BLOCKSEC

Security Audit
Report for Fiat24contracts

Date: January 12, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Mantle
Target	Fiat24contracts

Version History

Version	Date	Description
1.0	January 12, 2026	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 Fiat24contracts of Mantle.

Fiat24contracts 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. In addition, Fiat24 allows users to directly convert Fiat24 tokens into USDC, to swap USDC for other tokens through integrated aggregators, and to perform cross-chain transfers with the resulting USDC.

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/BufferPool.sol
- src/BufferPoolDst.sol
- src/Fiat24PartnerGateway.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
Fiat24contracts Pull Request #77	`Version 0`	`16a6ecd837199069096ff2b919665b440796f10e`
	`Version 1`	`a8515719948e26898aca5af37653057a54b35b83`
	`Version 2`	`c1211204273ce00645abcc5ced456e03f39bc700`
Fiat24contracts Pull Request #78	`Version 0`	`5b5d3ea773344780e4483804372d1aa0e9ba6cd7`
	`Version 1`	`0d676c944c881b5ec102e3f28713b275e5be0318`
	`Version 2`	`6e0575a5aa487e01273dfe224c3b9d9442cf1b12`

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

1.2 Disclaimer

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

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

The scope of this audit is limited to the code mentioned in Section 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*	*Low*
High	High	Medium
Low	Medium	Low

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

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

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

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

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

Chapter 2 Findings

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

- High Risk: 3
- Medium Risk: 3
- Low Risk: 6
- Recommendation: 8
- Note: 7

ID	Severity	Description	Category	Status
1	High	Incorrect parameter in the function `permitAndSwapUsdcToToken()`	Security Issue	Fixed
2	High	`DoS` in cross-chain operations due to lack of `receive()` function in `Fiat24PartnerGateway`	Security Issue	Fixed
3	High	Duplicate token transfer in function `delegateOnrampAndBridge()`	Security Issue	Fixed
4	Medium	Potential denial of service via front-running the `permit()` function	Security Issue	Fixed
5	Medium	User-defined fee amount allows for protocol fee evasion	Security Issue	Fixed
6	Medium	Fees are not collected in the function `_executeOnramp()`	Security Issue	Fixed
7	Low	Lack of the function to withdraw native tokens	Security Issue	Fixed
8	Low	Incomplete execution of `CASH_OPERATOR_ROLE` protected functions by `Fiat24PartnerGateway`	Security Issue	Fixed
9	Low	Inconsistent logic in the function `delegateDeposit()` and `depositTokenViaAggregatorToAccount()`	Security Issue	Fixed
10	Low	Incorrect refund address in function `_sendCrossChain()`	Security Issue	Confirmed
11	Low	Fees are deducted even when the `feeReceiver` is set to the zero address	Security Issue	Fixed
12	Low	Improper privilege for the function `updateDstOnrampReceiver()`	Security Issue	Fixed
13	-	Revise the function `permitAndSwapUsdcToToken()`	Recommendation	Fixed
14	-	Add validation for fiat token addresses	Recommendation	Fixed
15	-	Add non zero address checks	Recommendation	Fixed

16	-	Add validation for the parameter `amount`	Recommendation	Partially Fixed
17	-	Add validation for updated parameters	Recommendation	Fixed
18	-	Remove redundant `OPERATOR_ROLE` in the contract `BufferPoolDst`	Recommendation	Confirmed
19	-	Revise the parameter `isDeposit` in the event `TokenSupportUpdated`	Recommendation	Fixed
20	-	Add the parameter `outputAmount` to the event `DepositExecuted`	Recommendation	Fixed
21	-	Implementation contracts requires manual initialization	Note	-
22	-	Manual refund upon payload decoding failure in the function `lzCompose()`	Note	-
23	-	Potential fund stuck due to insufficient `Stargate` liquidity on the target chain	Note	-
24	-	Potential centralization risks	Note	-
25	-	Integrate proper function selectors in contract `BufferPoolDst`	Note	-
26	-	The function `delegateDeposit()` does not support fee-on-transfer tokens	Note	-
27	-	Ensure consistency with supported tokens between contracts	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Incorrect parameter in the function `permitAndSwapUsdcToToken()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `BufferPoolDst` implements the function `permitAndSwapUsdcToToken()`, which is restricted to the `CASH_OPERATOR_ROLE` to facilitate asset swaps on behalf of users. The logic incorrectly passes `msg.sender` to the function `permit()` and subsequently to the function `_executeUserSwap()`, which attempts to utilize the operator's allowance and balance instead of the user's assets. Because the operator does not typically hold the user's USDC or provide the permit signature, the transaction will either revert due to an invalid signature or fail to transfer the required tokens. This fundamental logic error prevents the protocol from executing intended swaps for its users.

```
272   function permitAndSwapUsdcToToken(
273       SwapParams calldata params,
274       PermitParams calldata permit
```

```
275   ) external nonReentrant whenNotPaused onlyRole(CASH_OPERATOR_ROLE) {
276       // Execute permit
277       IERC20PermitUpgradeable(address(usdc)).permit(
278           msg.sender,
279           address(this),
280           params.usdcAmount,
281           permit.deadline,
282           permit.v,
283           permit.r,
284           permit.s
285       );
286
287       _executeUserSwap(params);
288   }
```

Listing 2.1: src/BufferPoolDst.sol

```
294   function _executeUserSwap(SwapParams calldata params) internal {
295       if (params.usdcAmount == 0 || params.usdcAmount <= params.feeAmount) revert
              BufferPoolDst__InvalidPayload();
296       if (!whitelistedAggregators[params.aggregator]) revert
              BufferPoolDst__NotWhitelistedAggregator();
297
298       bytes4 selector = bytes4(params.swapCalldata);
299       if (!whitelistedSelectors[params.aggregator][selector]) revert
              BufferPoolDst__FunctionNotWhitelisted();
300
301       // Transfer total USDC from user
302       usdc.safeTransferFrom(msg.sender, address(this), params.usdcAmount);
303
304       // Transfer fee to feeReceiver
305       if (params.feeAmount > 0 && feeReceiver != address(0)) {
306           usdc.safeTransfer(feeReceiver, params.feeAmount);
307       }
308
309       // Calculate actual swap amount after fee
310       uint256 swapAmount = params.usdcAmount - params.feeAmount;
311
312       // Get balance before
313       uint256 balanceBefore = _getTokenBalance(params.tokenOut);
314
315       // Approve USDC to aggregator
316       usdc.safeApprove(params.aggregator, 0);
317       usdc.safeApprove(params.aggregator, swapAmount);
318
319       // Execute swap via aggregator
320       (bool success,) = params.aggregator.call(params.swapCalldata);
321       if (!success) revert BufferPoolDst__SwapFailed();
322
323       // Reset approval
324       usdc.safeApprove(params.aggregator, 0);
325
326       // Calculate output amount
```

```
327         uint256 balanceAfter = _getTokenBalance(params.tokenOut);
328         uint256 amountOut = balanceAfter - balanceBefore;
329
330         // Check slippage
331         if (amountOut < params.minAmountOut) revert BufferPoolDst__SlippageExceeded();
332
333         // Transfer output token to user
334         IERC20Upgradeable(params.tokenOut).safeTransfer(msg.sender, amountOut);
335
336         emit DirectSwapExecuted(params.refId, msg.sender, address(usdc), swapAmount, params.
                tokenOut, amountOut);
337     }
```

Listing 2.2: src/BufferPoolDst.sol

Impact This fundamental logic error prevents the protocol from executing intended swaps for its users.

Suggestion Revise the logic accordingly.

2.1.2 DoS in cross-chain operations due to lack of `receive()` function in `Fiat24PartnerGateway`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `Fiat24PartnerGateway` facilitates cross-chain transfers by calling the function `onrampAndBridge()` in the `BufferPool` contract. During this execution, `BufferPool` invokes the `Stargate` protocol's `sendToken()` function, passing `msg.sender` (which is the `Fiat24PartnerGateway` contract) as the refund address for any excess native messaging fees. However, `Fiat24PartnerGateway` does not implement a `receive()` function. Consequently, when `Stargate` attempts to refund unused gas fees to the gateway contract, the transaction will revert. This creates a `DoS` for cross-chain onramp operations initiated through the gateway.

```
401     function _executeOnrampAndBridge(CrossChainOnrampParams calldata params) internal returns (
            uint256 usdcOut) {
402         // Approve BufferPool to transfer tokens from Gateway
403         IERC20Upgradeable(params.tokenIn).safeApprove(address(bufferPool), 0);
404         IERC20Upgradeable(params.tokenIn).safeApprove(address(bufferPool), params.amountIn);
405
406         // Build BufferPool CrossChainOnrampParams - user is the actual user (for fee collection
                events)
407         IBufferPool.CrossChainOnrampParams memory crossChainParams = IBufferPool.
                CrossChainOnrampParams({
408             user: params.user,
409             tokenIn: params.tokenIn,
410             amountIn: params.amountIn,
411             minAmountOut: params.minUsdcOut,
412             feeAmount: params.feeAmount,
413             dstEid: params.dstEid,
```

```
414             dstAggregator: params.dstAggregator,
415             dstTokenOut: params.dstTokenOut,
416             dstReceiver: params.dstReceiver,
417             dstSwapCalldata: params.dstSwapCalldata,
418             dstMinAmountOut: params.dstMinAmountOut,
419             dstGasLimit: params.dstGasLimit,
420             bridgeMinAmount: params.bridgeMinAmount
421         });
422
423         // Get quote to return usdcOut for event
424         usdcOut = bufferPool.getQuote(params.tokenIn, params.amountIn);
425
426         // Call BufferPool.onrampAndBridge with msg.value for cross-chain fees
427         bufferPool.onrampAndBridge{value: msg.value}(crossChainParams);
428     }
```

Listing 2.3: src/Fiat24PartnerGateway.sol

```
825         // Send via Stargate using sendToken (taxi method for composability)
826         stargate.sendToken{value: msg.value}(sendParam, fee, msg.sender);
827     }
```

Listing 2.4: src/BufferPool.sol

Impact This creates a DoS for cross-chain onramp operations initiated through the gateway.

Suggestion Revise the logic accordingly.

2.1.3 Duplicate token transfer in function `delegateOnrampAndBridge()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `delegateOnrampAndBridge()`, the contract first calls the function `safeTransferFrom()` to transfer the user tokens into this contract. However, when constructing `crossChainParams`, the `user` is set to `params.user`. As a result, when the function `bufferPool.onrampAndBridg` is invoked, a token transfer to the contract `BufferPool` is required, causing the user to transfer tokens again, which is incorrect.

```
284     function delegateOnrampAndBridge(CrossChainOnrampParams calldata params)
285         external
286         payable
287         nonReentrant
288         whenNotPaused
289         onlyRole(CASH_OPERATOR_ROLE)
290     {
291         _validateUserAccount(params.user);
292         if (params.amountIn == 0) revert Gateway__InvalidAmount();
293         if (!supportedFiat24Tokens[params.tokenIn]) revert Gateway__TokenNotSupported();
294
295         // Transfer Fiat24 token from user to Gateway
296         IERC20Upgradeable(params.tokenIn).safeTransferFrom(
```

```
297             params.user,
298             address(this),
299             params.amountIn
300         );
301
302         // Execute cross-chain onramp via BufferPool
303         uint256 usdcOut = _executeOnrampAndBridge(params);
304
305         emit OnrampAndBridgeExecuted(
306             params.user,
307             params.tokenIn,
308             params.amountIn,
309             usdcOut,
310             params.dstEid,
311             params.dstReceiver,
312             params.dstTokenOut,
313             params.partnerRefId
314         );
315     }
```

Listing 2.5: src/Fiat24PartnerGateway.sol

```
401     function _executeOnrampAndBridge(CrossChainOnrampParams calldata params) internal returns (
            uint256 usdcOut) {
402         // Approve BufferPool to transfer tokens from Gateway
403         IERC20Upgradeable(params.tokenIn).safeApprove(address(bufferPool), 0);
404         IERC20Upgradeable(params.tokenIn).safeApprove(address(bufferPool), params.amountIn);
405
406         // Build BufferPool CrossChainOnrampParams - user is the actual user (for fee collection
                events)
407         IBufferPool.CrossChainOnrampParams memory crossChainParams = IBufferPool.
                CrossChainOnrampParams({
408             user: params.user,
409             tokenIn: params.tokenIn,
410             amountIn: params.amountIn,
411             minAmountOut: params.minUsdcOut,
412             feeAmount: params.feeAmount,
413             dstEid: params.dstEid,
414             dstAggregator: params.dstAggregator,
415             dstTokenOut: params.dstTokenOut,
416             dstReceiver: params.dstReceiver,
417             dstSwapCalldata: params.dstSwapCalldata,
418             dstMinAmountOut: params.dstMinAmountOut,
419             dstGasLimit: params.dstGasLimit,
420             bridgeMinAmount: params.bridgeMinAmount
421         });
422
423         // Get quote to return usdcOut for event
424         usdcOut = bufferPool.getQuote(params.tokenIn, params.amountIn);
425
426         // Call BufferPool.onrampAndBridge with msg.value for cross-chain fees
427         bufferPool.onrampAndBridge{value: msg.value}(crossChainParams);
428     }
```

Listing 2.6: src/Fiat24PartnerGateway.sol

```
296   function onrampAndBridge(CrossChainOnrampParams calldata params)
297       external
298       payable
299       nonReentrant
300       whenNotPaused
301       onlyRole(CASH_OPERATOR_ROLE)
302   {
303       if (params.amountIn == 0) revert BufferPool__InvalidAmount();
304       _validateUserAccount(params.user);
305
306       // Transfer Fiat24 token from user
307       IERC20Upgradeable(params.tokenIn).safeTransferFrom(
308           params.user,
309           address(this),
310           params.amountIn
311       );
312
313       // Calculate USDC output and deduct fee
314       uint256 usdcOut = getQuote(params.tokenIn, params.amountIn);
315       uint256 bridgeAmount = _deductFeeAndValidate(usdcOut, params.feeAmount, params.minAmountOut
              , params.user);
316
317       _sendCrossChain(params, bridgeAmount);
318
319       emit OnrampAndBridgeInitiated(
320           params.user,
321           params.tokenIn,
322           params.amountIn,
323           bridgeAmount,
324           params.dstEid,
325           params.dstReceiver,
326           params.dstAggregator,
327           params.dstTokenOut,
328           params.dstMinAmountOut
329       );
330   }
```

Listing 2.7: src/BufferPool.sol

Impact This behavior can lead to excessive deduction of user assets.

Suggestion Revise the logic accordingly.

2.1.4 Potential denial of service via front-running the `permit()` function

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `BufferPoolDst` implements the function `permitAndSwapUsdcToToken()`, which utilizes the permit mechanism to manage approvals. An attacker can front-run the execution by invoking the function `permit()` on the token contract directly with a signature that is obtained from the mempool. This interference invalidates the signature nonce before the original call is processed, causing the function `permitAndSwapUsdcToToken()` to revert. The failure of the permit call results in a persistent denial of service for administrative swap operations.

Similar vulnerabilities exist within the contract `BufferPool` where the permit mechanism is utilized.

```
272   function permitAndSwapUsdcToToken(
273       SwapParams calldata params,
274       PermitParams calldata permit
275   ) external nonReentrant whenNotPaused onlyRole(CASH_OPERATOR_ROLE) {
276       // Execute permit
277       IERC20PermitUpgradeable(address(usdc)).permit(
278           msg.sender,
279           address(this),
280           params.usdcAmount,
281           permit.deadline,
282           permit.v,
283           permit.r,
284           permit.s
285       );
286
287       _executeUserSwap(params);
288   }
```

Listing 2.8: src/BufferPoolDst.sol

Impact The failure of the permit call results in a persistent denial of service for operations that utilize the permit mechanism.

Suggestion Revise the logic accordingly.

2.1.5 User-defined fee amount allows for protocol fee evasion

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `BufferPool` implements the function `swapUsdcToToken()`, which processes asset swaps by invoking the internal function `_executeSwap()`. The function `_executeSwap()` utilizes the `params.feeAmount` parameter that is provided directly by the user within the swap configuration. Because the logic does not validate the input against a mandatory fee rate, users can specify a zero value to evade charges. This vulnerability allows users to bypass protocol fees and is present in every function where the caller supplies the fee amount parameter.

```
546   function swapUsdcToToken(SwapParams calldata params) external nonReentrant whenNotPaused {
547       _validateUserAccount(msg.sender);
548
549       // Transfer USDC from user
```

```
550         usdc.safeTransferFrom(msg.sender, address(this), params.usdcAmount);
551
552         _executeSwap(params, msg.sender);
553     }
```

Listing 2.9: src/BufferPool.sol

```
683     function _executeSwap(SwapParams memory params, address recipient) internal returns (uint256
            amountOut) {
684         if (params.usdcAmount == 0) revert BufferPool__InvalidAmount();
685         if (!whitelistedAggregators[params.aggregator]) revert BufferPool__NotWhitelistedAggregator
                ();
686
687         bytes4 selector = bytes4(params.swapCalldata);
688         if (!whitelistedSelectors[params.aggregator][selector]) revert
                BufferPool__FunctionNotWhitelisted();
689
690         // Deduct fee (skip minAmount check here, slippage is checked on output)
691         uint256 swapAmount = _deductFeeAndValidate(params.usdcAmount, params.feeAmount, 0,
                recipient);
```

Listing 2.10: src/BufferPool.sol

Impact Users can evade protocol fees by providing a zero value for the fee amount parameter during swap operations.

Suggestion Revise the logic accordingly.

2.1.6 Fees are not collected in the function `_executeOnramp()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `_executeOnramp()`, the `feeAmount` is assigned to `OnrampParams.feeAmount` while `SwapParams.feeAmount` is set to 0. However, in the function `BufferPool.onrampAndSwap()`, the `OnrampParams.feeAmount` field is ignored. The function proceeds to calculate the USDC output and then constructs a local `SwapParams` struct using the `feeAmount` of the `SwapParams`. This will result in zero fees being collected.

```
374         // Build BufferPool OnrampParams - use Gateway as user
375         IBufferPool.OnrampParams memory onrampParams = IBufferPool.OnrampParams({
376             user: address(this), // Gateway receives the result
377             tokenIn: params.tokenIn,
378             amountIn: params.amountIn,
379             minAmountOut: params.minUsdcOut,
380             feeAmount: params.feeAmount
381         });
382
383         // Build BufferPool SwapParams
384         IBufferPool.SwapParams memory swapParams = IBufferPool.SwapParams({
385             usdcAmount: 0, // Will be set by BufferPool based on onramp output
```

```
386             feeAmount: 0, // Fee already in onrampParams
387             tokenOut: params.tokenOut,
388             minAmountOut: params.minAmountOut,
389             aggregator: params.aggregator,
390             swapCalldata: params.swapCalldata
391         });
392
393         // Call BufferPool.onrampAndSwap
394         // BufferPool transfers from Gateway, sends result to Gateway
395         bufferPool.onrampAndSwap(onrampParams, swapParams);
```

Listing 2.11: src/Fiat24PartnerGateway.sol

```
576     function onrampAndSwap(
577         OnrampParams calldata onrampParams,
578         SwapParams calldata swapParams
579     ) external nonReentrant whenNotPaused onlyRole(CASH_OPERATOR_ROLE) {
580         if (onrampParams.amountIn == 0) revert BufferPool__InvalidAmount();
581         _validateUserAccount(onrampParams.user);
582
583         // Transfer Fiat24 token from user
584         IERC20Upgradeable(onrampParams.tokenIn).safeTransferFrom(
585             onrampParams.user,
586             address(this),
587             onrampParams.amountIn
588         );
589
590         // Calculate USDC output
591         uint256 usdcOut = getQuote(onrampParams.tokenIn, onrampParams.amountIn);
592         if (usdcOut < onrampParams.minAmountOut) revert BufferPool__SlippageExceeded();
593
594         // Execute swap with the USDC output
595         SwapParams memory swapParamsWithAmount = SwapParams({
596             usdcAmount: usdcOut,
597             feeAmount: swapParams.feeAmount,
598             tokenOut: swapParams.tokenOut,
599             minAmountOut: swapParams.minAmountOut,
600             aggregator: swapParams.aggregator,
601             swapCalldata: swapParams.swapCalldata
602         });
603         _executeSwap(swapParamsWithAmount, onrampParams.user);
604
605         emit OnrampExecuted(onrampParams.user, onrampParams.tokenIn, onrampParams.amountIn, usdcOut
                );
606     }
```

Listing 2.12: src/BufferPool.sol

Impact This will result in zero fees being collected.

Suggestion Revise the logic accordingly.

2.1.7 Lack of the function to withdraw native tokens

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `delegateDeposit()` is used to allow a delegated contract to perform deposit operations. However, this function is declared as `payable`, which allows the caller to attach `msg.value` to the transaction. If a user mistakenly includes native tokens when invoking this function to deposit an `ERC20`, those native tokens will remain in the contract, since the contract `Fiat24PartnerGateway` lacks the function to withdraw native tokens.

```
182   function delegateDeposit(DepositParams calldata params)
183       external
184       payable
185       nonReentrant
186       whenNotPaused
187       onlyRole(CASH_OPERATOR_ROLE)
188   {
189       _validateUserAccount(params.user);
190       if (params.amount == 0) revert Gateway__InvalidAmount();
191
192       // Transfer tokens from user to this contract (user must have approved)
193       IERC20Upgradeable(params.inputToken).safeTransferFrom(
194           params.user,
195           address(this),
196           params.amount
197       );
198
199       // Execute deposit via Fiat24CryptoDeposit.depositTokenViaAggregator
200       uint256 outputAmount = _executeDeposit(params);
201
202       emit DepositExecuted(params.user, params.inputToken, params.amount, params.partnerRefId);
203   }
```

Listing 2.13: src/Fiat24PartnerGateway.sol

```
284   function delegateOnrampAndBridge(CrossChainOnrampParams calldata params)
285       external
286       payable
287       nonReentrant
288       whenNotPaused
289       onlyRole(CASH_OPERATOR_ROLE)
290   {
291       _validateUserAccount(params.user);
292       if (params.amountIn == 0) revert Gateway__InvalidAmount();
293       if (!supportedFiat24Tokens[params.tokenIn]) revert Gateway__TokenNotSupported();
294
295       // Transfer Fiat24 token from user to Gateway
296       IERC20Upgradeable(params.tokenIn).safeTransferFrom(
297           params.user,
298           address(this),
```

```
299             params.amountIn
300         );
301
302         // Execute cross-chain onramp via BufferPool
303         uint256 usdcOut = _executeOnrampAndBridge(params);
304
305         emit OnrampAndBridgeExecuted(
306             params.user,
307             params.tokenIn,
308             params.amountIn,
309             usdcOut,
310             params.dstEid,
311             params.dstReceiver,
312             params.dstTokenOut,
313             params.partnerRefId
314         );
315     }
```

Listing 2.14: src/Fiat24PartnerGateway.sol

Impact If a user mistakenly includes native tokens when invoking this function to deposit an ERC20, those native tokens will remain in the contract.

Suggestion Revise the logic accordingly.

2.1.8 Incomplete execution of CASH_OPERATOR_ROLE protected functions by Fiat24PartnerGateway

Severity Low

Status Fixed in Version 2

Introduced by Version 1

Description The Fiat24PartnerGateway contract is assigned the CASH_OPERATOR_ROLE in the BufferPool contract, enabling it to invoke functions that are protected by the CASH_OPERATOR_ROLE permission. However, the function permitAndOnramp() of the contract BufferPool, which is also privileged with CASH_OPERATOR_ROLE, is not invoked by the contract Fiat24PartnerGateway. As a result, the CASH_OPERATOR_ROLE holder of Fiat24PartnerGateway cannot perform all operations protected by this role.

```
251     function permitAndOnramp(
252         OnrampParams calldata params,
253         PermitParams calldata permit
254     )
255         external
256         nonReentrant
257         whenNotPaused
258         onlyRole(CASH_OPERATOR_ROLE)
259     {
260         if (params.amountIn == 0) revert BufferPool__InvalidAmount();
261         _validateUserAccount(params.user);
262
263         // Execute permit
```

```
264       IERC20PermitUpgradeable(params.tokenIn).permit(
265           params.user,
266           address(this),
267           params.amountIn,
268           permit.deadline,
269           permit.v,
270           permit.r,
271           permit.s
272       );
273
274       // Transfer Fiat24 token from user
275       IERC20Upgradeable(params.tokenIn).safeTransferFrom(
276           params.user,
277           address(this),
278           params.amountIn
279       );
280
281       // Calculate USDC output and deduct fee
282       uint256 usdcOut = getQuote(params.tokenIn, params.amountIn);
283       uint256 userAmount = _deductFeeAndValidate(usdcOut, params.feeAmount, params.minAmountOut,
              params.user);
284
285       // Transfer USDC to user
286       usdc.safeTransfer(params.user, userAmount);
287
288       emit OnrampExecuted(params.user, params.tokenIn, params.amountIn, userAmount);
289   }
```

Listing 2.15: src/BufferPool.sol

Impact As a result, the `CASH_OPERATOR_ROLE` holder of `Fiat24PartnerGateway` cannot perform all operations protected by this role.

Suggestion Revise the logic accordingly.

2.1.9 Inconsistent logic in the function `delegateDeposit()` and `depositTokenViaAggregatorToAccount()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `delegateDeposit()` is designed to allow a delegated contract to perform deposit operations. Internally, it invokes the function `depositTokenViaAggregatorToAccount()` to complete the deposit by converting the user provided token into `Fiat24`. However, the current implementation does not include any handling logic for the native token. As a result, `delegateDeposit()` cannot correctly process native token deposits, leading to inconsistent behavior compared to `depositTokenViaAggregatorToAccount()` and potentially causing unexpected execution failures.

```
182   function delegateDeposit(DepositParams calldata params)
```

```
183       external
184       payable
185       nonReentrant
186       whenNotPaused
187       onlyRole(CASH_OPERATOR_ROLE)
188   {
189       _validateUserAccount(params.user);
190       if (params.amount == 0) revert Gateway__InvalidAmount();
191
192       // Transfer tokens from user to this contract (user must have approved)
193       IERC20Upgradeable(params.inputToken).safeTransferFrom(
194           params.user,
195           address(this),
196           params.amount
197       );
198
199       // Execute deposit via Fiat24CryptoDeposit.depositTokenViaAggregator
200       uint256 outputAmount = _executeDeposit(params);
201
202       emit DepositExecuted(params.user, params.inputToken, params.amount, params.partnerRefId);
203   }
```

Listing 2.16: src/Fiat24PartnerGateway.sol

Impact This will cause the execution logic of the function `delegateDeposit()` and the function `depositTokenViaAggregatorToAccount()` to be inconsistent.

Suggestion Revise the logic accordingly.

2.1.10 Incorrect refund address in function `_sendCrossChain()`

Severity Low

Status Confirmed

Introduced by `Version 1`

Description The contract `BufferPool` implements the function `onrampAndBridge()`, which is restricted to the `CASH_OPERATOR_ROLE` and utilizes the internal function `_sendCrossChain()` to facilitate cross-chain transfers via `Stargate`. The current implementation passes the `msg.value` to cover the native messaging fee while designating the `msg.sender`, which is the operator, as the recipient for any gas refunds. This configuration creates a contradiction if the native fee is intended to be covered by the user through the `feeAmount` deduction. If the user provides the capital for the bridge fee, the refund address should logically be set to the user rather than the operator to ensure an equitable distribution of unused funds.

```
307   function onrampAndBridge(CrossChainOnrampParams calldata params)
308       external
309       payable
310       nonReentrant
311       whenNotPaused
312       onlyRole(CASH_OPERATOR_ROLE)
313   {
```

```
314     if (params.amountIn == 0) revert BufferPool__InvalidAmount();
315     _validateUserAccount(params.user);
316
317     // Transfer Fiat24 token from user
318     IERC20Upgradeable(params.tokenIn).safeTransferFrom(
319         params.user,
320         address(this),
321         params.amountIn
322     );
323
324     // Calculate USDC output and deduct fee
325     uint256 usdcOut = getQuote(params.tokenIn, params.amountIn);
326     uint256 bridgeAmount = _deductFeeAndValidate(usdcOut, params.feeAmount, params.minAmountOut
            , params.user);
327
328     _sendCrossChain(params, bridgeAmount);
```

Listing 2.17: src/BufferPool.sol

```
814 function _sendCrossChain(
815     CrossChainOnrampParams calldata params,
816     uint256 usdcOut
817 ) internal {
818     if (params.dstEid == 0) revert BufferPool__InvalidDstEid();
819
820     bytes memory composeMsg = _buildComposeMsg(params);
821     uint128 dstGas = params.dstGasLimit == 0 ? defaultDstGasLimit : params.dstGasLimit;
822
823     address receiver = dstOnrampReceivers[params.dstEid];
824     if (receiver == address(0)) revert BufferPool__ZeroAddress();
825
826     // Build extraOptions using OptionsBuilder for compose (taxi method)
827     bytes memory extraOptions = OptionsBuilder.newOptions()
828         .addExecutorLzComposeOption(0, dstGas, 0);
829
830     SendParam memory sendParam = SendParam({
831         dstEid: params.dstEid,
832         to: _addressToBytes32(receiver),
833         amountLD: usdcOut,
834         minAmountLD: params.bridgeMinAmount,
835         extraOptions: extraOptions,
836         composeMsg: composeMsg,
837         oftCmd: bytes("")
838     });
839
840     // Approve USDC to Stargate
841     usdc.safeApprove(address(stargate), 0);
842     usdc.safeApprove(address(stargate), usdcOut);
843
844     // Build fee struct - msg.value is the native fee for LayerZero messaging
845     MessagingFee memory fee = MessagingFee({
846         nativeFee: msg.value,
847         lzTokenFee: 0
```

```
848         });
849
850         // Send via Stargate using sendToken (taxi method for composability)
851         stargate.sendToken{value: msg.value}(sendParam, fee, msg.sender);
852     }
```

Listing 2.18: src/BufferPool.sol

Impact The current refund logic may cause a misallocation of funds where protocol operators retain gas refunds that were originally funded by users.

Suggestion Revise the logic accordingly.

2.1.11 Fees are deducted even when the `feeReceiver` is set to the zero address

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `BufferPool` features a function `setFeeReceiver()` with documentation stating that setting the receiver to the zero address should disable fee collection. However, the internal function `_deductFeeAndValidate()` continues to subtract the fee amount from the user's principal regardless of the receiver's status. While the logic skips the transfer when the receiver is zero, the deducted funds remain trapped in the contract balance, which may be taken as the provided liquidity. This contradiction between the documented intent and the implementation leads to unintended capital loss for users and accounting errors within the pool.

```
501     /**
502      * @notice Update fee receiver address
503      * @param _feeReceiver New fee receiver address (can be zero to disable fee collection)
504      */
505     function setFeeReceiver(address _feeReceiver) external onlyRole(OPERATOR_ADMIN_ROLE) {
506         address oldReceiver = feeReceiver;
507         feeReceiver = _feeReceiver;
508         emit FeeReceiverUpdated(oldReceiver, _feeReceiver);
509     }
```

Listing 2.19: src/BufferPool.sol

```
890     function _deductFeeAndValidate(
891         uint256 amount,
892         uint256 feeAmount,
893         uint256 minAmount,
894         address user
895     ) internal returns (uint256 netAmount) {
896         // Validate fee doesn't exceed amount
897         if (feeAmount > 0 && amount <= feeAmount) revert BufferPool__InvalidAmount();
898
899         // Calculate net amount after fee
900         netAmount = amount - feeAmount;
901         if (netAmount < minAmount) revert BufferPool__SlippageExceeded();
902
```

```
903         // Transfer fee to feeReceiver
904         if (feeAmount > 0 && feeReceiver != address(0)) {
905             usdc.safeTransfer(feeReceiver, feeAmount);
906             emit FeeCollected(user, feeAmount);
907         }
908     }
```

Listing 2.20: src/BufferPool.sol

```
434     function addLiquidity(uint256 amount) external onlyRole(LIQUIDITY_MANAGER_ROLE) {
435         usdc.safeTransferFrom(msg.sender, address(this), amount);
436         emit LiquidityAdded(msg.sender, amount);
437     }
438
439     /**
440      * @notice Remove USDC liquidity from the pool to liquidityReceiver
441      * @param amount Amount of USDC to remove
442      */
443     function removeLiquidity(uint256 amount) external onlyRole(LIQUIDITY_MANAGER_ROLE) {
444         usdc.safeTransfer(liquidityReceiver, amount);
445         emit LiquidityRemoved(liquidityReceiver, amount);
446     }
```

Listing 2.21: src/BufferPool.sol

Impact This contradiction between the documented intent and the implementation leads to unintended capital loss for users and accounting errors within the pool.

Suggestion Revise the logic accordingly.

2.1.12 Improper privilege for the function `updateDstOnrampReceiver()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `BufferPool` implements the function `updateDstOnrampReceiver()` to manage the destination chain addresses where cross-chain funds are directed. While critical administrative functions such as the function `setFeeReceiver()` are restricted to the `OPERATOR_ADMIN_ROLE`, the configuration of destination receivers is currently accessible to the lower-privileged `OPERATOR_ROLE`. This inconsistency grants excessive authority to a standard operator and deviates from the established access control patterns found within the rest of the protocol.

```
479     /**
480      * @notice Update destination onramp receiver contract (BufferPoolDst)
481      * @param _dstOnrampReceiver New receiver address
482      */
483     function updateDstOnrampReceiver(uint32 _dstEid, address _dstOnrampReceiver) external onlyRole(
            OPERATOR_ROLE) {
484         if (_dstOnrampReceiver == address(0)) revert BufferPool__ZeroAddress();
485         address oldReceiver = dstOnrampReceivers[_dstEid];
486         dstOnrampReceivers[_dstEid] = _dstOnrampReceiver;
```

```
487         emit DstOnrampReceiverUpdated(_dstEid, oldReceiver, _dstOnrampReceiver);
488     }
```

Listing 2.22: src/BufferPool.sol

Impact Assigning such sensitive configuration capabilities to a non-admin role violates the principle of least privilege and increases the risk of unauthorized redirection of cross-chain assets.

Suggestion Revise the logic accordingly.

2.2 Recommendation

2.2.1 Revise the function `permitAndSwapUsdcToToken()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `BufferPool` implements the function `permitAndSwapUsdcToToken()`, which utilizes the `permit()` function to authorize a transfer for the `msg.sender`. In general, if the user is the direct caller (`msg.sender`), the use of a signature-based permit is redundant as they could simply execute the function `approve()`.

```
560     function permitAndSwapUsdcToToken(
561         SwapParams calldata params,
562         PermitParams calldata permit
563     ) external nonReentrant whenNotPaused {
564         _validateUserAccount(msg.sender);
565 
566         IERC20PermitUpgradeable(address(usdc)).permit(
567             msg.sender,
568             address(this),
569             params.usdcAmount,
570             permit.deadline,
571             permit.v,
572             permit.r,
573             permit.s
574         );
575 
576         // Transfer USDC from user
577         usdc.safeTransferFrom(msg.sender, address(this), params.usdcAmount);
578 
579         _executeSwap(params, msg.sender);
580     }
```

Listing 2.23: src/BufferPool.sol

Suggestion Revise the logic accordingly.

2.2.2 Add validation for fiat token addresses

Status Fixed in `Version 2`

Introduced by Version 1

Description Function `withdrawFiatTokens()` allows the liquidity manager to transfer assets to a designated receiver. The logic does not validate the input token address against the authorized list of fiat tokens. This lack of restriction enables the withdrawal of any ERC20 tokens from the contract, which leads to the potential misappropriation of unintended protocol assets.

```
453   function withdrawFiatTokens(
454       address token,
455       uint256 amount
456   ) external onlyRole(LIQUIDITY_MANAGER_ROLE) {
457       IERC20Upgradeable(token).safeTransfer(liquidityReceiver, amount);
458   }
```

Listing 2.24: src/BufferPool.sol

Suggestion Revise the logic accordingly.

2.2.3 Add non zero address checks

Status Fixed in Version 2

Introduced by Version 1

Description In function `withdrawFiatTokens()`, the address variable (i.e., `token`) is not checked to ensure it is not zero. It is recommended to add such checks to prevent potential misoperations.

```
453   function withdrawFiatTokens(
454       address token,
455       uint256 amount
456   ) external onlyRole(LIQUIDITY_MANAGER_ROLE) {
457       IERC20Upgradeable(token).safeTransfer(liquidityReceiver, amount);
458   }
```

Listing 2.25: src/BufferPool.sol

In the function `initialize()`, several address variables (i.e., `_fiat24CryptoRelay`, `_fiat24CryptoDeposit`, and `_bufferPool`) are not checked to ensure they are not zero. It is recommended to add such checks to prevent potential mis-operations.

```
150   function initialize(
151       address _admin,
152       string calldata _partnerId,
153       address _fiat24Account,
154       address _bufferPool,
155       address _fiat24CryptoRelay,
156       address _fiat24CryptoDeposit,
157       address _emergencyReceiver
158   ) public initializer {
159       if (_admin == address(0) || _fiat24Account == address(0) || _emergencyReceiver == address
              (0)) {
160           revert Gateway__ZeroAddress();
161       }
162
```

```
163         __AccessControlEnumerable_init();
164         __Pausable_init();
165         __ReentrancyGuard_init();
166
167         _grantRole(DEFAULT_ADMIN_ROLE, _admin);
168         _grantRole(OPERATOR_ADMIN_ROLE, _admin);
169
170         partnerId = _partnerId;
171         fiat24Account = IFiat24Account(_fiat24Account);
172         bufferPool = IBufferPool(_bufferPool);
173         fiat24CryptoRelay = IFiat24CryptoRelay(_fiat24CryptoRelay);
174         fiat24CryptoDeposit = IFiat24CryptoDepositAggregator(_fiat24CryptoDeposit);
175         emergencyReceiver = _emergencyReceiver;
176     }
```

Listing 2.26: src/Fiat24PartnerGateway.sol

Suggestion Add non-zero address checks accordingly.

2.2.4 Add validation for the parameter `amount`

Status Partially Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `BufferPool`, several functions do not validate the input `amount` parameter. If an incorrect input is provided due to user error (i.e., `amount` is 0), it may trigger meaningless state changes or events and result in unnecessary gas consumption. The function `addLiquidity()`, `removeLiquidity()`, and `withdrawFiatTokens()` are all affected.

```
434     function addLiquidity(uint256 amount) external onlyRole(LIQUIDITY_MANAGER_ROLE) {
435         usdc.safeTransferFrom(msg.sender, address(this), amount);
436         emit LiquidityAdded(msg.sender, amount);
437     }
```

Listing 2.27: src/BufferPool.sol

```
443     function removeLiquidity(uint256 amount) external onlyRole(LIQUIDITY_MANAGER_ROLE) {
444         usdc.safeTransfer(liquidityReceiver, amount);
445         emit LiquidityRemoved(liquidityReceiver, amount);
446     }
```

Listing 2.28: src/BufferPool.sol

```
453     function withdrawFiatTokens(
454         address token,
455         uint256 amount
456     ) external onlyRole(LIQUIDITY_MANAGER_ROLE) {
457         IERC20Upgradeable(token).safeTransfer(liquidityReceiver, amount);
458     }
```

Listing 2.29: src/BufferPool.sol

Suggestion It is recommended to add validation checks to ensure the `amount` is greater than 0.

Clarification from BlockSec The function `withdrawFiatTokens()` is fixed while the functions `addLiquidity()` and `removeLiquidity()` are unfixed.

2.2.5 Add validation for updated parameters

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `BufferPool`, privileged roles can invoke various update functions to modify previously configured parameters. However, there are no checks to ensure that the new value is different from the old value (e.g., the function `updateLiquidityReceiver()`). This may result in meaningless operations and does not align with the intended semantics of update logic. The contracts `BufferPoolDst` and `Fiat24PartnerGateway` also have similar problems.

```
494   function updateLiquidityReceiver(address _liquidityReceiver) external onlyRole(
          OPERATOR_ADMIN_ROLE) {
495       if (_liquidityReceiver == address(0)) revert BufferPool__ZeroAddress();
496       address oldReceiver = liquidityReceiver;
497       liquidityReceiver = _liquidityReceiver;
498       emit LiquidityReceiverUpdated(oldReceiver, _liquidityReceiver);
499   }
```

Listing 2.30: src/BufferPool.sol

Suggestion It is recommended to add validation checks to ensure that new parameters differ from existing values before applying updates.

2.2.6 Remove redundant `OPERATOR_ROLE` in the contract `BufferPoolDst`

Status Confirmed

Introduced by `Version 1`

Description The `OPERATOR_ROLE` is defined in the contract `BufferPoolDst` and granted to the admin in the function `initialize()`. However, this role is never checked or used in any function modifiers in the contract.

```
30   bytes32 public constant OPERATOR_ROLE = keccak256("OPERATOR_ROLE");
```

Listing 2.31: src/BufferPoolDst.sol

Suggestion Remove the redundant `OPERATOR_ROLE`.

2.2.7 Revise the parameter `isDeposit` in the event `TokenSupportUpdated`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `Fiat24PartnerGateway`, the event `TokenSupportUpdated` includes a parameter named `isDeposit`. In the function `setsupportedfiat24token()`, this event is emitted

with the parameter `isDeposit` hardcoded to be `false`. It is recommended to revise the parameter `isDeposit` in the event `TokenSupportUpdated`.

```
462   function setSupportedFiat24Token(address token, bool supported) external onlyRole(
          OPERATOR_ADMIN_ROLE) {
463       if (token == address(0)) revert Gateway__ZeroAddress();
464       supportedFiat24Tokens[token] = supported;
465       emit TokenSupportUpdated(token, false, supported);
466   }
```

Listing 2.32: src/Fiat24PartnerGateway.sol

Suggestion Revise the logic accordingly.

2.2.8 Add the parameter `outputAmount` to the event `DepositExecuted`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `delegateDeposit()`, the parameter `outputAmount` is defined to receive the return value of the function `_executeDeposit()`. This parameter is not included in the event `DepositExecuted`. Meanwhile, other functions (e.g., `delegateOnramp()`) pass the returned value as a parameter and use it when emitting events.

It is recommended to include the parameter `outputAmount` in the emitted event `DepositExecuted` to record the relevant execution information.

```
182   function delegateDeposit(DepositParams calldata params)
183       external
184       payable
185       nonReentrant
186       whenNotPaused
187       onlyRole(CASH_OPERATOR_ROLE)
188   {
189       _validateUserAccount(params.user);
190       if (params.amount == 0) revert Gateway__InvalidAmount();
191
192       // Transfer tokens from user to this contract (user must have approved)
193       IERC20Upgradeable(params.inputToken).safeTransferFrom(
194           params.user,
195           address(this),
196           params.amount
197       );
198
199       // Execute deposit via Fiat24CryptoDeposit.depositTokenViaAggregator
200       uint256 outputAmount = _executeDeposit(params);
201
202       emit DepositExecuted(params.user, params.inputToken, params.amount, params.partnerRefId);
203   }
```

Listing 2.33: src/Fiat24PartnerGateway.sol

Suggestion It is recommended to include the parameter `outputAmount` in the emitted event `DepositExecuted` to record the relevant execution information.

2.3 Note

2.3.1 Implementation contracts requires manual initialization

Introduced by Version 1

Description The current implementation contracts `BufferPool`, `BufferPoolDst` and `Fiat24PartnerGateway` do not implement a mechanism to disable initializers within their constructors. Consequently, the function `initialize()` must be called manually on the logic contracts immediately after deployment to prevent malicious initialization by unauthorized parties.

```
183   function initialize(
184       address _admin,
185       address _usdc,
186       address _usd24,
187       address _stargate,
188       address _fiat24Account,
189       address _fiat24CryptoRelay,
190       address _liquidityReceiver,
191       address[] calldata _validFiat24Tokens
192   ) public initializer {
```

Listing 2.34: src/BufferPool.sol

```
133   function initialize(
134       address _admin,
135       address _usdc,
136       address _lzEndpoint,
137       address _stargateUsdc
138   ) public initializer {
```

Listing 2.35: src/BufferPoolDst.sol

```
150   function initialize(
151       address _admin,
152       string calldata _partnerId,
153       address _fiat24Account,
154       address _bufferPool,
155       address _fiat24CryptoRelay,
156       address _fiat24CryptoDeposit,
157       address _emergencyReceiver
158   ) public initializer {
159       if (_admin == address(0) || _fiat24Account == address(0) || _emergencyReceiver == address
              (0)) {
160           revert Gateway__ZeroAddress();
161       }
162
163       __AccessControlEnumerable_init();
164       __Pausable_init();
165       __ReentrancyGuard_init();
166
167       _grantRole(DEFAULT_ADMIN_ROLE, _admin);
168       _grantRole(OPERATOR_ADMIN_ROLE, _admin);
```

```
169
170         partnerId = _partnerId;
171         fiat24Account = IFiat24Account(_fiat24Account);
172         bufferPool = IBufferPool(_bufferPool);
173         fiat24CryptoRelay = IFiat24CryptoRelay(_fiat24CryptoRelay);
174         fiat24CryptoDeposit = IFiat24CryptoDepositAggregator(_fiat24CryptoDeposit);
175         emergencyReceiver = _emergencyReceiver;
176     }
```

Listing 2.36: src/Fiat24PartnerGateway.sol

2.3.2 Manual refund upon payload decoding failure in the function `lzCompose()`

Introduced by `Version 1`

Description In the contract `BufferPoolDst`, the function `lzCompose()` handles cross-chain messages decoding and token receipt. The contract attempts to decode the incoming `composeMsg` into a struct `DstSwapPayload` to retrieve execution parameters, including the recipient's address.

If the payload decoding fails, the contract emits an event `DecodeFailed`, and returns immediately. Since the recipient address is extracted from the payload, the contract cannot determine the intended recipient to perform a refund. Consequently, the `amountUsdcReceived` remains held by the contract. The project should monitor the event `DecodeFailed`, if a failure occurs, the `OPERATOR_ADMIN_ROLE` should utilize the function `emergencyWithdraw()` to recover the stuck tokens and manually refund to the user.

2.3.3 Potential fund stuck due to insufficient `Stargate` liquidity on the target chain

Introduced by `Version 1`

Description The contract `BufferPool` utilizes `Stargate` to bridge USDC and trigger the function `lzCompose()` on the destination chain. While the destination `Stargate` pool lacks liquidity, the `sendToken()` transaction on the source chain will still succeed. However, on the destination chain, `Stargate` will fail to transfer the USDC to the contract `BufferPoolDst`, resulting in the function `lzCompose()` in the contract `BufferPoolDst` not being invoked. Users' funds will remain stuck until `Stargate` liquidity recovers.

2.3.4 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., `DEFAULT_ADMIN_ROLE`) can conduct sensitive operations, which introduces potential centralization risks. For example, `OPERATOR_ADMIN_ROLE` can update the liquidity receiver address. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

2.3.5 Integrate proper function selectors in contract `BufferPoolDst`

Introduced by `Version 1`

Description In the contract `BufferPoolDst`, it is noted that only exact input swap functions should be integrated instead of exact output swap functions. Otherwise, when the contract `BufferPoolDst` interacts with aggregators via the function `_executeSwap()`, unused USDC tokens will be stuck in the contract.

2.3.6 The function `delegateDeposit()` does not support fee-on-transfer tokens

Introduced by `Version 1`

Description The function `delegateDeposit()` invokes the function `_executeDeposit()` and passes the user specified input token amount `params.amount` to the function `fiat24CryptoDeposit.depositTokenViaAggregatorToAccount()`. The contract `fiat24CryptoDeposit` proceeds the swap logic based on the `params.amount` rather than the difference of the token balance. Therefore, the contract does not support fee-on-transfer tokens deposit.

2.3.7 Ensure consistency with supported tokens between contracts

Introduced by `Version 1`

Description The contract `Fiat24PartnerGateway` uses the state variable `supportedFiat24Tokens` to record a set of valid fiat tokens. While the contract `fiat24CryptoRelay` independently uses its own state variable `validXXX24Tokens` to record a set of valid fiat tokens. To ensure functionality, it is noted to ensure the consistency of the state variables `supportedFiat24Tokens` and `validXXX24Tokens` between the contracts.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS