name. This could lead to financial loss to the protocol when there are misoperations.

```
function addFiatToken(address _fiatToken, uint256 _rateUsdToFiat, string calldata _fiatName) external {

    if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert Fiat24CardAuthorizationMarqeta__NotOperator(_msgSender());

    require(_fiatToken != address(0), "Zero address");
    require(!validXXX24Tokens[_fiatToken], "Already exists token");
    require(_rateUsdToFiat > 0, "Rate must be > 0");

    validXXX24Tokens[_fiatToken] = true;
    XXX24Tokens[_fiatName] = _fiatToken;
    exchangeRates[usd24Address][_fiatToken] = _rateUsdToFiat;

    emit FiatTokenAndRateAddedInMarqeta(_fiatToken, _rateUsdToFiat, _fiatName);
}
```

Listing 2.18: src/Fiat24CardAuthorizationMarqeta.sol

Impact This could lead to financial loss to the protocol when there are misoperations.

Suggestion Add duplicate checks accordingly.

2.2.3 Add zero address checks

Status Confirmed

Introduced by Version 1

Description In the function constructor() of the contract Fiat24CardAuthorizationMarqeta, several address variables (e.g., eur24Address_, usd24Address_, chf24Address_, gbp24Address_, cnh24Address_) are not checked to ensure they are not zero. Similar checks are also recommended to add for arrays fiatTokenOperatorRoles, cashOperatorRoles, and fiatTokenPausers in the contract Fiat24TokenFactory.

```
function initialize(
    address admin,
    address fiat24AccountAddress_,
    address eur24Address_,
    address usd24Address_,
    address chf24Address_,
    address gbp24Address_,
    address cnh24Address_
) public initializer {
    __AccessControl_init_unchained();
    __Pausable_init_unchained();
    _setupRole(DEFAULT_ADMIN_ROLE, admin);
    _setupRole(OPERATOR_ADMIN_ROLE, admin);
    fiat24AccountAddress = fiat24AccountAddress_;
    eur24Address = eur24Address_;
    usd24Address = usd24Address_;
    chf24Address = chf24Address_;
    gbp24Address = gbp24Address_;
    cnh24Address = cnh24Address_;
```

Listing 2.19: src/Fiat24CardAuthorizationMarqeta.sol

```
address[] public fiatTokenOperatorRoles;

address[] public cashOperatorRoles;
```

```

address[] public fiatTokenPausers;
```

Listing 2.20: src/FiatTokenFactory.sol

Suggestion Add zero address checks accordingly.

2.2.4 Lack of duplication check in the function `addTokenAddress()`

Status Confirmed

Introduced by `Version 1`

Description In the contract `FiatTokenFactory`, the function `addTokenAddress()` does not verify whether the `tokenAddress` already exists in the `allTokens` array before pushing the new address. This could lead to duplicate entries in the array.

```
function addTokenAddress(address tokenAddress) external onlyRole(DEFAULT_ADMIN_ROLE) {
    allTokens.push(tokenAddress);
    emit ConfigUpdated("addTokenAddress(address)", "", abi.encode(tokenAddress));
}
```

Listing 2.21: src/FiatTokenFactory.sol

Suggestion Add duplicate checks accordingly.

2.2.5 Confusing naming for the variable `_amountOutMinimum`

Status Confirmed

Introduced by `Version 1`

Description In the contract `Fiat24CryptoDeposit`, the functions `depositTokenViaUsdc()` and `permitAndDepositTokenViaUsdc()` allow users to deposit `USDC` in exchange for other `Fiat24` tokens. Both functions include a parameter named `_amountOutMinimum`, which is used to validate whether the input `_amount` (representing the `USDC` amount being deposited) meets a minimum requirement (`_amount < _amountOutMinimum`).

However, the current naming of `_amountOutMinimum` is misleading because it looks like this parameter represents the minimum expected output amount of `Fiat24` tokens, when in reality, it serves as the minimum required input amount of `USDC`. This inconsistency in naming could cause confusion for developers and users interacting with these functions. The contracts `Fiat24CryptoDeposit_Base`, `Fiat24CryptoDeposit2` have the same problem.

```
function depositTokenViaUsdc(address _inputToken, address _outputToken, uint256 _amount,
     uint256 _amountOutMinimum) nonReentrant external returns (uint256) {
    if (paused()) revert Fiat24CryptoDeposit__Paused();
    if (_amount < _amountOutMinimum || _amount == 0) revert
        Fiat24CryptoDeposit__AmountLessThanMinimum(_amount);
    if (_inputToken != usdc) revert Fiat24CryptoDeposit__NotValidInputToken(_inputToken);
    if (!validXXX24Tokens[_outputToken]) revert Fiat24CryptoDeposit__NotValidOutputToken(
        _outputToken);
    uint256 tokenId = IFiat24Account(fiat24account).historicOwnership(_msgSender());
    if (tokenId == 0) revert Fiat24CryptoDeposit__AddressHasNoToken(_msgSender());

    TransferHelper.safeTransferFrom(_inputToken, _msgSender(), address(this), _amount);
    return _processDeposit(_msgSender(), _inputToken, _outputToken, _amount, _amount, tokenId);
}
```

Listing 2.22: src/Fiat24CryptoDeposit.sol

```
function permitAndDepositTokenViaUsdc(
    address userAddress,
    address _inputToken,
    address _outputToken,
    uint256 _amount,
    uint256 _amountOutMinimum,
    uint256 _feeAmountViaUsdc,
    uint256 _deadline,
    uint8 _v,
    bytes32 _r,
    bytes32 _s
) external nonReentrant payable returns (uint256) {
    if (paused()) revert Fiat24CryptoDeposit__Paused();
    if (!hasRole(CASH_OPERATOR_ROLE, _msgSender())) revert Fiat24Token__NotCashOperator(_msgSender());
    if (_inputToken != usdc) revert Fiat24CryptoDeposit__NotValidInputToken(_inputToken);
    if (!validXXX24Tokens[_outputToken]) revert Fiat24CryptoDeposit__NotValidOutputToken(_outputToken);
    if (_amount < _amountOutMinimum || _amount == 0) revert Fiat24CryptoDeposit__AmountLessThanMinimum(_amount);

    try IERC20PermitUpgradeable(_inputToken).permit(
        userAddress,
        address(this),
        _amount,
        _deadline,
        _v, _r, _s
    ) {
    } catch {
        emit PermitFailed(userAddress, _inputToken, _amount);
    }

    uint256 tokenId = IFiat24Account(fiat24account).historicOwnership(userAddress);
    if (tokenId == 0) revert Fiat24CryptoDeposit__AddressHasNoToken(userAddress);

    TransferHelper.safeTransferFrom(_inputToken, userAddress, address(this), _amount);

    if (_feeAmountViaUsdc >= MAX_FEE_AMOUNT_USDC) {
        _feeAmountViaUsdc = MAX_FEE_AMOUNT_USDC;
    }

    if (_feeAmountViaUsdc >= _amount) {
        revert Fiat24CryptoDeposit__FeeAmountExceedsOutput(_feeAmountViaUsdc, _amount);
    }

    uint256 usdcFactAmount = _amount - _feeAmountViaUsdc;
    TransferHelper.safeTransfer(usdc, feeReceiver, _feeAmountViaUsdc);

    return _processDeposit(userAddress, _inputToken, _outputToken, _amount, usdcFactAmount, tokenId);
}
```

Listing 2.23: src/Fiat24CryptoDeposit.sol

Suggestion It is recommended to rename `_amountOutMinimum` to `_amountUsdcMinimum` or a similar name that accurately reflects its purpose.

2.2.6 Lack of non zero value check in the function `updateExchangeRates()`

Status Confirmed

Introduced by `Version 1`

Description In the contract `Fiat24CardAuthorizationMarqeta`, the function `updateExchangeRates()` does not validate whether the input rate values (i.e., `_usd_eur`, `_usd_chf`, `_usd_gbp`, `_usd_cnh`) are greater than zero before updating the exchange rates. The function `updateExchangeRates()` in the contract `Fiat24CryptoRelay` has the same problem.

```
function updateExchangeRates(uint256 _usd_eur, uint256 _usd_chf, uint256 _usd_gbp, uint256 _usd_cnh, bool _isMarketClosed) external {
    if (hasRole(RATES_UPDATER_OPERATOR_ROLE, _msgSender())) {
        exchangeRates[usd24Address][eur24Address] = _usd_eur;
        exchangeRates[usd24Address][chf24Address] = _usd_chf;
        exchangeRates[usd24Address][gbp24Address] = _usd_gbp;
        exchangeRates[usd24Address][cnh24Address] = _usd_cnh;
        marketClosed = _isMarketClosed;
        emit ExchangeRatesUpdatedByOperator(
            _msgSender(),
            exchangeRates[usd24Address][eur24Address],
            exchangeRates[usd24Address][chf24Address],
            exchangeRates[usd24Address][gbp24Address],
            exchangeRates[usd24Address][cnh24Address],
            marketClosed
        );
    } else if ((hasRole(RATES_UPDATER_ROBOT_ROLE, _msgSender()))) {
        uint256 rateDiff_usd_eur = (exchangeRates[usd24Address][eur24Address] > _usd_eur)
            ? (exchangeRates[usd24Address][eur24Address] - _usd_eur)
            : (_usd_eur - exchangeRates[usd24Address][eur24Address]);
        rateDiff_usd_eur = (rateDiff_usd_eur * 10000) / exchangeRates[usd24Address][eur24Address];
        uint256 rateDiff_usd_chf = (exchangeRates[usd24Address][chf24Address] > _usd_chf)
            ? (exchangeRates[usd24Address][chf24Address] - _usd_chf)
            : (_usd_chf - exchangeRates[usd24Address][chf24Address]);
        rateDiff_usd_chf = (rateDiff_usd_chf * 10000) / exchangeRates[usd24Address][chf24Address];
        uint256 rateDiff_usd_gbp = (exchangeRates[usd24Address][gbp24Address] > _usd_gbp)
            ? (exchangeRates[usd24Address][gbp24Address] - _usd_gbp)
            : (_usd_gbp - exchangeRates[usd24Address][gbp24Address]);
        rateDiff_usd_gbp = (rateDiff_usd_gbp * 10000) / exchangeRates[usd24Address][gbp24Address];
        uint256 rateDiff_usd_cnh = (exchangeRates[usd24Address][cnh24Address] > _usd_cnh)
            ? (exchangeRates[usd24Address][cnh24Address] - _usd_cnh)
            : (_usd_cnh - exchangeRates[usd24Address][cnh24Address]);
        rateDiff_usd_cnh = (rateDiff_usd_cnh * 10000) / exchangeRates[usd24Address][cnh24Address];
        if (rateDiff_usd_eur < 300) exchangeRates[usd24Address][eur24Address] = _usd_eur;
        if (rateDiff_usd_chf < 300) exchangeRates[usd24Address][chf24Address] = _usd_chf;
        if (rateDiff_usd_gbp < 300) exchangeRates[usd24Address][gbp24Address] = _usd_gbp;
        if (rateDiff_usd_cnh < 300) exchangeRates[usd24Address][cnh24Address] = _usd_cnh;
        marketClosed = _isMarketClosed;
        emit ExchangeRatesUpdatedByRobot(
            _msgSender(),
            exchangeRates[usd24Address][eur24Address],
            exchangeRates[usd24Address][chf24Address],
            exchangeRates[usd24Address][gbp24Address],
            exchangeRates[usd24Address][cnh24Address],
            marketClosed
        );
    } else {
        revert Fiat24CardAuthorizationMarqeta__NotRateUpdater((_msgSender()));
    }
}
```

Listing 2.24: src/Fiat24CardAuthorizationMarqeta.sol

```
function updateExchangeRates(uint256 _usd_eur, uint256 _usd_chf, uint256 _usd_gbp, uint256 _usd_cnh, bool _isMarketClosed) external {
    if (hasRole(RATES_UPDATER_OPERATOR_ROLE, _msgSender())) {
        exchangeRates[usd24][eur24] = _usd_eur;
        exchangeRates[usd24][chf24] = _usd_chf;
        exchangeRates[usd24][gbp24] = _usd_gbp;
        exchangeRates[usd24][cnh24] = _usd_cnh;
        marketClosed = _isMarketClosed;
        emit ExchangeRatesUpdatedByOperator(
            _msgSender(), exchangeRates[usd24][eur24], exchangeRates[usd24][chf24], exchangeRates[usd24][gbp24], exchangeRates[usd24][cnh24], marketClosed
        );
    } else if ((hasRole(RATES_UPDATER_ROBOT_ROLE, _msgSender()))) {
        uint256 rateDiff_usd_eur =
            (exchangeRates[usd24][eur24] > _usd_eur) ? (exchangeRates[usd24][eur24] - _usd_eur) : (_usd_eur - exchangeRates[usd24][eur24]);
        rateDiff_usd_eur = (rateDiff_usd_eur * XXX24_DIVISOR) / exchangeRates[usd24][eur24];
        uint256 rateDiff_usd_chf =
            (exchangeRates[usd24][chf24] > _usd_chf) ? (exchangeRates[usd24][chf24] - _usd_chf) : (_usd_chf - exchangeRates[usd24][chf24]);
        rateDiff_usd_chf = (rateDiff_usd_chf * XXX24_DIVISOR) / exchangeRates[usd24][chf24];
        uint256 rateDiff_usd_gbp =
            (exchangeRates[usd24][gbp24] > _usd_gbp) ? (exchangeRates[usd24][gbp24] - _usd_gbp) : (_usd_gbp - exchangeRates[usd24][gbp24]);
        rateDiff_usd_gbp = (rateDiff_usd_gbp * XXX24_DIVISOR) / exchangeRates[usd24][gbp24];
        uint256 rateDiff_usd_cnh =
            (exchangeRates[usd24][cnh24] > _usd_cnh) ? (exchangeRates[usd24][cnh24] - _usd_cnh) : (_usd_cnh - exchangeRates[usd24][cnh24]);
        rateDiff_usd_cnh = (rateDiff_usd_cnh * XXX24_DIVISOR) / exchangeRates[usd24][cnh24];
        if (rateDiff_usd_eur < 300) exchangeRates[usd24][eur24] = _usd_eur;
        if (rateDiff_usd_chf < 300) exchangeRates[usd24][chf24] = _usd_chf;
        if (rateDiff_usd_gbp < 300) exchangeRates[usd24][gbp24] = _usd_gbp;
        if (rateDiff_usd_cnh < 300) exchangeRates[usd24][cnh24] = _usd_cnh;
        marketClosed = _isMarketClosed;
        emit ExchangeRatesUpdatedByRobot(
            _msgSender(), exchangeRates[usd24][eur24], exchangeRates[usd24][chf24],
                exchangeRates[usd24][gbp24], exchangeRates[usd24][cnh24], marketClosed
        );
    } else {
        revert Fiat24CryptoDeposit__NotRateUpdater((_msgSender()));
    }
}
```

Listing 2.25: src/Fiat24CryptoRelay.sol

Impact This could lead to operational errors that update exchange rates to zero, which in subsequent currency conversions would cause financial loss to the protocol.

Suggestion Add non zero value checks accordingly.

2.3 Note

2.3.1 Atomicity in Fiat24 card authorization process

Introduced by Version 1

Description Our current assumption for the contract Fiat24CardAuthorizationMarqeta is as follows:

1.A user makes an offline payment using a physical card.

2.The card issuer forwards the transaction details to Fiat24's backend system.

3.An address with AUTHORIZER_ROLE subsequently deducts the corresponding amount from the user's Fiat24 account.

However, if the process is not atomic, the following risks may arise:

1.Double-spending attack

A user could swipe the card multiple times before the AUTHORIZER_ROLE executes the deduction. If the total spent exceeds their Fiat24 account balance, they could obtain goods without sufficient funds, profiting at the protocol's expense.

2.Exchange rate & interchange fee risks

The final paidAmount depends on dynamic factors like, fluctuating exchange rates at settlement time, and interchange fees based on the paidCurrency. If the calculated paidAmount exceeds the user's balance due to these variables, the deduction could fail after the goods are already taken, leaving the protocol with unrecoverable losses.

Thus, the project should ensure the atomicity in Fiat24 card authorization process.

Feedback from the project The project states that a pre-confirmation mechanism is implemented to ensure transactions are executed atomically to prevent the risks above.

2.3.2 Lack of fiat tokens removal mechanism

Introduced by Version 1

Description In the contracts Fiat24CardAuthorizationMarqeta, Fiat24CryptoDeposit_Base, Fiat24CryptoDeposit, Fiat24CryptoDeposit2, and Fiat24CryptoRelay, the function addFiatToken() is used to add new fiat tokens. They all lack a corresponding removal mechanism for cases when a particular fiat token needs to be discontinued or removed from support.

```
function addFiatToken(address _fiatToken, uint256 _rateUsdToFiat, string calldata _fiatName) external {

    if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert Fiat24CardAuthorizationMarqeta__NotOperator(_msgSender());

    require(_fiatToken != address(0), "Zero address");
    require(!validXXX24Tokens[_fiatToken], "Already exists token");
    require(_rateUsdToFiat > 0, "Rate must be > 0");

    validXXX24Tokens[_fiatToken] = true;
    XXX24Tokens[_fiatName] = _fiatToken;
    exchangeRates[usd24Address][_fiatToken] = _rateUsdToFiat;

    emit FiatTokenAndRateAddedInMarqeta(_fiatToken, _rateUsdToFiat, _fiatName);
}
```

Listing 2.26: src/Fiat24CardAuthorizationMarqeta.sol

Feedback from the project The project will add the removal mechanism in the future.

2.3.3 The parameter _amountOutMinimum should be validated in the backend

Introduced by Version 1

Description In the contracts Fiat24CryptoDeposit2 and Fiat24CryptoDeposit_Base, the function permitAndDepositTokenViaUsdc() is invoked by the CASH_OPERATOR_ROLE to control the _amountOutMinimum slippage parameter. This requires backend systems to strictly validate the parameter _amountOutMinimum. Otherwise, it may cause loss to users.

```
function permitAndDepositTokenViaUsdc(
    address userAddress,
    address _inputToken,
    address _outputToken,
    uint256 _amount,
    uint256 _amountOutMinimum,
    uint256 _feeAmountViaUsdc,
    uint256 _deadline,
    uint8 _v,
    bytes32 _r,
    bytes32 _s
) external nonReentrant payable returns (uint256) {
    if (paused()) revert Fiat24CryptoDeposit__Paused();
    if (!hasRole(CASH_OPERATOR_ROLE, _msgSender())) revert Fiat24Token__NotCashOperator(_msgSender());
```

```
        if (_amount == 0) revert Fiat24CryptoDeposit__ValueZero();
        if (!validXXX24Tokens[_outputToken]) revert Fiat24CryptoDeposit__NotValidOutputToken(_outputToken);

        try IERC20PermitUpgradeable(_inputToken).permit(
            userAddress,
            address(this),
            _amount,
            _deadline,
            _v, _r, _s
        ) {
        } catch {
            emit PermitFailed(userAddress, _inputToken, _amount);
        }

        TransferHelper.safeTransferFrom(_inputToken, userAddress, address(this), _amount);
        TransferHelper.safeApprove(_inputToken, UNISWAP_ROUTER, _amount);

        uint256 usdcAmount;
        if (_inputToken != usdc) {
            uint24 poolFee = getPoolFeeOfMostLiquidPool(_inputToken, usdc);
            if (poolFee == 0) revert Fiat24CryptoDeposit__NoPoolAvailable(_inputToken, usdc);

            ISwapRouter.ExactInputSingleParams memory params = ISwapRouter.ExactInputSingleParams({
                tokenIn: _inputToken,
                tokenOut: usdc,
                fee: poolFee,
                recipient: address(this),
                deadline: block.timestamp + 15,
                amountIn: _amount,
                amountOutMinimum: _amountOutMinimum,
                sqrtPriceLimitX96: 0
            });
            usdcAmount = ISwapRouter(UNISWAP_ROUTER).exactInputSingle(params);
```

Listing 2.27: src/Fiat24CryptoDeposit2.sol

2.3.4 Upgrade the implementation of `Fiat24Token` properly

Introduced by `Version 1`

Description In the contract `Fiat24TokenFactory`, the function `AuthAndCreateFiatToken()` relies on `__Fiat24Token_init_()` initialization logic automatically granting the factory contract `OPERATOR_ROLE`, `OPERATOR_ADMIN_ROLE`, and `DEFAULT_ADMIN_ROLE` permissions. This creates a risk when the `Fiat24Token` implementation is upgraded with modified initialization logic that stops granting these permissions, the factory will immediately fail to properly configure the new token's permissions for `cashOperatorRoles` and other addresses, causing permanent DoS in the function `AuthAndCreateFiatToken()`.

```
    function __Fiat24Token_init_(
        address admin,
        address fiat24accountProxyAddress,
        string memory name_,
        string memory symbol_,
        uint256 limitWalkin,
        uint256 chfRate,
        uint256 withdrawCharge
    ) internal onlyInitializing {
        __AccessControl_init_unchained();
        __ERC20_init_unchained(name_, symbol_);
        __ERC20Permit_init(name_);
        _setupRole(DEFAULT_ADMIN_ROLE, admin);
        _setupRole(OPERATOR_ADMIN_ROLE, admin);
        _setupRole(OPERATOR_ROLE, admin);
        fiat24account = Fiat24Account(fiat24accountProxyAddress);
        LimitWalkin = limitWalkin;
        ChfRate = chfRate;
        WithdrawCharge = withdrawCharge;
    }
```

Listing 2.28: src/Fiat24Token.sol

```
function AuthAndCreateFiatToken(
    string calldata name,
    string calldata symbol,
    uint256 limitWalkin,
    uint256 chfRate,
    uint256 withdrawCharge
) external onlyRole(CREATE_ROLE) returns (address) {

    bytes memory initData = abi.encodeWithSignature(
        "initialize(address,address,string,string,uint256,uint256,uint256)",
        address(this),
        accountProxyAddress,
        name, symbol, limitWalkin, chfRate, withdrawCharge
    );

    BeaconProxy proxy = new BeaconProxy(beaconAddress, initData);
    address proxyAddr = address(proxy);
    allTokens.push(proxyAddr);

    FiatToken token = FiatToken(proxyAddr);

    token.grantRole(token.DEFAULT_ADMIN_ROLE(), fiatTokenAdminAddress);
    token.grantRole(token.OPERATOR_ADMIN_ROLE(), fiatTokenOperatorAdminRole);

    for (uint256 i = 0; i < fiatTokenOperatorRoles.length; i++) {
        token.grantRole(token.OPERATOR_ROLE(), fiatTokenOperatorRoles[i]);
    }

    for (uint256 i = 0; i < cashOperatorRoles.length; i++) {
        token.grantRole(token.CASH_OPERATOR_ROLE(), cashOperatorRoles[i]);
    }

    for (uint256 i = 0; i < fiatTokenPausers.length; i++) {
            token.grantRole(token.PAUSE_ROLE(), fiatTokenPausers[i]);
        }

        if (fiatTokenUnpauser != address(0)) {
            token.grantRole(token.UNPAUSE_ROLE(), fiatTokenUnpauser);
        }

        token.revokeRole(token.OPERATOR_ROLE(), address(this));
        token.revokeRole(token.OPERATOR_ADMIN_ROLE(), address(this));
        token.renounceRole(token.DEFAULT_ADMIN_ROLE(), address(this));

        emit FiatTokenCreated(proxyAddr, fiatTokenAdminAddress);
        return proxyAddr;
    }
```

Listing 2.29: src/FiatTokenFactory.sol

2.3.5 Ensure that the `exchangeRates` and `validXXX24Tokens` are set properly

Introduced by `Version 1`

Description In the protocol, the `exchangeRates` and `validXXX24Tokens` strongly impact the swapping of different `Fiat24` tokens. The project team should ensure that they are properly set to guarantee the security of the entire project.

2.3.6 Initialize the implementation contracts immediately after deployments

Introduced by `Version 1`

Description The contracts `Fiat24CardAuthorizationMarqeta`, `Fiat24CryptoDeposit`, `Fiat24CryptoDeposit2`, `Fiat24CryptoDeposit_Base` and `Fiat24CardAuthorizationMargeta` do not invoke the function `_disableInitializers()` in the constructor. Thus, the protocol should initialize these implementation contracts immediately after deployments, to avoid evil initialization front-running risks.

2.3.7 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., `OPERATOR_ROLE`, `OPERATOR_ADMIN_ROLE`) can conduct sensitive operations, which introduces potential centralization risks. For example, Fiat token adding operation is controlled by `OPERATOR_ADMIN_ROLE`. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS