

ABRACADABRA MONEY · SECURITY REVIEW

Abracadabra Money GMX Swapper.

A security review of GMX Swapper.

CLIENT

Abracadabra
Money

PROJECT

GMX Swapper

AUDITORS

Zdravko Hristov,
0xCiphky .,
Nicholas Chew

AUDIT WINDOW

April 2 – April 7,
2025

FINAL REPORT

April 7, 2025

§ 00

Contents.

25 pages: overview, confidence ranking, scope, 16 findings, disclaimer, and about Guardian.

§ 01	Overview	P. 03
§ 02	Confidence Ranking	P. 04
§ 03	Scope & Classifications	P. 05
§ 04	Findings	P. 06
	Main Review · 15 findings	P. 07
	Remediation Review · 1 finding	P. 23
§ 05	Disclaimer	P. 24
§ 06	About Guardian	P. 25

§ 01

Overview.

Guardian reviewed GMX Swapper across 2 review rounds. The final report includes 16 Low findings.

AUDIT FIRM	REPOSITORY
Guardian	https://github.com/Abracadabra-money/abracadabra-money-contracts
CLIENT	METHODOLOGY
Abracadabra Money	Smart contract · Manual · Main Review · Remediation Review
PROJECT	
GMX Swapper	
PREPARED BY	
Zdravko Hristov, 0xCiphky ., Nicholas Chew	
AUDIT WINDOW	
April 2 – April 7, 2025	
FINAL REPORT	
April 7, 2025	

1.1 Confidence Ranking

5 / 5

Very High Confidence

Guardian assigns a Confidence Ranking of 5 out of 5, combining finding severity, remediation outcome, and the reviewer’s assessment of latent risk. The full scale is on the next page.

1	2	3	4	5
VERY LOW CONFIDENCE	LOW CONFIDENCE	MODERATE CONFIDENCE	HIGH CONFIDENCE	VERY HIGH CONFIDENCE

1.2 Findings at a Glance

SEVERITY	TOTAL	PENDING	ACK	PARTIAL	RESOLVED
■ Critical	0	0	0	0	0
■ High	0	0	0	0	0
■ Medium	0	0	0	0	0
■ Low	16	0	6	0	10
■ Informational	0	0	0	0	0
All severities	16	0	6	0	10

§ 02

Confidence Ranking.

Guardian assigns a project-level confidence score using the scale below. The ranking combines finding severity, remediation outcome, and the reviewer's assessment of latent risk.

CONFIDENCE RANKING	DEFINITION AND RECOMMENDATION	RISK PROFILE
5 Very High Confidence ASSIGNED	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. Recommendation: code is highly secure at time of audit.	0 High/Critical findings and few Low/Medium severity findings.
4 High Confidence	Code is clean, well-structured, and adheres to best practices. Design patterns are sound and test coverage is strong. Recommendation: suitable for deployment after remediations; consider periodic review with changes.	0-1 High/Critical findings per engagement week and little to no Medium severity findings.
3 Moderate Confidence	Code is functional but shows gaps in defensive programming or test coverage. Recommendation: remediate the findings and re-review the changes before deployment.	Some High/Critical findings, with various Medium severity findings.
2 Low Confidence	Recurring issues suggest systemic gaps rather than isolated mistakes. Recommendation: remediate, then commission a follow-up review at a frozen commit.	Over 6 High and Critical findings across the engagement.
1 Very Low Confidence	The volume and severity of findings indicate the codebase is not ready for production. Recommendation: substantial rework, then a full re-review.	Over 12 High and Critical findings across the engagement.

§ 03

Scope & Classifications.

Guardian reviewed GMX Swapper in <https://github.com/Abracadabra-money/abracadabra-money-contracts>.

3.1 Severity Matrix

	IMP · HIGH	IMP · MED	IMP · LOW
LIK · HIGH	Critical	High	Medium
LIK · MED	High	Medium	Low
LIK · LOW	Medium	Low	Low

IMPACT · HIGH

Significant loss of assets, harm to a group of users, or core protocol functionality is disrupted.

IMPACT · MEDIUM

Small amount of funds can be lost, or ancillary functionality is affected. Users may experience reduced or delayed receipt.

IMPACT · LOW

Unexpected behavior in some functionality that is notable but does not meet the criteria for higher severity.

LIKELIHOOD · HIGH

Possible with reasonable assumptions that mimic on-chain conditions; cost is low compared to amount gained.

LIKELIHOOD · MEDIUM

Possible only in uncommon cases or requires a large amount of capital relative to disruption caused.

LIKELIHOOD · LOW

Unlikely to ever occur in production.

§ 04

Findings.

Grouped by round. Visible IDs reset by severity within each round, so a finding is named the same way in the report as it was in the channel it was written in. Status is where each finding ended up; its own page says how.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
MAIN REVIEW					
L-01	Interface Not Inherited	Warning	LOW	ACKNOWLEDGED	p. 7
L-02	Execution Fee Param Always Zero	Optimization	LOW	RESOLVED	p. 8
L-03	Value Not Forwarded To Swapper	Warning	LOW	RESOLVED	p. 9
L-04	GMXSwapper Incompatible with GMXV2.2	Warning	LOW	ACKNOWLEDGED	p. 10
L-05	Potential DoS For Tokens Requiring Zero-Reset Approvals	DoS	LOW	RESOLVED	p. 12
L-06	Potential Reentrancy In swap Function	Warning	LOW	ACKNOWLEDGED	p. 13
L-07	Lack Of Exact Amount Tracking Before functionCall Execution	Warning	LOW	ACKNOWLEDGED	p. 14
L-08	Missing Custom Error When shareReturned < shareToMin	Best Practices	LOW	RESOLVED	p. 15
L-09	Missing Event Emissions In The swap Flow	Best Practices	LOW	ACKNOWLEDGED	p. 16
L-10	Unimplemented Function For Calculating Required Execution Fee	Best Practices	LOW	RESOLVED	p. 17
L-11	shareToMin slippage protection is not enough	Warning	LOW	ACKNOWLEDGED	p. 18
L-12	Execution Fee Param Always Zero	Warning	LOW	RESOLVED	p. 19
L-13	Swap Function Temporary DoS	DoS	LOW	RESOLVED	p. 20
L-14	Unused Variable dataStore	Best Practices	LOW	RESOLVED	p. 21
L-15	Non-Atomic swap Calls Can Lead to Fund Theft	Warning	LOW	RESOLVED	p. 22
REMEDIATION REVIEW					
L-01	Fee Underestimated for Less than 3 Oracles	Validation	LOW	RESOLVED	p. 23

LOW **L-01**

ACKNOWLEDGED

Interface Not Inherited

CATEGORY Warning
LOCATION GMSwapper.sol: 37
REVIEW Main Review

DESCRIPTION

The `GMSwapper` contract does not explicitly inherit the `ISwapperV2` interface, which is inconsistent with the other swapper implementations.

RECOMMENDATION

Consider inheriting the `ISwapper` interface.

RESOLUTION

Status: Acknowledged.

LOW **L-02**

RESOLVED

Execution Fee Param Always Zero

CATEGORY Optimization
LOCATION GMSwapper.sol
REVIEW Main Review

DESCRIPTION

The `CreateWithdrawalParams` sets the execution fee to `address(this).balance`, but the entire contract's balance was already sent to the ExchangeRouter: `exchangeRouter.sendWnt{value: address(this).balance}(withdrawalVault, address(this).balance);`

Consequently, the execution fee for the params is always 0. This is not a large issue because GMX will automatically reset the `params.executionFee` to however much ETH was sent to the ExchangeRouter.

RECOMMENDATION

Set the `executionFee` directly to 0 within the params.

RESOLUTION

Status: Resolved.

LOW **L-03**

RESOLVED

Value Not Forwarded To Swapper

CATEGORY	Warning
LOCATION	CauldronV4.sol
REVIEW	Main Review

— DESCRIPTION

For the GM withdrawal to succeed, the swapper must provide a native token execution fee to GMX. The tests pass because they send value directly through the payable `GMSwapper.swap` function to make native token available for the execution fee. However, the `swap` does not actually receive any native asset from the Cauldron. Consequently, a liquidator would need to directly send ETH to the `GMSwapper` contract before commencing liquidation, and if this is not done atomically, that ETH is at-risk of being stolen by another user.

— RECOMMENDATION

Clearly document this behavior and ensure ETH is sent to the swapper in the `cook` function prior to liquidation.

— RESOLUTION

Status: Resolved.

LOW **L-04**

ACKNOWLEDGED

GMSwapper Incompatible with GMXV2.2

CATEGORY	Warning
LOCATION	GMSwapper.sol
REVIEW	Main Review

— DESCRIPTION

The current call to `executeAtomicWithdrawal` uses the GMXv2.1 `CreateWithdrawalParams`, but in v2.2 the params are different:

```
```solidity
// Before
struct CreateWithdrawalParams {
 address receiver;
 address callbackContract;
 address uiFeeReceiver;
 address market;
 address[] longTokenSwapPath;
 address[] shortTokenSwapPath;
 uint256 minLongTokenAmount;
 uint256 minShortTokenAmount;
 bool shouldUnwrapNativeToken;
 uint256 executionFee;
 uint256 callbackGasLimit;
}
```

**L-04** GMSwapper Incompatible with GMXV2.2

CONTINUED

```
// After
struct CreateWithdrawalParams {
CreateWithdrawalParamsAddresses addresses;
uint256 minLongTokenAmount;
uint256 minShortTokenAmount;
bool shouldUnwrapNativeToken;
uint256 executionFee;
uint256 callbackGasLimit;
bytes32[] dataList;
}
...
```

Consequently, this GMSwapper will not function with GMXv2.2.\*\*

\*\*

**— RECOMMENDATION**

Be aware of this limitation for when GMX upgrades.

**— RESOLUTION**

Status: Acknowledged.

LOW **L-05**

RESOLVED

# Potential DoS For Tokens Requiring Zero-Reset Approvals

CATEGORY	DoS
LOCATION	GMSwapper.sol
REVIEW	Main Review

## — DESCRIPTION

Within the `GMSwapper` contract, certain tokens such as USDT need an explicit reset to zero before updating their allowance to a new nonzero value. Currently, the `GMSwapper` uses a check to see if the allowance is already `type(uint256).max` if not, it calls `safeApprove(..., type(uint256).max)`. However, if the preceding aggregator or external call (`swapDatum.to.functionCall(swapDatum.data)`) modifies the allowance to a non-zero amount other than `type(uint256).max`, a subsequent attempt to approve `type(uint256).max` may fail for tokens like USDT. This mismatch can cause a DoS for users attempting to swap these tokens, since the contract is unable to re-approve them without first resetting to zero.

## — RECOMMENDATION

Replace the current approval pattern with a safe retry mechanism that explicitly handles tokens requiring a zero allowance reset before setting a new nonzero allowance. This can be done using a utility function such as `[safeApproveWithRetry]`

(<https://github.com/Vectorized/solady/blob/main/src/utills/SafeTransferLib.sol#L361>) which:

- Sets the allowance to zero (if not already zero).
- Approves the target address for the desired `type(uint256).max` amount.

## — RESOLUTION

Status: Resolved.

LOW **L-06**

ACKNOWLEDGED

# Potential Reentrancy In swap Function

CATEGORY Warning  
LOCATION GMSwapper.sol  
REVIEW Main Review

## DESCRIPTION

Inside the `GMSwapper` contract, each `swapData[i]` entry specifies a target contract ( `swapDatum.to` ) and a arbitrary call data ( `swapDatum.data` ). When the code executes:

```
``solidity
swapDatum.to.functionCall(swapDatum.data);
`
```

that call can perform any function call on `swapDatum.to` . An attacker could exploit this call to grant allowances from the `GMSwapper` to himself, allowing him to transfer tokens out of the `GMSwapper` 's balance.

In future `swap` calls if the attacker receives the control during the flow, he could drain all the tokens remaining (just the `remainingBalance` , more would cause a revert) in the `GMSwapper` contract before the different legit `swapDatum.to.functionCall(swapDatum.data)` calls are completed.

## RECOMMENDATION

Considering adding the `nonReentrant` modifier to the `swap` function.

## RESOLUTION

Status: Acknowledged.

LOW **L-07**

ACKNOWLEDGED

# Lack Of Exact Amount Tracking Before functionCall Execution

CATEGORY	Warning
LOCATION	GMSSwapper.sol
REVIEW	Main Review

## — DESCRIPTION

Within the `GMSSwapper` contract's loop over `swapData[]`, the function calls:

```
``solidity
swapDatum.to.functionCall(swapDatum.data);`
```

directly after withdrawing tokens from GMX's `executeAtomicWithdrawal`, without knowing precisely how many tokens were actually received. As a result, the code and the actual caller can not precisely determine the exact token amount available for the `swap`, leaving the called aggregator contract (`swapDatum.to`) to infer or handle the balance in an ad-hoc manner. Therefore, this is an important limitation for users using the `GMSSwapper` as they do not know exactly the amounts they can operate with beforehand to properly build an optimal `swapData`.

## — RECOMMENDATION

Inform the users of this limitation as addressing this issue would require a major update to the current `GMSSwapper` code.

## — RESOLUTION

Status: Acknowledged.

LOW **L-08**

RESOLVED

# Missing Custom Error When `shareReturned < shareToMin`

CATEGORY	Best Practices
LOCATION	GMSwapper.sol
REVIEW	Main Review

## DESCRIPTION

At the end of the `GMSwapper.swap` function, the code calculates:

```
``solidity
extraShare = shareReturned - shareToMin;
`
```

This line risks an underflow if `shareReturned` (the total MIM shares actually deposited) ends up less than `shareToMin` (the user's minimum expected shares). In Solidity `^0.8.x`, this triggers an automatic underflow revert that would not clearly indicate that the user's slippage tolerance was violated.

## RECOMMENDATION

Before performing the subtraction, compare `shareReturned` and `shareToMin`. If `shareReturned < shareToMin`, revert with a clear revert message or custom error indicating that the slippage requirement was not met. For example:

```
``solidity
if (shareReturned < shareToMin) {
 revert SlippageExceeded(shareReturned, shareToMin);
}
extraShare = shareReturned - shareToMin;
``
```

## RESOLUTION

Status: Resolved.

LOW **L-09**

ACKNOWLEDGED

# Missing Event Emissions In The swap Flow

CATEGORY	Best Practices
LOCATION	GMSwapper.sol
REVIEW	Main Review

## — DESCRIPTION

Within the `GMSwapper` contract's `swap` function, there are no dedicated events emitted to indicate critical state changes such as the final `shareReturned` amount deposited in `BentoBox` or the token balances obtained after `executeAtomicWithdrawal`.

## — RECOMMENDATION

Consider emitting events for key steps:

- Post-withdrawal balances: Immediately after `executeAtomicWithdrawal`, emit an event detailing the new balance of tokens in the contract (e.g., `LogTokensReceived(fromToken, amountReceived, ...)`).
- Final MIM shares: Once the final deposit into `BentoBox` completes, emit an event such as `LogSwapResult(recipient, shareReturned, extraShare)`.

## — RESOLUTION

Status: Acknowledged.

LOW **L-10**

RESOLVED

# Unimplemented Function For Calculating Required Execution Fee

CATEGORY	Best Practices
LOCATION	GMWrapper.sol
REVIEW	Main Review

## DESCRIPTION

In the `GMWrapper`, there is a comment outlining the formula to compute the minimum execution fee for atomic withdrawals. However, this calculation is not implemented directly in any helper function. Developers or integrators must replicate the logic themselves, pulling the same parameters from `dataStore` and performing the required calculations.

## RECOMMENDATION

Create a dedicated utility function that does the work directly for the users. For example:

```
``solidity
function getMinExecutionFee(uint256 oraclePriceCount, uint256 currentGasPrice) public view
returns (uint256) {
 uint256 estimatedGasLimit = dataStore.getUint(Keys.withdrawalGasLimitKey());
 uint256 baseGasLimit = dataStore.getUint(Keys.ESTIMATED_GAS_FEE_BASE_AMOUNT_V2_1);
 baseGasLimit += dataStore.getUint(Keys.ESTIMATED_GAS_FEE_PER_ORACLE_PRICE) * oraclePriceCount;
 uint256 multiplierFactor = dataStore.getUint(Keys.ESTIMATED_GAS_FEE_MULTIPLIER_FACTOR);
 uint256 gasLimit = baseGasLimit + Precision.applyFactor(estimatedGasLimit, multiplierFactor);
 return gasLimit * currentGasPrice;
}
``
```

## RESOLUTION

Status: Resolved.

LOW **L-11**

ACKNOWLEDGED

# shareToMin slippage protection is not enough

CATEGORY	Warning
LOCATION	GMSwapper.sol
REVIEW	Main Review

## — DESCRIPTION

The `shareToMin` parameter guarantees that when the MIM tokens are deposited after all the operations back in the degenbox on the behalf of the user, the shares received will be at least `sharesToMin`. While this minimizes the risk of getting sandwiched, the degenbox share price can still change due to strategy performance. For example:

- 1000 shares and 1000 assets (1:1)
- The user specifies `sharesToMin = 100`, which is equal to `100 assets` at that time
- Before the transaction is executed, the strategy reports a loss of 500 assets and the new ratio becomes `2:1`.
- Now `sharesToMin = 100` is actually `50` assets.

## — RECOMMENDATION

Be sure to document this risk and enforce proper slippage values in your frontend for the swap data as well.

## — RESOLUTION

Status: Acknowledged.

LOW **L-12**

RESOLVED

# Execution Fee Param Always Zero

CATEGORY	Warning
LOCATION	GMSwapper.sol
REVIEW	Main Review

## DESCRIPTION

The `CreateWithdrawalParams` sets the execution fee to `address(this).balance`, but the entire contract's balance was already sent to the ExchangeRouter: `exchangeRouter.sendWnt{value: address(this).balance}(withdrawalVault, address(this).balance);`

Consequently, the execution fee for the params is always 0. This is not a large issue because GMX will automatically reset the `params.executionFee` to however much ETH was sent to the ExchangeRouter.

## RECOMMENDATION

Set the `executionFee` directly to 0 within the params.

## RESOLUTION

Status: Resolved.

LOW **L-13**

RESOLVED

# Swap Function Temporary DoS

CATEGORY	DoS
LOCATION	GMSSwapper.sol
REVIEW	Main Review

## — DESCRIPTION

The swap function allows users to perform external swaps through `functionCall`, allowing them to set custom to and data. However, a user could exploit this to reset the maximum MIM token approval granted to the box contract in the constructor. As a result, no further swaps can be executed until the approval is restored, causing a temporary denial of service.

## — RECOMMENDATION

Consider imposing restrictions on the external swap call to prevent changes to the MIM token approval on the box contract.

## — RESOLUTION

Status: Resolved.

LOW **L-14**

RESOLVED

## Unused Variable dataStore

CATEGORY	Best Practices
LOCATION	GMSwapper.sol
REVIEW	Main Review

### DESCRIPTION

The `dataStore` variable is assigned in the constructor but is only referenced in commented-out code.

### RECOMMENDATION

Remove the `dataStore` variable if it serves no purpose.

### RESOLUTION

Status: Resolved.

LOW **L-15**

RESOLVED

# Non-Atomic swap Calls Can Lead to Fund Theft

CATEGORY	Warning
LOCATION	GMSSwapper.sol
REVIEW	Main Review

## — DESCRIPTION

The `swap` function is intended to be called atomically following `Cauldron.removeCollateral` (i.e., within the same transaction). However, if executed in separate transactions, any tokens sitting idle in the swapper contract become vulnerable.

Additionally, swap relies on user-supplied `swapData` to convert withdrawn tokens into MIM. If `swapData` is misconfigured to receive a different token (e.g., not MIM), those tokens are left unhandled in the contract. Since swap allows arbitrary input data, a malicious user could simply craft a `swapData` payload to drain all tokens left behind by a prior failed or partial call.

## — RECOMMENDATION

- Document clearly that the `swap` function should only be used as part of an atomic flow.
- Consider reverting if MIM balance is zero after the swap (before `box.deposit`) to prevent misconfigured `swapData` from succeeding.

## — RESOLUTION

Status: Resolved.

LOW **L-01**

RESOLVED

# Fee Underestimated for Less than 3 Oracles

CATEGORY	Validation
LOCATION	GMSSwapper.sol
REVIEW	Remediation Review

## DESCRIPTION

The `getMinExecutionFee` function calculates the required execution fee using user-supplied `oraclePriceCount` and `currentGasPrice`. However, if `oraclePriceCount < 3`, the function will underestimate the required fee.

In practice, GMX always uses a minimum of 3 oracle prices for withdrawals, as enforced by `GasUtils::estimateWithdrawalOraclePriceCount` [source](https://github.com/gmx-io/gmx-synthetics/blob/main/contracts/gas/GasUtils.sol#L304) and validated during fee checks in `WithdrawalUtils` [source](https://github.com/gmx-io/gmx-synthetics/blob/main/contracts/withdrawal/WithdrawalUtils.sol#L128).

Accepting values below 3 in `getMinExecutionFee` can mislead integrators or frontends into providing insufficient fees, resulting in failed transactions and poor UX.

## RECOMMENDATION

Update `getMinExecutionFee` to validate that `oraclePriceCount >= 3`, or hardcode the minimum internally to align with GMX's expectations.

## RESOLUTION

Status: Resolved.

## § 05

# Disclaimer.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian's position is that each company and individual are responsible for their own due diligence and continuous security. Guardian's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis.

Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract's safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

§ 06

# About Guardian.

Guardian is a smart contract security firm. Every engagement is worked by two independent teams competing on the same scope, each with its own tooling and methods, so no single reviewer's blind spot survives.

**\$45B+**DIGITAL ASSETS UNDER  
REVIEW**200+**

ENGAGEMENTS COMPLETED

**500+**CRITICAL & HIGH FINDINGS  
REPORTED

Reach us at [audits@guardianaudits.com](mailto:audits@guardianaudits.com).