



Security Audit

Report for Fiat24

Date: September 12, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction	1
1.1 About Target Contracts	1
1.2 Disclaimer	1
1.3 Procedure of Auditing	2
1.3.1 Security Issues	2
1.3.2 Additional Recommendation	3
1.4 Security Model	3
Chapter 2 Findings	4
2.1 Security Issue	4
2.1.1 Circumvention of date execution restriction in the function <code>aggregate3()</code> .	4
2.2 Recommendation	5
2.2.1 Add validation in the function <code>depositTokenViaUsdcToAccount()</code>	5
2.2.2 Apply CEI pattern in functions <code>depositETHToAccount()</code>	6
2.2.3 Revise the typo	6
2.3 Note	7
2.3.1 Potential centralization risks	7
2.3.2 OpenZeppelin <code>Initializable</code> upgrade migration risks	7
2.3.3 Uniswap V3 token compatibility issues	7

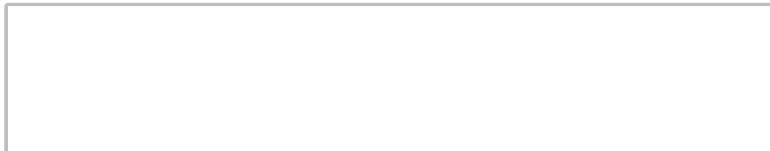
Report Manifest

Item	Description
Client	Mantle
Target	Fiat24

Version History

Version	Date	Description
1.0	September 12, 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 pluggable fiat infrastructure for Web3 projects. It streamlines the fiat money transfer and payments for both traditional and web3 native users by providing experiences that they're most comfortable with. With support for most of the major payment channels such as SEPA, SIC, SWIFT and Debit Card processings, Fiat24 results in a standard payment provider to web3 users and applications.

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

- src/Fiat24Account.sol
- src/Fiat24CryptoDeposit_Base.sol
- src/Fiat24CryptoDeposit.sol
- src/Fiat24CryptoDeposit2.sol
- src/libraries/Multicall3.sol
- src/F24Multicall.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. 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
Fiat24	Version 0	b48179b488413b5cc3fdd6cc38926d571dd75028
	Version 1	84d809ed397dbe0dd31adc4ab7fe5c5335d2302f
	Version 2	d5d39c48d281058f71d59fe9e8e26a511e98f721

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,

¹<https://github.com/mantle-xyz/fiat24contracts>

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 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 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	Medium
High	High	Medium
Low	Medium	Low
	High	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.
- **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.

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

³<https://cwe.mitre.org/>

Chapter 2 Findings

In total, we found **one** potential security issue. Besides, we have **three** recommendations and **three** notes.

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

ID	Severity	Description	Category	Status
1	Low	Circumvention of date execution restriction in the function <code>aggregate3()</code>	Security Issue	Confirmed
2	-	Add validation in the function <code>depositTokenViaUsdcToAccount()</code>	Recommendation	Fixed
3	-	Apply CEI pattern in functions <code>depositETHToAccount()</code>	Recommendation	Confirmed
4	-	Revise the typo	Recommendation	Confirmed
5	-	Potential centralization risks	Note	-
6	-	OpenZeppelin Initializable upgrade migration risks	Note	-
7	-	Uniswap V3 token compatibility issues	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Circumvention of date execution restriction in the function `aggregate3()`

Severity Low

Status Confirmed

Introduced by Version 1

Description The function `execute()` checks a date-based execution flag to prevent duplicate operations. However, the function `aggregate3()` is publicly accessible, which allows an executor to circumvent the date restriction that is enforced in the function `execute()`. This contradiction permits the same batch of calls to be executed multiple times for a date that is already marked as completed.

```
46   function execute(
47     Call3[] calldata calls,
48     string calldata dateString
49   ) external payable onlyRole(EXECUTOR_ROLE) {
50
51     require(!isDateExecuted[dateString], "SafeExecutor: Already executed for this date");
52
53     aggregate3(calls);
54
55     // Record that the execution for this date has been completed.
56     isDateExecuted[dateString] = true;
```

Listing 2.1: src/F24Multicall.sol

Impact This flaw compromises the intended single-execution-per-day guarantee, potentially leading to repeated and unauthorized state changes.

Suggestion Revise the logic accordingly.

Feedback from the project The project will never call the function `aggregate3()` directly.

2.2 Recommendation

2.2.1 Add validation in the function `depositTokenViaUsdcToAccount()`

Status Fixed in Version 2

Introduced by Version 1

Description The function `depositTokenViaUsdcToAccount()` in the contract `Fiat24CryptoDeposit_Base` does not validate if the `msg.value` is larger than 0. It is recommended to add such validation for better code readability and potential gas optimization on revert.

```

277     function depositTokenViaUsdcToAccount(address _targetAccount, address _inputToken, address
        _outputToken, uint256 _amount, uint256 _amountOutMinimum) nonReentrant payable external
        returns (uint256) {
278     if (_amount == 0) revert Fiat24CryptoDeposit__ValueZero();

```

Listing 2.2: src/Fiat24CryptoDeposit_Base.sol

Suggestion Add the validation logic accordingly.

2.2.2 Apply CEI pattern in functions `depositETHToAccount()`

Status Confirmed

Introduced by Version 1

Description The function `depositETHToAccount()` processes LayerZero message sending after refunding Ether to users, which violates the Checks-Effects-Interactions(CEI) pattern. It is recommended to apply such a pattern to follow the best practice.

```

265     (bool success, ) = msg.sender.call{value: address(this).balance - beforeBalance}("");
266     if (!success) revert Fiat24CryptoDeposit__EthRefundFailed();
267
268     if (usdcAmount == 0) revert Fiat24CryptoDeposit__SwapOutputAmountZero();
269
270     _processDepositToAccountLayerZero(_targetAccount, address(0), amountIn, usdcAmount,
        _outputToken, nativeFee);
271     emit SentDepositedEth(_targetAccount, weth, _outputToken, amountIn, usdcAmount);
272     emit DepositToAccount(_msgSender(), _targetAccount, weth, amountIn);
273     return usdcAmount;

```

Listing 2.3: src/Fiat24CryptoDeposit_Base.sol

```

268     (bool success, ) = msg.sender.call{value: address(this).balance - beforeBalance}("");
269     if (!success) revert Fiat24CryptoDeposit__EthRefundFailed();
270
271     if (usdcAmount == 0) revert Fiat24CryptoDeposit__SwapOutputAmountZero();
272
273     _processDepositToAccountLayerZero(_targetAccount, address(0), amountIn, usdcAmount,
        _outputToken, nativeFee);
274     emit SentDepositedEth(_targetAccount, weth, _outputToken, amountIn, usdcAmount);
275     emit DepositToAccount(_msgSender(), _targetAccount, weth, amountIn);
276     return usdcAmount;

```

Listing 2.4: src/Fiat24CryptoDeposit2.sol

Suggestion Apply CEI pattern in functions `depositETHToAccount()`.

2.2.3 Revise the typo

Status Confirmed

Introduced by Version 1

Description There is a typo, which affects the readability of the code and is recommended to revise it. Specifically:

In the contract `Fiat24Account`, the below revert message in the function `mintByWallet()` should be revised. The NFT is minted by another `wallet`, not the `client`.

```
123     require(!_tokenId.hasFirstDigit(INTERNALDIGIT), "9xx cannot be mint by client");
```

Listing 2.5: src/Fiat24Account.sol

Suggestion Revise code typos accordingly.

Feedback from the project The function `mintByWallet()` may be redesigned and has been temporarily reverted at the code level.

2.3 Note

2.3.1 Potential centralization risks

Introduced by Version 1

Description In this project, several privileged roles (e.g., `OPERATOR_ADMIN_ROLE`) can conduct sensitive operations, which introduces potential centralization risks. For example, `OPERATOR_ADMIN_ROLE` can set the `usdcDepositAddress` to receive the `USDC` deposited by users, based on the protocol. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

2.3.2 OpenZeppelin Initializable upgrade migration risks

Introduced by Version 1

Description The project currently uses OpenZeppelin's `Initializable` contract (v4.4.1) to implement upgradeable contracts. It is important to note that `Initializable` versions v5.0.0 and later introduce ERC-7201 namespaced storage to mitigate storage collision risks. This change relocates initialization state variables from direct storage slots (e.g., `_initialized`) to namespaced storage structures (e.g., `$_initialized`). When upgrading to newer `Initializable` versions, the project must ensure proper migration of initialization state to prevent contracts from being reinitialized, which could lead to severe security vulnerabilities including state corruption and unauthorized access.

2.3.3 Uniswap V3 token compatibility issues

Introduced by Version 1

Description Since UniswapV3 does not officially support fee-on-transfer and rebasing tokens in their router contract, it is important to note that such tokens should not be used to deposit in functions such as `depositTokenViaUsdc()`.

```
134     function depositTokenViaUsdc(address _inputToken, address _outputToken, uint256 _amount,
135                                     uint256 _amountOutMinimum) nonReentrant external returns (uint256) {
136         if (paused()) revert Fiat24CryptoDeposit__Paused();
```

```

136     if (_amount == 0) revert Fiat24CryptoDeposit__ValueZero();
137     if (!validXXX24Tokens[_outputToken]) revert Fiat24CryptoDeposit__NotValidOutputToken(
138         _outputToken);
139     uint256 tokenId = IFiat24Account(fiat24account).historicOwnership(_msgSender());
140     if (tokenId == 0) revert Fiat24CryptoDeposit__AddressHasNoToken(_msgSender());
141
142     // Transfer token from user
143     TransferHelper.safeTransferFrom(_inputToken, _msgSender(), address(this), _amount);
144
145     uint256 usdcAmount;
146     if (_inputToken != usdc) {
147         // Convert input token to USDC via Uniswap
148         TransferHelper.safeApprove(_inputToken, UNISWAP_ROUTER, _amount);
149         uint24 poolFee = getPoolFeeOfMostLiquidPool(_inputToken, usdc);
150         if (poolFee == 0) revert Fiat24CryptoDeposit__NoPoolAvailable(_inputToken, usdc);
151
152         ISwapRouter.ExactInputSingleParams memory params = ISwapRouter.ExactInputSingleParams({
153             tokenIn: _inputToken,
154             tokenOut: usdc,
155             fee: poolFee,
156             recipient: address(this),
157             deadline: block.timestamp + 15,
158             amountIn: _amount,
159             amountOutMinimum: _amountOutMinimum,
160             sqrtPriceLimitX96: 0
161         });
162
163         usdcAmount = ISwapRouter(UNISWAP_ROUTER).exactInputSingle(params);
164     } else {
165         usdcAmount = _amount;
166     }
167
168     return _processDeposit(_msgSender(), _inputToken, _outputToken, _amount, usdcAmount,
169     tokenId);
170 }
```

Listing 2.6: src/Fiat24CryptoDeposit.sol

