Security Audit Report for Fiat24

Date: June 9, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Mantle
Target	Fiat24

Version History

Version	Date	Description
1.0	June 9, 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 of Mantle. Fiat24 is a on-chain Web3 banking protocol. It provides functionalities for users to convert crypto assets USDC into fiat currency in their Iban fiat balance. Note this audit only focuses on the smart contracts in the following directories/files:

- src/Fiat24Account.sol
- src/Fiat24CryptoRelay.sol
- src/Fiat24CryptoDeposit2.sol
- src/Fiat24Token.sol

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	Version 1	4e5e5fef12b8484bdcf834d4266d2a36a82d5a6d
	Version 2	a46cc473f4e7cf32ba6a2eebd4177e9d4422cdf2

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.

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

The scope of this audit is limited to the code mentioned in Section 1.1. 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 Software Security

- Reentrancy
- DoS
- Access control
- Data handling and data flow
- Exception handling
- Untrusted external call and control flow
- Initialization consistency
- Events operation
- Error-prone randomness
- Improper use of the proxy system

1.3.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Permission management
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

1.3.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

1.3.4 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 1.1.

Table 1.1: Vulnerability Severity Classification

		Likelihood	
		High	*Low*
Impact	*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:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.

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

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

- **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 **sixteen** potential security issues. Besides, we have **eight** recommendations and **five** notes.

- High Risk: 5
- Medium Risk: 4
- Low Risk: 7
- Recommendation: 8
- Note: 5

ID	Severity	Description	Category	Status
1	High	DoS due to the incorrect refund ETH value	Security Issue	Fixed
2	High	Loss of funds due to the incorrect decoding logic	Security Issue	Fixed
3	High	Improper slippage calculations leading to sandwich attacks	Security Issue	Fixed
4	High	Incorrect use of `_msgSender()` in the function `permitAndDepositTokenViaUsdc()`	Security Issue	Fixed
5	High	Incorrect invocation of the function `transferFrom()`	Security Issue	Fixed
6	Medium	Incorrect use of `usdcAmount` when constructing the variable `payload`	Security Issue	Fixed
7	Medium	Improper approval mechanism when the input `_inputToken` is `USDC`	Security Issue	Fixed
8	Medium	Inaccurate fee estimation due to the incorrect construction of the variable `payload`	Security Issue	Fixed
9	Medium	Potential loss of funds due to improper access control	Security Issue	Confirmed
10	Low	Improper initialization of the variables `lzNativeFee` and `RELAY_GAS_LIMIT`	Security Issue	Fixed
11	Low	Incorrect checks for spreads	Security Issue	Fixed
12	Low	Users can own multiple accounts	Security Issue	Confirmed
13	Low	Improper value of the variable `slippage`	Security Issue	Fixed
14	Low	Potential collision due to duplicate zeros	Security Issue	Confirmed
15	Low	Incorrect custom error usage in the functions `pause()` and `unpause()`	Security Issue	Fixed
16	Low	Lack of logic on handling the residual tokens after swaps	Security Issue	Confirmed
17	-	Add boundary checks	Recommendation	Confirmed
18	-	Lack of non zero address checks	Recommendation	Fixed

19	-	Revise the calculation to avoid potential precision loss	Recommendation	Confirmed
20	-	Redundant code	Recommendation	Partially Fixed
21	-	Potential risk of implementation contract initialization	Recommendation	Confirmed
22	-	Add checks between the variables `usdcAmount` and `_feeAmountViaUsdc`	Recommendation	Fixed
23	-	Revise the misleading variable name	Recommendation	Fixed
24	-	Potential reentrancy risk	Recommendation	Fixed
25	-	The fee charging mechanism in the function `_refundToSource()`	Note	-
26	-	The prohibition of the functions `_mintByClient()`, `mintByClient()`, and `mintByWallet()`	Note	-
27	-	The redeeming mechanism of the `Fiat24Token` token	Note	-
28	-	Fixed `TokenId` for special accounts	Note	-
29	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 DoS due to the incorrect refund ETH value

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `Fiat24CryptoDeposit2` implements the function `depositETH()` which handles ETH-to-USDC conversions while requiring a `nativeFee` for cross-chain messaging. However, the function refunds the entire balance of ETH to users before executing the function `_lzSend()`, failing to account for the reserved nativeFee that should remain for LayerZero operations. This will lead to DoS in the `depositETH()` function due to the insufficient native fee.

```
function depositETH(address _outputToken, uint256 nativeFee) nonReentrant external payable returns (uint256) {
    if (paused()) revert Fiat24CryptoDeposit__Paused();
    if (msg.value == 0) revert Fiat24CryptoDeposit__ValueZero();
    if (!validXXX24Tokens[_outputToken]) revert Fiat24CryptoDeposit__NotValidOutputToken(_outputToken);

    // ETH -> USDC Conversion (via Uniswap)
    uint24 poolFee = getPoolFeeOfMostLiquidPool(weth, usdc);
    if (poolFee == 0) revert Fiat24CryptoDeposit__NoPoolAvailable(weth, usdc);
```

```
        uint256 amountIn = msg.value - nativeFee;

        ISwapRouter.ExactInputSingleParams memory params = ISwapRouter.ExactInputSingleParams({
            tokenIn: weth,
            tokenOut: usdc,
            fee: poolFee,
            recipient: address(this),
            deadline: block.timestamp + 15,
            amountIn: amountIn,
            amountOutMinimum: getQuote(weth, usdc, poolFee, amountIn)
        .sub(getQuote(weth, usdc, poolFee, amountIn).mul(slippage).div(100)),
            sqrtPriceLimitX96: 0
        });

        uint256 usdcAmount = ISwapRouter(UNISWAP_ROUTER).exactInputSingle{value: amountIn}(params);
        IPeripheryPaymentsWithFee(UNISWAP_PERIPHERY_PAYMENTS).refundETH();

        (bool success, ) = msg.sender.call{value: address(this).balance}("");
        if (!success) revert Fiat24CryptoDeposit__EthRefundFailed();

        if (usdcAmount == 0) revert Fiat24CryptoDeposit__SwapOutputAmountZero();
        if (usdcAmount > maxUsdcDepositAmount) revert Fiat24CryptoDeposit__UsdcAmountHigherMaxDepositAmount(usdcAmount, maxUsdcDepositAmount);
        if (usdcAmount < minUsdcDepositAmount) revert Fiat24CryptoDeposit__UsdcAmountLowerMinDepositAmount(usdcAmount, minUsdcDepositAmount);

        // Transfer USDC to the designated deposit address
        TransferHelper.safeTransfer(usdc, usdcDepositAddress, usdcAmount);

        bytes memory payload = abi.encode(
            _msgSender(),
            address(0),
            msg.value,
            usdcAmount,
            _outputToken
        );

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

        MessagingFee memory fee = MessagingFee({
            nativeFee: nativeFee,
            lzTokenFee: 0
        });

        _lzSend(dstChainId, payload, defaultWorkerOptions, fee, payable(msg.sender));

        emit SentDepositedEth(_msgSender(), weth, _outputToken, amountIn, usdcAmount);
        return usdcAmount;
    }
```

Listing 2.1: src/Fiat24CryptoDeposit2.sol

Impact DoS in the `depositETH()` function due to the incorrect refund ETH value.

Suggestion Revise the code logic accordingly.

2.1.2 Loss of funds due to the incorrect decoding logic

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `Fiat24CryptoRelay`, if a cross-chain transaction fails on the Mantle network, the refunding logic is triggered via the function `_refundToSource()`, which sends back the original message (i.e., the variable `payload` constructed on the source chains). However, the `_lzReceive()` function in the contract `Fiat24CryptoDeposit2` decodes the `payload` into only three elements, whereas it is originally constructed with five elements (e.g., in the function `depositETH()`). This incorrect decoding logic leads to a DoS issue during the refund process, potentially resulting in a loss of funds.

```
        bytes memory payload = abi.encode(
            _msgSender(),
            address(0),
            msg.value,
            usdcAmount,
            _outputToken
        );

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

        MessagingFee memory fee = MessagingFee({
            nativeFee: nativeFee,
            lzTokenFee: 0
        });

        _lzSend(dstChainId, payload, defaultWorkerOptions, fee, payable(msg.sender));

        emit SentDepositedEth(_msgSender(), weth, _outputToken, amountIn, usdcAmount);
        return usdcAmount;
    }
```

Listing 2.2: src/Fiat24CryptoDeposit2.sol

```
    function _lzReceive(
        Origin calldata _origin,
        bytes32 _guid,
        bytes calldata payload,
        address /* _executor */,
        bytes calldata /* _extraData */
    ) internal override {

        require(peers(_origin.srcEid) == _origin.sender, "Invalid sender");
        try this.processMessage(payload) {

            emit MessageProcessed(_origin.srcEid, _guid);
        } catch Error(string memory reason) {
            _refundToSource(_origin.srcEid, _origin.sender, payload, reason);
        } catch {
            _refundToSource(_origin.srcEid, _origin.sender, payload, "Unknown failure");
        }
    }

    function processMessage(bytes calldata payload) nonReentrant external {

        if (paused()) revert Fiat24CryptoDeposit__Paused();

        require(msg.sender == address(this), "Only internal calls");

        (address user, address inputToken, uint256 inputAmount, uint256 usdcAmount, address outputToken) =
                          abi.decode(payload, (address, address, uint256, uint256, address));
```

Listing 2.3: src/Fiat24CryptoRelay.sol

```
    function _lzReceive(
        Origin calldata /* _origin */,
        bytes32 _guid,
        bytes calldata _payload,
        address /* _executor */,
        bytes calldata /* _extraData */
    ) internal override {
        (address user, uint256 usdcAmount, address outputToken) = abi.decode(_payload, (address, uint256, address));
        _handleRefund(_guid, user, usdcAmount, outputToken);
    }
```

Listing 2.4: src/Fiat24CryptoDeposit2.sol

Impact This incorrect decoding logic leads to a DoS issue during the refund process, potentially resulting in a loss of funds.

Suggestion Revise the logic accordingly.

2.1.3 Improper slippage calculations leading to sandwich attacks

Severity High

Status Fixed in Version 2

Introduced by Version 1

Description The deposit functions (e.g., depositETH() ,depositTokenViaUsdc()) of the contract Fiat24CryptoDeposit2 calculate the swap slippage (i.e., the variable amountOutMinimum) for swaps using the function getQuote(), which relies on Uniswap V3-like pools' spot price. This design exposes a vulnerability as the use of the spot price is susceptible to manipulation through large trades. Specifically, malicious actors can exploit this by front-running swaps to inflate the price, thereby forcing victims to accept unfavorable rates that align with the manipulated amountOutMinimum. As a result, users may lose funds.

```
    ISwapRouter.ExactInputSingleParams memory params = ISwapRouter.ExactInputSingleParams({
        tokenIn: weth,
        tokenOut: usdc,
        fee: poolFee,
        recipient: address(this),
        deadline: block.timestamp + 15,
        amountIn: amountIn,
        amountOutMinimum: getQuote(weth, usdc, poolFee, amountIn)
    .sub(getQuote(weth, usdc, poolFee, amountIn).mul(slippage).div(100)),
        sqrtPriceLimitX96: 0
    });
```

Listing 2.5: src/Fiat24CryptoDeposit2.sol

```
function getQuote(address _inputToken, address _outputToken, uint24 _fee, uint256 _amount) public returns (uint256) {
    return IQuoter(UNISWAP_QUOTER).quoteExactInputSingle(_inputToken, _outputToken, _fee, _amount, 0);
}
```

Listing 2.6: src/Fiat24CryptoDeposit2.sol

Impact Users may lose funds.

Suggestion Revise the logic accordingly.

2.1.4 Incorrect use of _msgSender() in the function permitAndDepositTokenViaUsdc()

Severity High

Status Fixed in Version 2

Introduced by Version 1

Description The function permitAndDepositTokenViaUsdc() allows the role CASH_OPERATOR_ROLE to perform deposits on behalf of users. However, this function incorrectly uses _msgSender() (i.e., CASH_OPERATOR_ROLE) as the receiver in the construction of the variable payload, instead of using the variable userAddress. As a result, users will lose funds.

Additionally, the fee refund address (i.e., payable(msg.sender)) specified in the invocation of the function _lzSend() should also be specified to userAddress since the fee (i.e., the variable _feeAmountViaUsdc) is pre-charged to the users.

```
function permitAndDepositTokenViaUsdc(
    address userAddress,
    address _inputToken,
    address _outputToken,
    uint256 _amount,
    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);

        uint256 amountOutMinUSDC = getQuote(_inputToken, usdc, poolFee, _amount);
        ISwapRouter.ExactInputSingleParams memory params = ISwapRouter.ExactInputSingleParams({
            tokenIn: _inputToken,
            tokenOut: usdc,
            fee: poolFee,
            recipient: address(this),
            deadline: block.timestamp + 15,
            amountIn: _amount,
            amountOutMinimum: amountOutMinUSDC.sub(amountOutMinUSDC.mul(slippage).div(100)),
            sqrtPriceLimitX96: 0
        });
        usdcAmount = ISwapRouter(UNISWAP_ROUTER).exactInputSingle(params);
    } else {
        usdcAmount = _amount;
    }

    if (usdcAmount == 0) revert Fiat24CryptoDeposit__SwapOutputAmountZero();
    if (usdcAmount > maxUsdcDepositAmount) revert Fiat24CryptoDeposit__UsdcAmountHigherMaxDepositAmount(usdcAmount, maxUsdcDepositAmount);
    if (usdcAmount < minUsdcDepositAmount) revert Fiat24CryptoDeposit__UsdcAmountLowerMinDepositAmount(usdcAmount, minUsdcDepositAmount);

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

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

    bytes memory payload = abi.encode(
        _msgSender(),
        _inputToken,
        _amount,
        usdcAmount,
        _outputToken
    );
    bytes memory options = OptionsBuilder
        .newOptions()
        .addExecutorLzReceiveOption(relay_gas_limit, 0);
    MessagingFee memory fee = MessagingFee({
        nativeFee: msg.value,
        lzTokenFee: 0
    });
    _lzSend(dstChainId, payload, options, fee, payable(msg.sender));

    emit SentDepositedTokenViaUsd(userAddress, _inputToken, _outputToken, _amount, usdcFactAmount);
    return usdcFactAmount;
}
```

Listing 2.7: src/Fiat24CryptoDeposit2.sol

Impact Users will lose funds.

Suggestion Revise the logic accordingly.

Feedback from the project The project team stated that since they are covering the gas fees, the refund of the fees to the role `CASH_OPERATOR_ROLE` is designed to ensure that they do not incur losses in cases where the cross-chain communication fails.

Note The fee refund address is set to be the role `CASH_OPERATOR_ROLE` since the project is covering the gas fees.

2.1.5 Incorrect invocation of the function `transferFrom()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `Fiat24Token`, the function `_permitAndTransferFrom()` first uses the permit mechanism to increase the allowance granted by `userAddress` to the Fiat24Token contract. Subsequently, it calls the function transferFrom() to transfer assets from userAddress to the recipient. However, within the context of the function transferFrom() invocation, the _msgSender() is the address of the CASH_OPERATOR_ROLE, and the currentAllowance being checked is the allowance granted by userAddress to the CASH_OPERATOR_ROLE. As a result, the execution of the function _permitAndTransferFrom() fails.

Additionally, the functions permitAndClientPayoutRef() and permitAndClientPayout() invoking the function transferFromByAccountId() have the same issue.

```
function _permitAndTransferFrom(
    address userAddress,
    address recipient,
    uint256 amount,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) internal returns (bool) {
    try IERC20PermitUpgradeable(address(this)).permit(
        userAddress,
        address(this),
        amount,
        deadline,
        v, r, s
    ) {
    } catch {
        emit PermitFailed(userAddress, address(this), amount);
    }

    return transferFrom(userAddress,recipient,amount);
}
```

Listing 2.8: src/Fiat24Token.sol

```
function permitAndTransferFrom(
    address userAddress,
    address recipient,
    uint256 amount,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) public returns (bool) {
    require(hasRole(CASH_OPERATOR_ROLE, _msgSender()), "Fiat24Token: Not a Cash Operator");
    return _permitAndTransferFrom(userAddress, recipient, amount, deadline, v, r, s);
}
```

Listing 2.9: src/Fiat24Token.sol

```
function transferFrom(address sender, address recipient, uint256 amount) public virtual
     override returns (bool) {
    _transfer(sender, recipient, amount);

```

```
        uint256 currentAllowance = allowance(sender, _msgSender());
        require(currentAllowance >= amount, "ERC20: transfer amount exceeds allowance");
        unchecked {
            _approve(sender, _msgSender(), currentAllowance - amount);
        }

        return true;
    }
```

Listing 2.10: src/Fiat24Token.sol

Impact Potential DoS due to the incorrect invocation of the function `transferFrom()`.

Suggestion Use `this.transferFrom()` instead.

2.1.6 Incorrect use of `usdcAmount` when constructing the variable `payload`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `permitAndDepositTokenViaUSDC()`, a portion of `USDC` tokens (i.e., `_feeAmountViaUsdc`) is deducted from the deposit amount (i.e., `usdcAmount`) to cover cross-chain message fees. However, when constructing the variable `payload`, the function uses `usdcAmount` instead of `usdcFactAmount`, which reflects the actual amount after the fee deduction. As a result, the protocol sends excess `USDC` tokens to the receiver on the Mantle network.

```
        uint256 usdcFactAmount = usdcAmount - _feeAmountViaUsdc;
        TransferHelper.safeTransfer(usdc, feeReceiver, _feeAmountViaUsdc);
        TransferHelper.safeTransfer(usdc, usdcDepositAddress, usdcFactAmount);

        bytes memory payload = abi.encode(
            _msgSender(),
            _inputToken,
            _amount,
            usdcAmount,
            _outputToken
        );
```

Listing 2.11: src/Fiat24CryptoDeposit2.sol

Impact The protocol sends excess `USDC` tokens to the receiver on the Mantle network.

Suggestion Use `usdcFactAmount` during the payload construction.

2.1.7 Improper approval mechanism when the input `_inputToken` is `USDC`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `Fiat24CryptoDeposit2` invokes `safeApprove()` for the `_inputToken` in the function `depositTokenViaUsdc()`, which grants the `UNISWAP_ROUTER` an allowance even when _inputToken is USDC. This creates a vulnerability because no swap operation is required for USDC deposits, rendering the approval unnecessary. The redundant approval exposes the contract Fiat24CryptoDeposit2 to risks where malicious actors could exploit the granted allowance of USDC.

```
function depositTokenViaUsdc(address _inputToken, address _outputToken, uint256 _amount) nonReentrant payable external returns (uint256) {
    if (paused()) revert Fiat24CryptoDeposit__Paused();
    if (_amount == 0) revert Fiat24CryptoDeposit__ValueZero();
    if (!validXXX24Tokens[_outputToken]) revert Fiat24CryptoDeposit__NotValidOutputToken(_outputToken);

    // Transfer token from user and approve to Uniswap
    TransferHelper.safeTransferFrom(_inputToken, _msgSender(), address(this), _amount);
    TransferHelper.safeApprove(_inputToken, UNISWAP_ROUTER, _amount);

    uint256 usdcAmount;
    if (_inputToken != usdc) {
```

Listing 2.12: src/Fiat24CryptoDeposit2.sol

Impact The redundant approval exposes the contract Fiat24CryptoDeposit2 to risks where malicious actors could exploit the granted allowance of USDC.

Suggestion Revise the logic accordingly.

2.1.8 Inaccurate fee estimation due to the incorrect construction of the variable payload

Severity Medium

Status Fixed in Version 2

Introduced by Version 1

Description The function quote() in the contract Fiat24CryptoDeposit2 constructs an incorrect payload for gas fee estimation. The payload contains only 3 elements (_userAddress, _usdcAmount, _outputToken), while the actual cross-chain message execution (e.g., in the function permitAndDepositTokenViaUsdc()) uses 5 elements (including _inputToken and _amount). This mismatch may lead to inaccurate gas fee estimations, as LayerZero's fee calculation depends on the payload size and structure.

```
function quote(
    uint32 _dstEid,
    address _userAddress,
    uint256 _usdcAmount,
    address _outputToken
) public view returns (MessagingFee memory fee) {
    bytes memory payload = abi.encode(
        _userAddress,
        _usdcAmount,
        _outputToken
    );

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

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

Listing 2.13: src/Fiat24CryptoDeposit2.sol

Impact Inaccurate gas fee estimations.

Suggestion Revise the construction of the variable `payload` in the function `quote()`.

2.1.9 Potential loss of funds due to improper access control

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description Anyone can invoke the function `retryFailedRefund()` to retry refunding for other users. However, a malicious user can invoke the function `retryFailedRefund()` for others repeatedly when the `USDC` balance of the `usdcDepositAddress` is insufficient or the allowance that the `usdcDepositAddress` approved to the `Fiat24CryptoDeposit2` is insufficient. When normal users want to get their refunds when the `usdcDepositAddress` has enough `USDC`, they will not be able to get back their funds due to the variable `refund.retryCount` consumed by a malicious user.

```
function retryFailedRefund(bytes32 _guid) external {
    if (paused()) revert Fiat24CryptoDeposit__Paused();

    FailedRefund storage refund = failedMessages[_guid];
    require(refund.user != address(0), "No failed message");
    require(refund.retryCount < MAX_RETRY_COUNT, "Max retries reached");

    try IERC20Upgradeable(usdc).transferFrom(usdcDepositAddress, refund.user, refund.usdcAmount
        ) {
        delete failedMessages[_guid];
        emit RefundRetried(_guid);
    } catch {
        unchecked { refund.retryCount++; }
        emit RefundProcessFailed(_guid, refund.user, refund.usdcAmount);
    }
}
```

Listing 2.14: src/Fiat24CryptoDeposit2.sol

Impact Potential funds loss of users.

Suggestion Revise the code logic accordingly.

2.1.10 Improper initialization of the variables `lzNativeFee` and `RELAY_GAS_LIMIT`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contracts `Fiat24CryptoDeposit2` and `Fiat24CryptoRelay`, the storage variables `lzNativeFee` and `RELAY_GAS_LIMIT` are improperly initialized due to the use of a proxy pattern. As a result, these variables may remain unset, potentially leading to incorrect fee calculations and failed cross-chain operations.

```
uint256 private lzNativeFee = 198009600000000;
```

Listing 2.15: src/Fiat24CryptoDeposit2.sol

```
uint128 public RELAY_GAS_LIMIT = 500000;
```

Listing 2.16: src/Fiat24CryptoRelay.sol

Impact The variables `lzNativeFee` and `RELAY_GAS_LIMIT` variables may remain unset, potentially leading to incorrect fee calculations and failed cross-chain operations.

Suggestion Initialize the variables `lzNativeFee` and `RELAY_GAS_LIMIT` in the function `initialize()`.

2.1.11 Incorrect checks for spreads

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The functions `changeMarketClosedSpread()` and `changeExchangeSpread()` perform `require` checks for the value of spreads (i.e., `marketClosedSpread` and `exchangeSpread`), which check if input values fall between `9000` and `11000`. However, the accompanying error messages state that the allowed range is `9000` to `10000`, creating a discrepancy between the validation logic and the error messages. This inconsistency introduces ambiguity, as administrators or external callers may misinterpret the permissible bounds for configuring spreads.

```
    require(_marketClosedSpread >= 9000 && _marketClosedSpread <= 11000, "Spread must be between 9000 and 10000");
```

Listing 2.17: src/Fiat24CryptoRelay.sol

```
    require(_exchangeSpread >= 9000 && _exchangeSpread <= 11000, "Spread must be between 9000 and 10000");
```

Listing 2.18: src/Fiat24CryptoRelay.sol

Impact This inconsistency introduces ambiguity, as administrators or external callers may misinterpret the permissible bounds for configuring spreads.

Suggestion Unify the validation logics and the error messages.

2.1.12 Users can own multiple accounts

Severity Low

Status Confirmed

Introduced by `Version 1`

Description The contract `Fiat24Account` implements an NFT-based account system where token transfers are conditionally restricted in the function `_beforeTokenTransfer()`. While the logic aims to enforce "one account per user" by checking `balanceOf(to) < 1` (line 356 and 367) and `historicOwnership` constraints. This design creates ambiguity about whether multi-account ownership is intentionally permitted for specific states (i.e., `Status.Tourist`). Specifically, contracts or EOAs with code (refer to the EIP-7702) can receive multiple NFTs with the status `Status.Tourist` (line 350-351). As a result, users can own multiple accounts (i.e., NFTs).

```
344  function _beforeTokenTransfer(address from, address to, uint256 tokenId)
345      internal
346      virtual
347      override(ERC721EnumerableUpgradeable, ERC721PausableUpgradeable)
348  {
349      require(!paused(), "Account transfers suspended");
350      if (AddressUpgradeable.isContract(to) && (from != address(0))) {
351          require(this.status(tokenId) == Status.Tourist, "Not allowed to transfer account");
352      } else {
353          if ((from != address(0) && to != address(0))) {
354              if (_exists(9106)) {
355                  require(
356                      (balanceOf(to) < 1 && (historicOwnership[to] == 0 || historicOwnership[to]
                             == tokenId))
357                          || (tokenOfOwnerByIndex(to, 0) == 9106 && this.status(tokenId) == Status.
                                 Closed),
358                      "Not allowed. The target address has an account or once had another account.
                             "
359                  );
360                  require(
361                      (this.status(tokenId) == Status.Live || this.status(tokenId) == Status.
                             Tourist)
362                          || (balanceOf(to) > 0 && tokenOfOwnerByIndex(to, 0) == 9106 && this.
                                 status(tokenId) == Status.Closed),
363                      "Transfer not allowed in this status"
364                  );
365              } else {
366                  require(
367                      balanceOf(to) < 1 && (historicOwnership[to] == 0 || historicOwnership[to] ==
                              tokenId),
368                      "Not allowed. The target address has an account or once had another account.
                             "
369                  );
370                  require(this.status(tokenId) == Status.Live || this.status(tokenId) == Status.
                         Tourist, "Transfer not allowed in this status");
371              }
372          }
```

```
        }
        super._beforeTokenTransfer(from, to, tokenId);
    }
```

Listing 2.19: src/Fiat24Account.sol

Impact Users can own multiple accounts (i.e., NFTs).

Suggestion Revise the logic accordingly.

Feedback from the project The project stated that each user can own multiple NFTs with the `Tourist` status.

2.1.13 Improper value of the variable `slippage`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description `5%` is too large for a slippage check in a bank protocol. An MEV bot can conduct a sandwich attack to steal the funds of users.

```
        slippage = 5;
```

Listing 2.20: src/Fiat24CryptoDeposit2.sol

Impact Loss of funds due to improper value of the variable `slippage`.

Suggestion Set a proper value of the variable `slippage`.

2.1.14 Potential collision due to duplicate zeros

Severity Low

Status Confirmed

Introduced by `Version 1`

Description The contract `Fiat24Token` implements the function `bytes32ToString()` which converts bytes32 data to strings by removing zero bytes. The contradiction arises when distinct bytes32 values that differ only in zero bytes generate identical string outputs, which influences the functions (e.g., `cashDepositNOK()`) where these strings are used as unique keys for tracking pacs008 transactions. This behavior may cause the system to incorrectly identify transactions.

```
    function bytes32ToString(bytes32 _bytes32) internal pure returns (string memory) {
        bytes memory bytesArray = new bytes(32);
        uint256 bytesArrayIndex = 0;
        for (uint256 i = 0; i < 32; i++) {
            if (_bytes32[i] != 0) {
                bytesArray[bytesArrayIndex] = _bytes32[i];
                bytesArrayIndex++;
            }
        }
        bytes memory trimmedBytes = new bytes(bytesArrayIndex);
        for (uint256 j = 0; j < bytesArrayIndex; j++) {
```

```
        trimmedBytes[j] = bytesArray[j];
    }
    return string(trimmedBytes);
}
```

Listing 2.21: src/Fiat24Token.sol

```
function cashDepositNOK(uint256 recipientAccountId, uint256 amount, string memory exaccId, string memory bankId, string memory trxId) external {
    require(hasRole(CASH_OPERATOR_ROLE, _msgSender()), "Fiat24Token: Not a Cash Operator");
    bytes32 key = keccak256(abi.encodePacked(bankId, "-", trxId));
    require(pacs008[bytes32ToString(key)] == 0, "Fiat24Token: pacs008 already processed");
    pacs008[bytes32ToString(key)] = block.number;
    transferFrom(fiat24account.ownerOf(9101), fiat24account.ownerOf(9103), amount);
    emit CashDepositNOK(recipientAccountId, 9103, amount, exaccId, bankId, trxId);
}
```

Listing 2.22: src/Fiat24Token.sol

Impact This may lead to incorrect `pacs008` tracking, blocking valid transactions.

Suggestion Revise the logic accordingly.

Feedback from the project The project stated that this is a case with an extremely low probability. Currently, the project employs a manual process to handle exceptional cases. The project might plan to adopt a more efficient approach for recording this information in the future.

2.1.15 Incorrect custom error usage in the functions `pause()` and `unpause()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contracts `Fiat24CryptoRelay` and `Fiat24CryptoDeposit2` implement a pausing mechanism with the functions `pause()` and `unpause()`, which revert transactions using the custom error `Fiat24CardAuthorizationMarqeta__NotPauser()`. However, this error is defined in the other contract (i.e., `Fiat25CardAuthorizationMarqeta`). The misalignment creates a critical inconsistency, potentially disrupting off-chain operations.

```
function pause() external {
    if (!(hasRole(PAUSE_ROLE, _msgSender()))) revert Fiat24CardAuthorizationMarqeta__NotPauser(_msgSender());
    _pause();
}

function unpause() external {
    if (!(hasRole(UNPAUSE_ROLE, _msgSender()))) revert Fiat24CardAuthorizationMarqeta__NotUnpauser(_msgSender());
    _unpause();
}
```

Listing 2.23: src/Fiat24CryptoRelay.sol

```
function pause() external {
    if (!(hasRole(PAUSE_ROLE, _msgSender()))) revert Fiat24CardAuthorizationMarqeta__NotPauser(
        _msgSender());
    _pause();
}

function unpause() external {
    if (!(hasRole(UNPAUSE_ROLE, _msgSender()))) revert
        Fiat24CardAuthorizationMarqeta__NotUnpauser(_msgSender());
    _unpause();
}
```

Listing 2.24: src/Fiat24CryptoDeposit2.sol

Impact The misalignment of the custom error potentially disrupts off-chain operations

Suggestion Revise the custom error.

2.1.16 Lack of logic on handling the residual tokens after swaps

Severity Low

Status Confirmed

Introduced by `Version 1`

Description The functions `depositTokenViaUsdc()`, `permitAndDepositTokenViaUsdc()`, as well as `depositTokenViaEth()` perform swaps in Uniswap V3-like pools during deposits. However, these functions lack logic to handle any residual `inputToken` tokens remaining after the swap. Specifically, if the pool lacks sufficient liquidity, not all `_inputToken` tokens are consumed. This can result in locked funds in the contract `Fiat24CryptoDeposit2`.

Additionally, the approval granted to `UNISWAP_ROUTER` via the function `safeApprove()` may not be fully utilized, leaving excess user-approved tokens exposed to potential misuse.

```
    ISwapRouter.ExactInputSingleParams memory params = ISwapRouter.ExactInputSingleParams({
        tokenIn: _inputToken,
        tokenOut: weth,
        fee: poolFee,
        recipient: address(this),
        deadline: block.timestamp + 15,
        amountIn: _amount,
        amountOutMinimum: getQuote(_inputToken, weth, poolFee, _amount).sub(getQuote(
            _inputToken, weth, poolFee, _amount).mul(slippage).div(100)),
        sqrtPriceLimitX96: 0
    });
    uint256 outputAmount = ISwapRouter(UNISWAP_ROUTER).exactInputSingle(params);
    if (outputAmount == 0) revert Fiat24CryptoDeposit__SwapOutputAmountZero();
```

Listing 2.25: src/Fiat24CryptoDeposit2.sol

Impact Residual tokens will be locked in the contract.

Suggestion Add logic handling the residual tokens after the swap.

Note The residual ERC20 tokens will be locked in the contract due to the lack of withdraw functions for the ERC20 tokens.

2.2 Recommendation

2.2.1 Add boundary checks

Status Confirmed

Introduced by Version 1

Description To avoid potential mis-operations, it is recommended to add boundary checks for the functions changeMaxUsdcDepositAmount(), changeMinUsdcDepositAmount(), as well as changeSlippage().

```
function changeMaxUsdcDepositAmount(uint256 _maxUsdcDepositAmount) external {
    if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
        Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
    maxUsdcDepositAmount = _maxUsdcDepositAmount;
}

function changeMinUsdcDepositAmount(uint256 _minUsdcDepositAmount) external {
    if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
        Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
    minUsdcDepositAmount = _minUsdcDepositAmount;
}

function changeSlippage(uint256 _slippage) external {
    if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
        Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
    slippage = _slippage;
}
```

Listing 2.26: src/Fiat24CryptoDeposit2.sol

Suggestion Add boundary checks for the aforementioned functions.

2.2.2 Lack of non zero address checks

Status Fixed in Version 2

Introduced by Version 1

Description In the functions changeUsdcAddress() and initialize(), several address variables (e.g., _usdcAddress, _cnh24) are not checked to ensure they are not zero. It is recommended to add such checks to prevent potential mis-operations.

```
function changeUsdcAddress(address _usdcAddress) external {
    if (!hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) revert
        Fiat24CryptoDeposit__NotOperatorAdmin(_msgSender());
    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 2.27: src/Fiat24CryptoDeposit2.sol

```
function initialize(
    address admin,
    address _delegate,
    address _fiat24account,
    address _usd24,
    address _eur24,
    address _chf24,
    address _gbp24,
    address _cnh24,
    address _usdc
) public initializer {

    require(admin != address(0), "admin is zero");
    require(_delegate != address(0), "delegate is zero");
    require(_fiat24account != address(0), "fiat24account is zero");
    require(_usd24 != address(0), "usd24 is zero");
    require(_eur24 != address(0), "eur24 is zero");
    require(_chf24 != address(0), "chf24 is zero");
    require(_gbp24 != address(0), "gbp24 is zero");
    require(_usdc != address(0), "usdc is zero");
```

Listing 2.28: src/Fiat24CryptoRelay.sol

Suggestion Add non-zero address checks accordingly.

2.2.3 Revise the calculation to avoid potential precision loss

Status Confirmed

Introduced by `Version 1`

Description In the function `processMessage()` of the contract `Fiat24CryptoRelay`, when calculating the variable `outputAmount`, the multiplication operation should be performed before the division operation to prevent precision loss.

This issue is also present in the function `moneyExchangeExactIn()` when calculating the variable `outputAmount` .

```
    uint256 outputAmount = (usdcAmount - feeInUSDC)
        .div(USDC_DIVISOR)
        .mul(exchangeRates[usdc][usd24])
        .div(XXX24_DIVISOR);
    outputAmount = outputAmount
        .mul(getExchangeRate(usd24, outputToken))
        .div(XXX24_DIVISOR)
        .mul(getSpread(usd24, outputToken, false))
        .div(XXX24_DIVISOR);
```

Listing 2.29: src/Fiat24CryptoRelay.sol

```
    uint256 outputAmount =
        _inputAmount * getExchangeRate(_inputToken, _outputToken) / XXX24_DIVISOR * getSpread(_inputToken, _outputToken, false) / XXX24_DIVISOR;
```

Listing 2.30: src/Fiat24CryptoRelay.sol

Suggestion Revise the code logic accordingly.

2.2.4 Redundant code

Status Partially Fixed

Introduced by `Version 1`

Description There are several unused imports, variables, events, functions. It is recommended to remove them for better code readability. Specifically, the following code should be removed or revised.

1. The input `_tokenId` in the function `getFee()` is redundant.

```
function getFee(uint256 _tokenId, uint256 _usdcAmount) public view returns (uint256 feeInUSDC) {

    // updating
    uint256 _fee = standardFee;
    feeInUSDC = _usdcAmount * _fee / 10000;
}
```

Listing 2.31: src/Fiat24CryptoRelay.sol

2. The following verification is redundant.

```
    require(peers(_origin.srcEid) == _origin.sender, "Invalid sender");
```

Listing 2.32: src/Fiat24CryptoRelay.sol

3. The following variables are redundant.

```
mapping(uint256 => uint256) public oldTokenId;
```

Listing 2.33: src/Fiat24Account.sol

```
    uint256 tokenId = this.tokenOfOwnerByIndex(_msgSender(), 0);
```

Listing 2.34: src/Fiat24Account.sol

4. The use of `this` is redundant.

```
        historicOwnership[this.ownerOf(tokenId)] = tokenId;
```

Listing 2.35: src/Fiat24Account.sol

5. The following functions are redundant.

```
function setMinDigitForSale(uint8 minDigit) external {
    require(hasRole(OPERATOR_ADMIN_ROLE, msg.sender), "Not an admin operator");
    minDigitForSale = minDigit;
}
```

Listing 2.36: src/Fiat24Account.sol

```
function setFiat24LockAddress(address fiat24lockAddress_) external {
    require(hasRole(OPERATOR_ADMIN_ROLE, msg.sender), "Fiat24Token: Not an operator admin");
    fiat24lockAddress = fiat24lockAddress_;
}
```

Listing 2.37: src/Fiat24Token.sol

6. The verification of `from != address(0)` is redundant.

```
        } else if (from != address(0) && fiat24account.balanceOf(from) > 0) {
```

Listing 2.38: src/Fiat24Token.sol

```
        } else if (to != address(0) && fiat24account.balanceOf(to) > 0) {
```

Listing 2.39: src/Fiat24Token.sol

7. The following imports, variables, and events are redundant.

```
import "./interfaces/IFiat24Lock.sol";
import "./interfaces/ArbSys.sol";
```

Listing 2.40: src/Fiat24Token.sol

src/Fiat24Token.sol #L32-32
src/Fiat24Token.sol #L39-39

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

Listing 2.41: src/Fiat24Token.sol

```
event CashLocked(uint256 indexed recipientAccountId, address indexed recipientAddress, string
    exaccId, string bankId, string trxId);
```

Listing 2.42: src/Fiat24Token.sol

```
import "@openzeppelin/contracts/access/Ownable.sol";
```

Listing 2.43: src/Fiat24CryptoRelay.sol

```
import "@uniswap/v3-core/contracts/interfaces/IUniswapV3Factory.sol";
import "@uniswap/v3-core/contracts/interfaces/IUniswapV3Pool.sol";
import "@uniswap/v3-periphery/contracts/interfaces/ISwapRouter.sol";
import "@uniswap/v3-periphery/contracts/interfaces/IPeripheryPaymentsWithFee.sol";
import "@uniswap/v3-periphery/contracts/interfaces/IQuoter.sol";
```

Listing 2.44: src/Fiat24CryptoRelay.sol

```
import {ICrossChainMessenger} from "./interfaces/ICrossChainMessenger.sol";
```

Listing 2.45: src/Fiat24CryptoRelay.sol

```
import "./interfaces/IF24.sol";
import "./interfaces/IF24TimeLock.sol";
```

Listing 2.46: src/Fiat24CryptoRelay.sol

Suggestion Remove the redundant code.

Note The points 1, 3, 4, 5 and 6 are unfixed.

2.2.5 Potential risk of implementation contract initialization

Status Confirmed

Introduced by `Version 1`

Description The contracts `Fiat24Token`, `Fiat24Account`, `Fiat24CryptoDeposit2`, and `Fiat24CryptoRelay` are implementation contracts, which can be initialized by a malicious user. This can lead to unexpected behaviors.

Suggestion Add solutions to avoid potential risk of contract initialization.

Feedback from the project The project stated that they will invoke the function `initialize()` when deploying the implementation contracts to avoid the malicious initializations.

2.2.6 Add checks between the variables `usdcAmount` and `_feeAmountViaUsdc`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `permitAndDepositTokenViaUsdc()` should ensure that the variable `usdcAmount` is greater than the variable `_feeAmountViaUsdc`. This safeguard would prevent an overflow when calculating the variable `usdcFactAmount`, which could otherwise lead to a revert.

```
        if (usdcAmount == 0) revert Fiat24CryptoDeposit__SwapOutputAmountZero();
        if (usdcAmount > maxUsdcDepositAmount) revert
            Fiat24CryptoDeposit__UsdcAmountHigherMaxDepositAmount(usdcAmount, maxUsdcDepositAmount
            );
        if (usdcAmount < minUsdcDepositAmount) revert
            Fiat24CryptoDeposit__UsdcAmountLowerMinDepositAmount(usdcAmount, minUsdcDepositAmount)
            ;

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

        uint256 usdcFactAmount = usdcAmount - _feeAmountViaUsdc;
```

Listing 2.47: src/Fiat24CryptoDeposit2.sol

Suggestion Add checks between the variables `usdcAmount` and `_feeAmountViaUsdc`.

2.2.7 Revise the misleading variable name

Status Fixed in Version 2

Introduced by Version 1

Description The contract Fiat24CryptoDeposit2 uses the variable name dstChainId, which implies it represents a chain ID. The contradiction arises because the variable actually stores an endpoint ID, which is a distinct concept in LayerZero's cross-chain messaging protocol. This discrepancy creates confusion that may lead to improper usage of the function _lzSend(), as the parameter requires an endpoint ID rather than a conventional chain ID.

```
    dstChainId = _dstChainId;
```

Listing 2.48: src/Fiat24CryptoDeposit2.sol

```

    _lzSend(dstChainId, payload, defaultWorkerOptions, fee, payable(msg.sender));
```

Listing 2.49: src/Fiat24CryptoDeposit2.sol

Suggestion Use dstEid instead.

2.2.8 Potential reentrancy risk

Status Fixed in Version 2

Introduced by Version 1

Description In the contract Fiat24Account, the function safeTransferFrom() does not adhere to the checks-effects-interactions pattern, which may lead to reentrancy risks. Specifically, the function safeTransferFrom() of the ERC721 standard invokes the callback function (i.e., the function onERC721Received()) on the receiver contract. A contract could use its function onERC721Received() to invoke the other functions in the Fiat24Account contract before the variable historicOwnership is updated.

```
function safeTransferFrom(address from, address to, uint256 tokenId) public virtual override(
    ERC721Upgradeable, IERC721Upgradeable) {
    super.safeTransferFrom(from, to, tokenId);
    if (status[tokenId] != Status.Tourist) {
        historicOwnership[to] = tokenId;
    }
}
```

Listing 2.50: src/Fiat24Account.sol

Suggestion Revise the code logic accordingly to adhere to the checks-effects-interactions pattern.

2.3 Note

2.3.1 The fee charging mechanism in the function _refundToSource()

Introduced by Version 1

Description The contract `Fiat24CryptoRelay` implements the function `_refundToSource()` to handle cross-chain refund operations. This function uses a fixed fee (i.e., `fixedNativeFee`) for the message forwarding process. The fee is paid by the contract `Fiat24CryptoRelay` itself. The project team stated that they would periodically deposit fees into the `Fiat24CryptoRelay` contract to cover the gas costs incurred during refunds.

```
function _refundToSource(
    uint32 _srcEid,
    bytes32 _srcOApp,
    bytes calldata refundPayload,
    string memory reason
) internal {

    bytes memory options = OptionsBuilder
        .newOptions()
        .addExecutorLzReceiveOption(RELAY_GAS_LIMIT, 0);

    MessagingFee memory fee = MessagingFee({
        nativeFee: fixedNativeFee,
        lzTokenFee: 0
    });

    this.externalLzSend{value: fixedNativeFee}(
        _srcEid,
        refundPayload,
        options,
        fee,
        payable(address(this))
    );

    emit RefundSent(_srcEid, _srcOApp, reason);
}
```

Listing 2.51: src/Fiat24CryptoRelay.sol

2.3.2 The prohibition of the functions `_mintByClient()`, `mintByClient()`, and `mintByWallet()`

Introduced by Version 1

Description In the contract `Fiat24Account`, the functions `_mintByClient()`, `mintByClient()`, and `mintByWallet()` are currently disabled using `revert`. The project stated that these interfaces are temporarily disabled but may be re-enabled in the future through contract upgrades.

```
function mintByClient(uint256 _tokenId) external {
    revert("This function is disabled");
    _mintByClient(_tokenId);
}

function _mintByClient(uint256 _tokenId) internal {
    revert("This function is disabled");
    require(!_tokenId.hasFirstDigit(INTERNALDIGIT), "9xx cannot be mint by client");
```

```
        require(_tokenId.numDigits() <= maxDigitForSale, "Number of digits of accountId > max. digits");
        require(_mintAllowed(_msgSender(), _tokenId), "Not allowed. The address has/had another NFT.");
        _mint(_msgSender(), _tokenId);
        status[_tokenId] = Status.Tourist;
        initilizeTouristLimit(_tokenId);
        nickNames[_tokenId] = string(abi.encodePacked("Account ", StringsUpgradeable.toString(_tokenId)));
    }

    // Allow the wallet provider (i.e. the account with the specified conditions) to
    // mint a new Fiat24 account (in the form of an NFT) for another address
    function mintByWallet(address to, uint256 _tokenId) external {
        revert("This function is disabled");
        require(this.balanceOf(_msgSender()) > 0, "Minting address has no account");
        uint256 minterTokenId = this.tokenOfOwnerByIndex(_msgSender(), 0);
        require(minterTokenId.hasFirstDigit(MERCHANTDIGIT) && (minterTokenId >= 8 && minterTokenId <= 8999), "Incorrect account id for wallet");
        require(walletProviderMap[minterTokenId].isAvailable, "Account not wallet provider");
        require(_tokenId.numDigits() >= 5, "mintByWallet only for 5+ digits tokens");
        require(_tokenId.numDigits() <= maxDigitForSale, "Number of digits of accountId > max. digits");
        require(!_tokenId.hasFirstDigit(INTERNALDIGIT), "9xx cannot be mint by client");
        require(!_tokenId.hasFirstDigit(MERCHANTDIGIT), "Merchant account cannot be minted by wallet");
        require(_mintAllowed(to, _tokenId), "Not allowed. The target address has an account or once had another account.");
        walletProvider[_tokenId] = minterTokenId;
        status[_tokenId] = Status.Tourist;
        _mint(to, _tokenId);
    }
```

Listing 2.52: src/Fiat24Account.sol

2.3.3 The redeeming mechanism of the `Fiat24Token` token

Introduced by `Version 1`

Description The contract `Fiat24Token` operates as an accounting token that users obtain by depositing `USDC`. The project stated that there is no redeeming mechanism that allows users to burn `Fiat24` tokens and reclaim their `USDC`.

2.3.4 Fixed `TokenId` for special accounts

Introduced by `Version 1`

Description The protocol introduced several special accounts which are hardcoded as fixed IDs (e.g., `CRYPTO_DESK`, `TREASURY_DESK`, and `FEE_DESK` in contract `Fiat24CryptoRelay`). We assume these accounts are systematically managed through off-chain processes, or else this may lead to significant financial losses.

2.3.5 Potential centralization risks

Introduced by `Version 1`

Description In this protocol, privileged roles (e.g., `CASH_OPERATOR_ROLE`, `DEFAULT_ADMIN_ROLE`, `OPERATOR_ADMIN_ROLE`) can conduct sensitive operations, which introduces potential centralization risks. For example, the `CASH_OPERATOR_ROLE` can directly transfer tokens from users to any recipient. 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