BLOCKSEC

Security Audit
Report for Fiat24 Contracts

Date: June 8, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Mantle
Target	Fiat24 Contracts

Version History

Version	Date	Description
1.0	June 8, 2026	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-06-11 03:12:48+0000

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 the Audit Target

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

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of Fiat24 Contracts of Mantle.

The Fiat24 Contracts of Mantle is a fiat-crypto payment infrastructure on Mantle that enables card authorization processing and operator-controlled vault management for fiat token operations. This update introduces a delegation vault and a double-booking mechanism. The delegation vault lets operators handle payouts, asset settlements, and rebalancing between Fiat24 tokens and USDC via the contract BufferPool and CryptoDeposit. The double-booking mechanism in the contract Fiat24CardAuthorizationMarqeta ensures that purchases via Ethena are properly tracked on-chain, preserving a verifiable record that links each payment to corresponding users.

Note this audit only focuses on the smart contracts in the following directories/files. Code prior to and including the baseline version 0, where applicable, is outside the scope of this audit and assumes to be reliable and secure.

- src/DelegationVault.sol
- src/Fiat24CardAuthorizationMarqeta.sol
- src/Fiat24CryptoDeposit_Avax.sol

Other files are not within the scope of the audit. Additionally, all dependencies of the Target 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. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
fiat24contracts	`Version 0`	`b670bb8fca437938970d2faef82ed7167b833b0e`
	`Version 1`	`98502dbcca573d8c5d622e82365a2ce5095de356`
	`Version 2`	`ad908801e2dc30fb805551dc117268312f6fdbc7`

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

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 Target, 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 the Target.

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 the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target 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 1.1.

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:

- **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 **two** potential security issues. Besides, we have **five** recommendations and **three** notes.

- Medium Risk: 1
- Low Risk: 1
- Recommendation: 5
- Note: 3

ID	Severity	Description	Category	Status
1	Medium	Lock of native tokens in the contract `DelegationVault`	Security Issue	Fixed
2	Low	Lack of wallet provider checks in the function `settleTokenToAccount()`	Security Issue	Fixed
3	-	Revise the misleading names in the skip mechanism	Recommendation	Confirmed
4	-	Revise the input name `amountOutMinimum` of the `_depositTOkenViaUsdc()` function	Recommendation	Fixed
5	-	Remove redundant code	Recommendation	Partially Fixed
6	-	Correct misleading error names	Recommendation	Partially Fixed
7	-	Revise the inconsistent error message in the `setAlternativeInputTokens()` function	Recommendation	Fixed
8	-	Ensure proper off-chain parameter validations	Note	-
9	-	Proper approvals for double booking operations	Note	-
10	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Lock of native tokens in the contract `DelegationVault`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `DelegationVault`, the function `receive()` is defined to receive native tokens (i.e., `MNT`). However, there are no functions spending native tokens. As a result, received native tokens may be locked in the contract `DelegationVault`.

```
477    receive() external payable {}
```

Listing 2.1: src/DelegationVault.sol

Impact The native tokens may be locked in the contract `DelegationVault`.

Suggestion Revise the code logic accordingly.

2.1.2 Lack of wallet provider checks in the function `settleTokenToAccount()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `DelegationVault`, the function `settleTokenToAccount()` settles tokens to a provided account (i.e., `accountId`). However, it lacks checks to ensure the account satisfies the wallet-provider validation (i.e., `_validateWalletProvider()`). As a result, tokens can be settled to accounts, which are not registered with the wallet provider, violating the intended design.

```
216   function settleTokenToAccount(address token, uint256 accountId, uint256 amount) external
          nonReentrant whenNotPaused onlyRole(SETTLEMENT_ROLE) {
217       _requireValidFiatToken(token);
218       _validateSettlementAccount(accountId);
219       address receiver = fiat24Account.ownerOf(accountId);
220       _transferTokenToAccount(token, accountId, amount);
221       emit TokenSettledToAccount(token, accountId, receiver, amount);
222   }
```

Listing 2.2: src/DelegationVault.sol

```
447   function _validateWalletProvider(uint256 tokenId) internal view {
448       uint256 requiredWalletProvider = walletProvider;
449       if (requiredWalletProvider == 0) {
450           revert DelegationVault__WalletProviderNotSet();
451       }
452
453       uint256 actualWalletProvider = fiat24Account.walletProvider(tokenId);
454       if (actualWalletProvider != requiredWalletProvider) {
455           revert DelegationVault__WalletProviderMismatch(tokenId, actualWalletProvider,
                  requiredWalletProvider);
456       }
457   }
```

Listing 2.3: src/DelegationVault.sol

Impact Tokens can be settled to accounts, which are not registered with the wallet provider, violating the intended design.

Suggestion Add the wallet-provider validation in the function `settleTokenToAccount()`.

2.2 Recommendation

2.2.1 Revise the misleading names in the skip mechanism

Status Confirmed

Introduced by `Version 1`

Description In the contract `DelegationVault`, the mapping `skipFxToUsd24Tokens` lets registered Fiat24 tokens skip the USD24 conversion in the function `_fxToUsd24()`. However, based

on the project's intended design, all registered Fiat24 tokens (e.g., EUR24, CHF24, GBP24) should skip the conversion. It is recommended to revise the corresponding names (e.g., the event name `UserFxToUsd24`, the variable name `usd24Received`, the function names `_fxToUsd24()` and `permitAndFxToUsd24()`) for better code readability.

```
403   function _fxToUsd24(address tokenIn, uint256 amountIn, uint256 minUsd24Out) internal returns (
          uint256 usd24Received) {
404       if (tokenIn == address(usd24) || skipFxToUsd24Tokens[tokenIn]) {
405           if (amountIn < minUsd24Out) revert DelegationVault__InvalidAmount();
406           return amountIn;
407       }
408
409       uint256 beforeBalance = usd24.balanceOf(address(this));
410       _approveExact(tokenIn, address(cryptoRelay), amountIn);
411       uint256 relayOutput = cryptoRelay.moneyExchangeExactIn(tokenIn, address(usd24), amountIn,
              minUsd24Out);
412       usd24Received = usd24.balanceOf(address(this)) - beforeBalance;
413       if (usd24Received < relayOutput) revert DelegationVault__InvalidAmount();
414   }
```

Listing 2.4: src/DelegationVault.sol

```
192   function permitAndFxToUsd24(PermitFxToUsd24Params calldata params) external nonReentrant
          whenNotPaused onlyRole(CASH_OPERATOR_ROLE) returns (uint256 usd24Received) {
193       if (params.user == address(0) || params.tokenIn == address(0)) revert
              DelegationVault__ZeroAddress();
194       if (params.amountIn == 0) revert DelegationVault__InvalidAmount();
195       _requireValidFiatToken(params.tokenIn);
196       _validateUserAccount(params.user);
197
198       try IERC20PermitUpgradeable(params.tokenIn).permit(params.user, address(this), params.
              amountIn, params.deadline, params.v, params.r, params.s) {}
199       catch {
200           emit PermitFailed(params.user, params.tokenIn, params.amountIn);
201       }
202
203       IERC20Upgradeable(params.tokenIn).safeTransferFrom(params.user, address(this), params.
              amountIn);
204       usd24Received = _fxToUsd24(params.tokenIn, params.amountIn, params.minUsd24Out);
205
206       emit UserFxToUsd24(params.user, params.tokenIn, params.amountIn, usd24Received);
207   }
```

Listing 2.5: src/DelegationVault.sol

Suggestion Revise the misleading names in the skip mechanism.

2.2.2 Revise the input name `amountOutMinimum` of the `_depositTOkenViaUsdc()` function

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `DelegationVault`, the function `_depositTOkenViaUsdc()` forwards the input `amountOutMinimum` to the function `cryptoDeposit.depositTokenViaUsdc()`, where it is interpreted the variable `amountOutMinimum` as the intermediate minimum amount (i.e., the minimum output value of USDC). If callers treat the input `amountOutMinimum` as the minimum value of the final output tokens, this mismatch can cause incorrect slippage protection and potential losses. Therefore, it is recommended to revise the input name for better readability.

```
376   function _depositTokenViaUsdc(address inputToken, address outputToken, uint256 amount, uint256
          amountOutMinimum) internal returns (uint256 outputAmount) {
377       if (inputToken == address(0) || outputToken == address(0)) revert
              DelegationVault__ZeroAddress();
378       if (amount == 0) revert DelegationVault__InvalidAmount();
379
380       _approveExact(inputToken, address(cryptoDeposit), amount);
381       outputAmount = cryptoDeposit.depositTokenViaUsdc(inputToken, outputToken, amount,
              amountOutMinimum);
382
383       emit DepositViaUsdcExecuted(inputToken, outputToken, amount, outputAmount);
384   }
```

Listing 2.6: src/DelegationVault.sol

Suggestion Revise the input name `amountOutMinimum` of the function `_depositTOkenViaUsdc()`.

2.2.3 Remove redundant code

Status Partially Fixed in `Version 2`

Introduced by `Version 1`

Description There are several unused constants and custom errors. It is recommended to remove them for better code readability. Specifically, the following code should be removed or revised.

```
31   bytes32 public constant REVERSE_ROLE = keccak256("REVERSE_ROLE");
```

Listing 2.7: src/Fiat24CardAuthorizationMarqeta.sol

```
23error Fiat24CardAuthorizationMarqeta__InsufficientAllowance(address token, uint256 required,
      uint256 allowance);
```

Listing 2.8: src/Fiat24CardAuthorizationMarqeta.sol

```
40   uint256 public constant SUNDRY = 9103;
```

Listing 2.9: src/Fiat24CardAuthorizationMarqeta.sol

```
41   uint256 public constant TREASURY = 9100;
```

Listing 2.10: src/Fiat24CardAuthorizationMarqeta.sol

```
107    error DelegationVault__VaultAccountNotOwned();
```

Listing 2.11: src/DelegationVault.sol

```
67    uint256 public constant PROXY_DELEGATION_ACCOUNT_ID = 92302;
```

Listing 2.12: src/DelegationVault.sol

Moreover, the function `depositTokenViaUsdc()` is redundant because the input token (i.e., `inputToken`) can only be USDC.

```
180    function depositTokenViaUsdc(address inputToken, address outputToken, uint256 amount, uint256
           amountOutMinimum)
```

Listing 2.13: src/DelegationVault.sol

Suggestion Remove the redundant code.

Feedback from the project The project will remove the redundant function `depositTokenViaUsdc()` in the next version.

2.2.4 Correct misleading error names

Status Partially Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `DelegationVault`, the error `DelegationVault__VaultAccountNotLive` is used for two semantically distinct failure conditions in the functions `_validateUserAccount()` and `_validateSettlementAccount()`. For example, the function `_validateSettlementAccount()` verifies the accounts' existence (i.e., `fiat24Account.exists(accountId)`), but the error name suggests the account is not live. It is recommended to introduce separate error names in the two scenarios for better readability.

```
416    function _validateUserAccount(address user) internal view {
417        uint256 tokenId = fiat24Account.historicOwnership(user);
418        if (tokenId == 0 || !fiat24Account.exists(tokenId)) {
419            revert DelegationVault__VaultAccountNotLive();
420        }
421
422        if (tokenId >= 9100 && tokenId <= 9199) {
423            revert DelegationVault__AccountNotSupport();
424        }
425
426        if (fiat24Account.status(tokenId) != 5) {
427            revert DelegationVault__VaultAccountNotLive();
428        }
429
430        _validateWalletProvider(tokenId);
431    }
432
433    function _validateSettlementAccount(uint256 accountId) internal view {
434        if (!fiat24Account.exists(accountId)) {
435            revert DelegationVault__VaultAccountNotLive();
```

```
436         }
437
438         if (accountId >= 9100 && accountId <= 9199) {
439             revert DelegationVault__AccountNotSupport();
440         }
441
442         if (fiat24Account.status(accountId) != 5) {
443             revert DelegationVault__VaultAccountNotLive();
444         }
445     }
```

Listing 2.14: src/DelegationVault.sol

Suggestion Correct the misleading error name.

Feedback from the project The project will revise the error `DelegationVault__VaultAccountNotLive` in the next version.

2.2.5 Revise the inconsistent error message in the `setAlternativeInputTokens()` function

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `Fiat24CardAuthorizationMarqeta`, the `setAlternativeInputTokens()` function performs a decimal check (i.e., `inputTokenDecimals > outputTokenDecimals`), but the error message states `"Input token decimals must be >= output token decimals"`. It is recommended to revise the inconsistent error message for better code readability.

```
499         require(inputTokenDecimals > outputTokenDecimals, "Input token decimals must be >=
                output token decimals");
```

Listing 2.15: src/Fiat24CardAuthorizationMarqeta.sol

Suggestion Revise the inconsistent error message in the function `setAlternativeInputTokens()`.

2.3 Note

2.3.1 Ensure proper off-chain parameter validations

Introduced by `Version 1`

Description In the contract `Fiat24CardAuthorizationMarqeta`, all entry functions (i.e., `authorize()`, `increment()`, `authorizeDoubleBooking()`, `incrementDoubleBooking()`, `advice()`, `reverse()`, and `reverseDoubleBooking()`) accept parameters from the role `AUTHORIZER_ROLE` with minimal on-chain validation. For instance, functions `authorize()` and `increment()` only validate the input `settlementCurrency_` against the mapping `validXXX24Tokens`, while parameters such as `tokenId_`, `cardCurrency_`, `transactionCurrency_`, `transactionAmount_`, `settlementAmount_`, and `originalPaidCurrency_` are used directly without on-chain constraints. The project must ensure the correctness and integrity of these parameters via the off-chain infrastructures.

```
194   function authorize(
195       string memory authorizationToken_,
196       string memory cardId_,
197       uint256 tokenId_,
198       address cardCurrency_,
199       string memory transactionCurrency_,
200       address settlementCurrency_,
201       uint256 transactionAmount_,
202       uint256 settlementAmount_
203   ) public {
204       if (!(hasRole(AUTHORIZER_ROLE, _msgSender()))) revert
              Fiat24CardAuthorizationMarqeta__NotAuthorizer(_msgSender());
205       if (paused()) revert Fiat24CardAuthorizationMarqeta__Suspended();
206       if (!validXXX24Tokens[settlementCurrency_]) revert
              Fiat24CardAuthorizationMarqeta__NotValidSettlementCurrency(settlementCurrency_);
207       address sender = IFiat24Account(fiat24AccountAddress).ownerOf(tokenId_);
208       address booked = IFiat24Account(fiat24AccountAddress).ownerOf(CARD_BOOKED);
209       (address paidCurrency, uint256 paidAmount) =
210           _calculatePayment(sender, cardCurrency_, transactionCurrency_, settlementCurrency_,
                  transactionAmount_, settlementAmount_);
211
212       IERC20Upgradeable(paidCurrency).safeTransferFrom(sender, booked, paidAmount == 0 ? 1 :
              paidAmount);
213       emit Authorized(authorizationToken_, tokenId_, sender, cardId_, paidCurrency, paidAmount);
214   }
```

Listing 2.16: src/Fiat24CardAuthorizationMarqeta.sol

```
265   function advice(
266       string memory authorizationToken_,
267       string memory originalAuthorizationToken_,
268       string memory cardId_,
269       uint256 tokenId_,
270       string memory transactionCurrency_,
271       address settlementCurrency_,
272       uint256 transactionAmount_,
273       uint256 settlementAmount_,
274       address originalPaidCurrency_
275   ) public {
276       if (!(hasRole(AUTHORIZER_ROLE, _msgSender()))) revert
              Fiat24CardAuthorizationMarqeta__NotAuthorizer(_msgSender());
277       if (paused()) revert Fiat24CardAuthorizationMarqeta__Suspended();
278       if (!validXXX24Tokens[settlementCurrency_]) revert
              Fiat24CardAuthorizationMarqeta__NotValidSettlementCurrency(settlementCurrency_);
279       address sender = IFiat24Account(fiat24AccountAddress).ownerOf(tokenId_);
280       address booked = IFiat24Account(fiat24AccountAddress).ownerOf(CARD_BOOKED);
281       address paidCurrency = originalPaidCurrency_; // Always pay back to the same currency
282       uint256 paidAmount;
283
284       if (validXXX24Tokens[XXX24Tokens[transactionCurrency_]]) {
285           paidAmount = transactionAmount_ * getRate(XXX24Tokens[transactionCurrency_],
                  originalPaidCurrency_)
```

```
286             * getSpread(XXX24Tokens[transactionCurrency_], originalPaidCurrency_, false) /
                    100000000;
287         } else {
288             if (settlementCurrency_ != eur24Address) revert
                    Fiat24CardAuthorizationMarqeta__DefaultSettlementCurrencyIsNotEUR(
                    settlementCurrency_);
289             paidAmount = settlementAmount_ * getRate(eur24Address, originalPaidCurrency_) *
                    getSpread(eur24Address, originalPaidCurrency_, false) / 100000000;
290         }
291
292         // Booking from #9110 to Client
293         IERC20Upgradeable(paidCurrency).safeTransferFrom(booked, sender, paidAmount);
294
295         emit Adviced(authorizationToken_, originalAuthorizationToken_, tokenId_, sender, cardId_,
                paidCurrency, paidAmount);
296     }
```

Listing 2.17: src/Fiat24CardAuthorizationMarqeta.sol

```
298     function reverse(
299         string memory authorizationToken_,
300         string memory originalAuthorizationToken_,
301         string memory cardId_,
302         uint256 tokenId_,
303         string memory transactionCurrency_,
304         address settlementCurrency_,
305         uint256 transactionAmount_,
306         uint256 settlementAmount_,
307         address originalPaidCurrency_
308     ) public {
309         if (!(hasRole(AUTHORIZER_ROLE, _msgSender()))) revert
                Fiat24CardAuthorizationMarqeta__NotAuthorizer(_msgSender());
310         if (paused()) revert Fiat24CardAuthorizationMarqeta__Suspended();
311         if (!validXXX24Tokens[settlementCurrency_]) revert
                Fiat24CardAuthorizationMarqeta__NotValidSettlementCurrency(settlementCurrency_);
312         address sender = IFiat24Account(fiat24AccountAddress).ownerOf(tokenId_);
313         address booked = IFiat24Account(fiat24AccountAddress).ownerOf(CARD_BOOKED);
314         address paidCurrency = originalPaidCurrency_; // Always pay back to the same currency
315         uint256 paidAmount =
316             _calculateReversalAmount(transactionCurrency_, settlementCurrency_, transactionAmount_,
                    settlementAmount_, originalPaidCurrency_);
317
318         // Booking from #9110 to Client
319         IERC20Upgradeable(paidCurrency).safeTransferFrom(booked, sender, paidAmount);
320
321         emit Reversed(authorizationToken_, originalAuthorizationToken_, tokenId_, sender, cardId_,
                paidCurrency, paidAmount);
322     }
```

Listing 2.18: src/Fiat24CardAuthorizationMarqeta.sol

```
324     function reverseDoubleBooking(ReverseDoubleBookingParams calldata params) public {
325         if (!(hasRole(AUTHORIZER_ROLE, _msgSender()))) revert
                Fiat24CardAuthorizationMarqeta__NotAuthorizer(_msgSender());
```

```
326         if (paused()) revert Fiat24CardAuthorizationMarqeta__Suspended();
327         if (!validXXX24Tokens[params.settlementCurrency]) revert
                Fiat24CardAuthorizationMarqeta__NotValidSettlementCurrency(params.settlementCurrency);
328         address merchant = IFiat24Account(fiat24AccountAddress).ownerOf(params.tokenId);
329         address user = IFiat24Account(fiat24AccountAddress).ownerOf(params.userTokenId);
330         address paidCurrency = params.originalPaidCurrency;
331         uint256 paidAmount = _calculateReversalAmount(
332             params.transactionCurrency, params.settlementCurrency, params.transactionAmount, params
                    .settlementAmount, params.originalPaidCurrency
333         );
334
335         _executeReverseDoubleBooking(
336             params.authorizationToken,
337             params.originalAuthorizationToken,
338             params.cardId,
339             params.tokenId,
340             params.userTokenId,
341             merchant,
342             user,
343             paidCurrency,
344             paidAmount
345         );
346     }
```

Listing 2.19: src/Fiat24CardAuthorizationMarqeta.sol

2.3.2 Proper approvals for double booking operations

Introduced by Version 1

Description In the contract `Fiat24CardAuthorizationMarqeta`, the functions `_executeDoubleBooking()` and `_executeReverseDoubleBooking()` transfers assets from both merchants and users via the function `safeTransferFrom()`. These operations require both parties to grant sufficient approvals to the contract `Fiat24CardAuthorizationMarqeta` in advance. Therefore, the project must ensure that merchants and users provide adequate approvals before invoking double booking logic to avoid invocation failures.

2.3.3 Potential centralization risks

Introduced by Version 1

Description In this project, several privileged roles (e.g., the role `CASH_OPERATOR_ROLE`) can conduct sensitive operations, which introduces potential centralization risks. 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