BLOCKSEC

Security Audit

Report for WcmAdapter Contract

Date: July 3, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Inverter Network
Target	WcmAdapter Contract

Version History

Version	Date	Description
1.0	July 3, 2026	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-07-06 03:25:46+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 WcmAdapter Contract of Inverter Network.

WcmAdapter is an incremental extension of the Bundler3 adapter system for World Markets on MegaETH. Bundler3 is built as a permissionless multicall dispatcher with adapter-based execution, transient initiator tracking, callback reentry support, and composable token movement across protocol integrations. This version introduces a dedicated adapter for the pinned USDm and wiTRY market, supporting exact-input and exact-output swaps through the World Markets router, balance-delta-based asset forwarding, bounded source-token refunds, and a Morpho debt repayment helper. The implementation also adds the required router interface and WCM-specific error handling while keeping router, chain, token, and market parameters fixed at deployment.

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/adapters/WcmAdapter.sol
- src/interfaces/IWcmAdapter.sol
- src/libraries/ErrorsLib.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
bundler3-adapters	`Version 0`	`2576d7438676a1f2a2b2648a3f5b0d53099ba676`
	`Version 1`	`6fa9631ebbeeebf04172b6cb6999ddaad3f718d1`
	`Version 2`	`a6d7e76d2d6bf5c6df94939966c8fdb42dd46557`

[1] https://github.com/InverterNetwork/bundler3-adapters

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 **one** potential security issue. Besides, we have **two** recommendations and **three** notes.

- Low Risk: 1
- Recommendation: 2
- Note: 3

ID	Severity	Description	Category	Status
1	Low	Lack of dust handling in function `_swapExactIn()`	Security Issue	Fixed
2	-	Add codehash validation in function `constructor()`	Recommendation	Fixed
3	-	Add balance checks in functions `_swapExactOut()` and `_swapExactIn()`	Recommendation	Fixed
4	-	Integration assumption	Note	-
5	-	Approve adapters as spenders	Note	-
6	-	Refund logic in function `buy()`	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Lack of dust handling in function `_swapExactIn()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In contract `WcmAdapter`, function `_swapExactIn()` forwards the full `amountIn` to the World Markets router (i.e., `ROUTER`) without ensuring that the input amount is a multiple of the router's position precision for `tokenIn`. The router accounts for USDm or wiTRY with different decimal places, corresponding to minimum raw units of 1e14 for USDm and 1e15 for wiTRY. Any remainder below the relevant minimum unit does not contribute to the swap output and is not refunded to the user. As a result, users may lose the remainder when performing exact-input swaps through the adapter.

```
44    /// @dev USDm has 4 World position decimals and 18 ERC-20 decimals.
45    uint256 internal constant USDM_WORLD_TICK = 1e14;
```

Listing 2.1: src/adapters/WcmAdapter.sol

```
176   function _swapExactIn(
177       address tokenIn,
178       address tokenOut,
179       uint256 amountIn,
180       uint256 minAmountOut,
181       address receiver,
```

```
182         uint256 deadline
183     ) internal returns (uint256 spent, uint256 received) {
184         _validateSwap(tokenIn, tokenOut, receiver, deadline);
185
186         uint256 tokenInBefore = IERC20(tokenIn).balanceOf(address(this));
187         uint256 tokenOutBefore = IERC20(tokenOut).balanceOf(address(this));
188
189         SafeERC20.forceApprove(IERC20(tokenIn), address(ROUTER), amountIn);
190
191         ROUTER.exactInputSingle(
192             IWcmSwapRouter.ExactInputSingleParams({
193                 tokenIn: tokenIn,
194                 tokenOut: tokenOut,
195                 fee: 0,
196                 recipient: address(this),
197                 deadline: deadline,
198                 amountIn: amountIn,
199                 amountOutMinimum: minAmountOut,
200                 sqrtPriceLimitX96: 0
201             })
202         );
```

Listing 2.2: src/adapters/WcmAdapter.sol

```
44    /// @dev USDm has 4 World position decimals and 18 ERC-20 decimals.
45    uint256 internal constant USDM_WORLD_TICK = 1e14;
```

Listing 2.3: src/adapters/WcmAdapter.sol

Impact When the `amountIn` carries a remainder, the vulnerability causes a direct loss of funds for the user.

Suggestion Round the `amountIn` down to the corresponding tick before the swap, and refund the remaining dust to the initiator.

2.2 Recommendation

2.2.1 Add codehash validation in function `constructor()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `WcmAdapter`, the `constructor()` only checks that the immutable variable `ROUTER_CODE_HASH` is non-zero and does not verify that `address(ROUTER).codehash == ROUTER_CODE_HASH`, as function `_validateSwap()` does. As a result, if the router address and `ROUTER_CODE_HASH` are mismatched at deployment, the issue remains undetected until the first swap reverts.

```
59    constructor(
60        address bundler3,
61        address morpho,
62        address router,
```

```
63        uint256 chainId,
64        bytes32 routerCodeHash,
65        address usdm,
66        address witry,
67        address marketOracle,
68        address marketIrm,
69        uint256 marketLltv
70    ) CoreAdapter(bundler3) {
71        require(morpho != address(0), ErrorsLib.ZeroAddress());
72        require(router != address(0), ErrorsLib.ZeroAddress());
73        require(chainId != 0, ErrorsLib.ZeroAmount());
74        require(routerCodeHash != bytes32(0), ErrorsLib.ZeroAmount());
75        require(usdm != address(0), ErrorsLib.ZeroAddress());
76        require(witry != address(0), ErrorsLib.ZeroAddress());
77        require(marketOracle != address(0), ErrorsLib.ZeroAddress());
78        require(marketIrm != address(0), ErrorsLib.ZeroAddress());
79        require(marketLltv != 0, ErrorsLib.ZeroAmount());
80        require(usdm != witry, ErrorsLib.InvalidWcmPair());
81
82        MORPHO = IMorpho(morpho);
83        ROUTER = IWcmSwapRouter(router);
84        CHAIN_ID = chainId;
85        ROUTER_CODE_HASH = routerCodeHash;
```

Listing 2.4: src/adapters/WcmAdapter.sol

```
265   function _validateSwap(address tokenIn, address tokenOut, address receiver, uint256 deadline)
          internal view {
266       require(block.chainid == CHAIN_ID, ErrorsLib.InvalidChainId());
267       require(address(ROUTER).codehash == ROUTER_CODE_HASH, ErrorsLib.InvalidWcmRouter());
```

Listing 2.5: src/adapters/WcmAdapter.sol

Suggestion Enforce `address(router).codehash == routerCodeHash` in the `constructor()`.

2.2.2 Add balance checks in functions `_swapExactOut()` and `_swapExactIn()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In contract `WcmAdapter`, functions `_swapExactIn()` and `_swapExactOut()` execute a swap via the `ROUTER`, which requires the adapter to hold sufficient balance. However, neither function checks the adapter balance. When the balance is insufficient, the transaction reverts at the `ROUTER` side, resulting in gas waste. It is recommended to check the adapter balance before invoking the `ROUTER`.

```
176   function _swapExactIn(
177       address tokenIn,
178       address tokenOut,
179       uint256 amountIn,
180       uint256 minAmountOut,
181       address receiver,
182       uint256 deadline
```

```
183   ) internal returns (uint256 spent, uint256 received) {
184       _validateSwap(tokenIn, tokenOut, receiver, deadline);
185
186       uint256 tokenInBefore = IERC20(tokenIn).balanceOf(address(this));
187       uint256 tokenOutBefore = IERC20(tokenOut).balanceOf(address(this));
188
189       SafeERC20.forceApprove(IERC20(tokenIn), address(ROUTER), amountIn);
190
191       ROUTER.exactInputSingle(
```

Listing 2.6: src/adapters/WcmAdapter.sol

```
216   function _swapExactOut(
217       address tokenIn,
218       address tokenOut,
219       uint256 amountOut,
220       uint256 maxAmountIn,
221       address receiver,
222       address refundReceiver,
223       uint256 deadline
224   ) internal returns (uint256 spent, uint256 received) {
225       _validateSwap(tokenIn, tokenOut, receiver, deadline);
226       require(refundReceiver != address(0), ErrorsLib.ZeroAddress());
227       require(refundReceiver != address(this), ErrorsLib.AdapterAddress());
228
229       uint256 tokenInBefore = IERC20(tokenIn).balanceOf(address(this));
230       uint256 tokenOutBefore = IERC20(tokenOut).balanceOf(address(this));
231
232       SafeERC20.forceApprove(IERC20(tokenIn), address(ROUTER), maxAmountIn);
233
234       ROUTER.exactOutputSingle(
```

Listing 2.7: src/adapters/WcmAdapter.sol

Suggestion Require that `IERC20(tokenIn).balanceOf(address(this))` is no less than the input amount (i.e., `amountIn` in the function `_swapExactIn()` and `maxAmountIn` in the function `_swapExactOut()`).

2.3 Note

2.3.1 Integration assumption

Introduced by `Version 1`

Description All the `Bundler3` adapters inherit from the contract `CoreAdapter`, whose functions `erc20Transfer()` and `nativeTransfer()` are guarded only by the modifier `onlyBundler3`. Any adapter balance can be claimed by anyone via a bundle. Therefore, integrators should note that every intermediate balance should be both produced and swept back to the user within the same multicall. There are two scenarios:

1. The contract `WcmAdapter` requires `tokenIn` to be transferred into the contract before a sell or buy operation. This transfer should be placed in the same multicall as the subsequent

swap. Otherwise, the balance stays in the adapter across multicalls and can be claimed by anyone via a separate bundle, resulting in a loss of funds.

2. The function `buyMorphoDebt()` rounds the debt up to the `USDM_WORLD_TICK` as `amountOut` and forwards the full amount to `receiver` (i.e., the General Adapter). As a result, the actual debt is smaller than the transferred amount, leaving a rounding surplus after repayment. The surplus should be swept back through the function `erc20Transfer()` in the same multicall.

Feedback from the project The project states that the note is an integration requirement of the `Bundler3` adapter model and confirms that integrators must ensure that any transfer into `WcmAdapter`, the corresponding WCM action, and any required sweep of leftover balances are executed within the same `Bundler3` multicall.

2.3.2 Approve adapters as spenders

Introduced by `Version 1`

Description In the contract `WcmAdapter`, the token approvals used to fund functions `sell()` and `buy()` should be granted to the adapter that pulls user funds (e.g., `GeneralAdapter1`), rather than to `Bundler3`. Contract `Bundler3` must never receive approvals since anyone can use it to invoke function `transferFrom()` directly and drain the approved funds. By contrast, adapters can only be invoked by contract `Bundler3` and always pull funds from the current initiator, ensuring that a user cannot consume others' adapter allowance.

Feedback from the project The project states that users should never approve tokens directly to `Bundler3`. For WCM flows, approvals should be granted to the adapter that pulls user funds, typically `GeneralAdapter1`, which preserves the intended allowance boundary.

2.3.3 Refund logic in function `buy()`

Introduced by `Version 1`

Description In contract `WcmAdapter`, function `buy()` passes `receiver` as the `refundReceiver` argument to the function `_swapExactOut()`, so the unspent funds are refunded to `receiver`. If the receiver is unrelated to the initiator, the unspent funds are refunded to a third party. Initiators should understand the refund logic and must set `receiver` appropriately. Notably, if the unspent funds should return to the initiator while the bought tokens are sent to the third party, set `receiver` to the initiator and separate the swap and the transfer into different actions.

```
121   /// @notice Buys an exact output amount through WCM.
122   /// @dev Tokens must have been sent to the adapter before this call. If more `tokenIn` than
          needed is present,
123   /// up to the unspent `maxAmountIn` remainder is refunded to `receiver`.
124   /// @param tokenIn Token to sell.
125   /// @param tokenOut Token to buy.
126   /// @param amountOut Exact output amount to buy.
127   /// @param maxAmountIn Maximum acceptable input amount.
128   /// @param receiver Address receiving the bought tokens.
129   /// @param deadline World router deadline.
130   function buy(
131       address tokenIn,
```

```
132         address tokenOut,
133         uint256 amountOut,
134         uint256 maxAmountIn,
135         address receiver,
136         uint256 deadline
137     ) external onlyBundler3 {
138         require(amountOut != 0, ErrorsLib.ZeroAmount());
139         require(maxAmountIn != 0, ErrorsLib.ZeroAmount());
140
141         _swapExactOut(tokenIn, tokenOut, amountOut, maxAmountIn, receiver, receiver, deadline);
142     }
```

Listing 2.8: src/adapters/WcmAdapter.sol

```
216     function _swapExactOut(
217         address tokenIn,
218         address tokenOut,
219         uint256 amountOut,
220         uint256 maxAmountIn,
221         address receiver,
222         address refundReceiver,
223         uint256 deadline
224     ) internal returns (uint256 spent, uint256 received) {
```

Listing 2.9: src/adapters/WcmAdapter.sol

Feedback from the project The project states that the `receiver` is intended to receive the full swap result in the function `buy()`, including any leftover amount. Additionally, this behavior has been documented clearly.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS