GA GUARDIAN

Abracadabra

Abracadabra BoundSpell

Security Assessment

December 16th, 2024

Summary

Audit Firm Guardian

Prepared By Robert Rodriguez, Nicholas Chew, Zdravko Hristov,

Michael Lett, Owen Thurm, Osman Ozdemir

Client Firm Abracadabra

Final Report Date December 16, 2024

Audit Summary

Abracadabra engaged Guardian to review the security of its review of their cross-chain Bound Spell staking system. From the 14th of November to the 18th of November, a team of 6 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Issues Detected Throughout the engagement 1 High/Critical issues were uncovered and promptly remediated by the Abracadabra team. Several issues impacted the fundamental behavior of the protocol, following their remediation Guardian believes the protocol to uphold the functionality described for the Abracadabra protocol.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **Crosschain**

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/abra-oft-fuzzing

Table of Contents

Project Overview

Project Summary

Project Name	Abracadabra
Language	Solidity
Codebase	https://github.com/Abracadabra-money/abracadabra-money-contracts/tree/boundspell-audit
Commit(s)	1e8b997f6187d87e104a33a4f161fd7f6120d385

Audit Summary

Delivery Date	December 16, 2024
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
High	1	0	0	0	0	1
Medium	0	0	0	0	0	0
Low	30	0	0	16	0	14

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	🔴 Critical	🟠 High	🟡 Medium
Likelihood: *Medium*	🟠 High	🟡 Medium	🟢 Low
Likelihood: *Low*	🟡 Medium	🟢 Low	🟢 Low

Impact

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

Medium A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.

Low Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.

Medium An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.

Low Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	stakeFor Function Is Not Overridden	Logical Error	High	Resolved
L-01	Missing Require Check In _updateInstantRedeemParams Function	Configuration	Low	Resolved
L-02	Centralization Risks	Centralization	Low	Acknowledged
L-03	Floating Pragma	Code Best Practices	Low	Acknowledged
L-04	BoundSpellActionSender.send Allows Sending A Zero Amount	Code Best Practices	Low	Resolved
L-05	lockingDeadline Parameter Can Be Removed	Code Best Practices	Low	Acknowledged
L-06	Incompatibility With Fee-On-Transfer Tokens	DoS	Low	Acknowledged
L-07	TokenLocker._redeemFor Function Claims To The To Address	Configuration	Low	Acknowledged
L-08	Lack Of A Double Step TransferOwnership Pattern	Code Best Practices	Low	Acknowledged
L-09	Unused Imports	Code Best Practices	Low	Resolved
L-10	MultiRewards Contract Only Supports 18 Decimal Tokens	Logical Error	Low	Resolved
L-11	Unclear Process For Handling Failed Messages On Destination Chain	Configuration	Low	Acknowledged
L-12	Use Of An Outdated Version Of Solady's safeTransferLib	Code Best Practices	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-13	Uncleared lastLockIndex	Logical Error	Low	Resolved
L-14	Funds Held Hostage With Lockup Period	Configuration	Low	Acknowledged
L-15	Unnecessary Parameter Data	Code Best Practices	Low	Resolved
L-16	Warning About minDstGasLookup	Code Best Practices	Low	Acknowledged
L-17	Lacking Constant Usage	Code Best Practices	Low	Resolved
L-18	Unused Error	Code Best Practices	Low	Resolved
L-19	Can't Send Native Token With CREATE3	Configuration	Low	Resolved
L-20	Late Redeem Is Incentivized	Configuration	Low	Acknowledged
L-21	DOS If Custom Params Are Disabled	DoS	Low	Acknowledged
L-22	Reward Tokens Are Not Removable	Configuration	Low	Acknowledged
L-23	Hardcoded LZ Params	Configuration	Low	Acknowledged
L-24	Hardcoded LOCAL_CHAIN_ID	Configuration	Low	Acknowledged
L-25	getRewardsFor Receiver Is Hardcoded	Configuration	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-26	Potential Revert On Zero Transfer	Configuration	Low	Resolved
L-27	Ether May Be Stuck In Contract	Logical Error	Low	Resolved
L-28	Layerzero Dust Issues	Configuration	Low	Acknowledged
L-29	Unclaimed Dust In MultiRewards Contract	Code Best Practices	Low	Acknowledged
L-30	Refund Address Should Be Configurable	Logical Error	Low	Resolved

H-01 | stakeFor Function Is Not Overridden

Category	Severity	Location	Status
Logical Error	High	SpellPowerStaking.sol	Resolved

Description

In the SpellPowerStaking contract there is a lockup period where users are unable to exit their stakes if they have lastAdded to their stake within the lockupPeriod.

However the lockup logic is only enforced for the stake function, and not for the stakeFor function as this is not overridden from the MultiRewards contract.

Therefore users who stake through the cross-chain staking system which uses the BoundSpellActionReceiver as an operator to stake for a user address will not have their stakes locked for the lockupPeriod.

Furthermore the whenNotPaused modifier is not applied to the stakeFor function as it is not overridden, so stakes through the cross-chain mechanism can still occur when the system is paused.

Recommendation

Override the stakeFor function in the SpellPowerStaking contract and apply the lastAdded storage write as well as the whenNotPaused modifier.

Resolution

Abracadabra Team: The issue was resolved in commit 0a684a7.

L-01 | Missing Require Check In _updateInstantRedeemParams Function

Category	Severity	Location	Status
Configuration	Low	TokenLocker.sol: 309	Resolved

Description

The `_updateInstantRedeemParams` function in the `TokenLocker` contract updates the parameters for instant redemption, including the `feeCollector` address. However, there is no require check to ensure that the `feeCollector` address is not set to `address(0)`.

If the `feeCollector` is set to `address(0)`, it can cause a denial of service in the insta-redemption process, as the `instantRedeem` function checks for a non-zero `feeCollector` address before proceeding.

Recommendation

Consider adding a require check in the `_updateInstantRedeemParams` function that enforces that the `feeCollector` address set is not the `address(0)`.

Resolution

Abracadabra Team: The issue was resolved in commit 05c994f.

L-02 | Centralization Risks

Category	Severity	Location	Status
Centralization	Low	MultiRewards.sol, SpellPowerStaking.sol, TokenLocker.sol	Acknowledged

Description

The contracts `TokenLocker`, `MultiRewards`, and `SpellPowerStaking` contain certain functions and permissions that could lead to centralization risks.

These risks arise from the ability of privileged roles (such as the owner or operators) to perform actions that could potentially be detrimental to the interests of regular users.

In the `MultiRewards` contract:
• Owner can never recover the `stakingToken` however he can recover any `rewardToken`.
• `getRewardsFor` function could be abused by an operator to send the accrued rewards to himself.
In the `SpellPowerStaking` contract:
• Owner can extend the `lockupPeriod` indefinitely.
In the `TokenLocker` contract:
• Operators have total control over the tokens within the contract. They are free to `redeemFor`, `instantRedeemFor` and `claimFor` on behalf of any user.

Recommendation

These centralization risks are noted to provide a clear understanding of the control dynamics within the contracts. No specific recommendations are provided as the intention is to document these aspects for awareness and consideration by stakeholders.

Resolution

Abracadabra Team: Acknowledged.

L-03 | Floating Pragma

Category	Severity	Location	Status
Code Best Practices	Low	Global	Acknowledged

Description

Contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma.

For example, an outdated pragma version might introduce bugs that affect the protocol negatively. All the contracts in scope are using the following floating pragma: `pragma solidity >=0.8.0;`

Recommendation

Consider locking the pragma version in all the smart contracts. It is not recommended to use a floating pragma in production. For example: `pragma solidity 0.8.20`.

Resolution

Abracadabra Team: Acknowledged.

L-04 | BoundSpellActionSender.send Allows Sending A Zero Amount

Category	Severity	Location	Status
Code Best Practices	Low	BoundSpellCrosschainActions.sol: 105	Resolved

Description

The `send` function in the `BoundSpellActionSender` contract allows users to initiate cross-chain actions with a zero amount. This can lead to unnecessary transactions that consume gas without performing any meaningful operation.

Recommendation

Add a require check in the `BoundSpellActionSender.send` function that enforces that the `amount` sent is higher than 0.

Resolution

Abracadabra Team: The issue was resolved in commit 4f50589.

L-05 | lockingDeadline Parameter Can Be Removed

Category	Severity	Location	Status
Code Best Practices	Low	TokenLocker.sol	Acknowledged

Description

In the `TokenLocker` contract the `lockingDeadline` parameter is passed to the `redeem` and `_redeemFor` functions but it does not seem to influence any logic within these functions. The parameter is not used to enforce any constraints, which makes it redundant in its current form.

The only check performed is within the `_createLock` function which simply enforces that the provided `lockingDeadline` is higher or equal to the current `block.timestamp`.

Recommendation

Consider removing the `lockingDeadline` parameter from the `TokenLocker` contract.

Resolution

Abracadabra Team: Acknowledged.

L-06 | Incompatibility With Fee-On-Transfer Tokens

Category	Severity	Location	Status
DoS	Low	Global	Acknowledged

Description

All the contracts within scope, upon a token transfer, always assume that the full amount specified will be transferred to the recipient. However, fee-on-transfer tokens deduct a fee from the transfer amount, resulting in the recipient receiving less than expected.

For example, tokens like USDT, while currently not implementing transfer fees, have the capability to do so. If a reward token with fees is ever added to the system, it could cause a total denial of service for the reward distribution process, as the expected and actual transferred amounts would not match.

Recommendation

Ensure that the contracts do not operate with fee-on-transfer tokens. In order to support this type of tokens, consider checking the balance before and after the token transfer to determine the total amount transferred after the fees were applied.

Resolution

Abracadabra Team: Acknowledged.

L-07 | TokenLocker._redeemFor Function Claims To The To Address

Category	Severity	Location	Status
Configuration	Low	TokenLocker.sol: 208	Acknowledged

Description

In the `TokenLocker` contract, the `_redeemFor` function is responsible for redeeming a specified amount of tokens. However, before creating a new lock for the redeemed amount, the function calls `_claim`, which releases any accrued tokens to the `to` address.

This behavior may not be expected by users who intend to redeem a specific amount to a particular address without affecting any accrued tokens.

Recommendation

Consider documenting the behavior of the `_redeemFor` function, informing users that it will also claim any accrued tokens to the `to` address. This will help users understand the full implications of using the function.

Resolution

Abracadabra Team: Acknowledged.

L-08 | Lack Of A Double Step TransferOwnership Pattern

Category	Severity	Location	Status
Code Best Practices	Low	MultiRewardsClaimingHandler.sol: 15, TokenLocker.sol: 16, BoundSpellCrosschainActions.sol: 36, BoundSpellCrosschainActions.sol: 186	Acknowledged

Description

The current ownership transfer process for all the contracts inheriting from the `OwnableOperators` contract involves the current owner calling the `transferOwnership` function:

```
function transferOwnership(address newOwner) external onlyOwner {owner = newOwner; emit OwnershipTransferred(msg.sender, newOwner);}
```

If the nominated EOA account is not a valid account, it is entirely possible that the owner may accidentally transfer ownership to an uncontrolled account, losing the access to all functions with the `onlyOwner` modifier.

Recommendation

It is recommended to implement a two-step process transfer ownership process where the owner nominates an account and the nominated account needs to call an `acceptOwnership` function for the transfer of the ownership to fully succeed.

This ensures the nominated EOA account is a valid and active account. This can be easily achieved by using OpenZeppelin's Ownable2Step contract.

Resolution

Abracadabra Team: Acknowledged.

L-09 | Unused Imports

Category	Severity	Location	Status
Code Best Practices	Low	TokenLocker.sol: 11	Resolved

Description

In the `TokenLocker` contract, the `MathLib` library is imported but not utilized anywhere within the contract.

Recommendation

Consider removing the import statement for the `MathLib` library in the `TokenLocker` contract.

Resolution

Abracadabra Team: The issue was resolved in commit 2d0e09e.

L-10 | MultiRewards Contract Only Supports 18 Decimal Tokens

Category	Severity	Location	Status
Logical Error	Low	MultiRewards.sol: 114, MultiRewards.sol: 122	Resolved

Description

The `MultiRewards` contract assumes that all staking tokens have 18 decimals, as indicated by the use of `1e18` in calculations for rewards:

```
function rewardPerToken(address rewardToken) public view returns
(uint256) {if (totalSupply == 0) {return
_rewardData[rewardToken].rewardPerTokenStored;}

uint256 timeElapsed = lastTimeRewardApplicable(rewardToken) -
_rewardData[rewardToken].lastUpdateTime

uint256 pendingRewardsPerToken = (timeElapsed *
_rewardData[rewardToken].rewardRate * 1e18) / totalSupply;

return _rewardData[rewardToken].rewardPerTokenStored +
pendingRewardsPerToken;}
```

This assumption will lead to incorrect reward calculations if tokens with different decimal places are used.

Recommendation

Consider adding a check in the `addReward` function that enforces that the staking token used has 18 decimals.

Resolution

Abracadabra Team: The issue was resolved in commit e419407.

L-11 | Unclear Process For Handling Failed Messages On Destination Chain

Category	Severity	Location	Status
Configuration	Low	MultiRewardsClaimingHandler.sol, BoundSpellCrosschainActions.sol	Acknowledged

Description

In the `MultiRewardsClaimingHandler` and `BoundSpellCrosschainActions` contracts, `LayerZero/OFT` calls are used to enable cross-chain operations. However, the procedure for managing scenarios where these messages fail to execute on the destination chain is not well-documented.

This can result in some uncertainty regarding who is responsible for retrying or addressing failed transactions, whether it should be handled by a backend system or individually by the users.

Recommendation

Provide comprehensive documentation outlining the process for handling failed messages on the destination chain. Ensure that failed messages occur only under temporary circumstances, such as when the destination contract is paused. Permanent failed messages should never occur.

Resolution

Abracadabra Team: Acknowledged.

L-12 | Use Of An Outdated Version Of Solady's safeTransferLib

Category	Severity	Location	Status
Code Best Practices	Low	Global	Resolved

Description

Multiple contracts within the system rely on Solady's `SafeTransferLib` library for token transfer operations. The version in use, as specified in the `soldeer.lock` file, is:

```
[[dependencies]]
name     = "solady"
version  = "0.0.231"
source   = "https://github.com/Vectorized/solady.git"
checksum = "2907b5036b9b3489891e69b49adba24c793940d4"
```

This version of Solady is outdated. In a more recent update, introduced via PR#1128, an important security improvement was added to `SafeTransferLib`.

The updated version includes a validation step to ensure that the token being transferred is a valid contract address with actual code.

By using an outdated Solady's version, the contracts miss out on this important safeguard. Upgrading to the latest version of Solady is recommended to enhance the security of any token transfer.

Recommendation

Consider using the latest Solady's version.

Resolution

Abracadabra Team: The issue was resolved in commit 2a7cde1.

L-13 | Uncleared lastLockIndex

Category	Severity	Location	Status
Logical Error	Low	TokenLocker.sol: 252-271	Resolved

Description

In the `TokenLocker` contract, `_releaseLocks` function iterates through all the locks, releases the unlocked ones and updates the `lastLockIndex`.

However, if all locks are released at the same time (e.g. a user waits long enough to claim all at once), this line will never hit, the `lastLockIndex` will not be updated and it will retain the previous lock count, even though the user no longer has any locks left.

Recommendation

Consider resetting `lastLockIndex` to 0 if the `_userLocks[user].length` becomes 0 after releasing locks.

Resolution

Abracadabra Team: The issue was resolved in commit 74f1b9d.

L-14 | Funds Held Hostage With Lockup Period

Category	Severity	Location	Status
Configuration	Low	SpellPowerStaking.sol: 49	Acknowledged

Description

The `setLockupPeriod` function can be used to extend the `lockupPeriod` without bound by the owner. This can be used to trap and hold hostage the funds of users who are deposited in the `SpellPowerStaking` contract.

This may become a risk in the event that the owner address is compromised. Instead the configured lockup period could be limited to reduce this risk.

Recommendation

Consider adding an upper bound for the lockup period in the `setLockupPeriod` function.

Resolution

Abracadabra Team: Acknowledged.

L-15 | Unnecessary Parameter Data

Category	Severity	Location	Status
Code Best Practices	Low	BoundSpellCrossChainActions.sol	Resolved

Description

In the `MINT_AND_STAKE_BOUNDSPELL` action flow the `MintBoundSpellAndStakeParams` which is used includes a `rewardHandlerParams` entry where the `data` and `value` values are always hardcoded to `""` and `0`.

These `rewardHandlerParams` values are never used during the execution of the mint and stake and therefore can be removed.

Recommendation

Consider removing the `rewardHandlerParams` from the `MintBoundSpellAndStakeParams`.

Resolution

Abracadabra Team: The issue was resolved in commit 6a9e62e.

L-16 | Warning About minDstGasLookup

Category	Severity	Location	Status
Code Best Practices	Low	Global	Acknowledged

Description

In the `notifyRewards` function of `MultiRewardsClaimingHandler` contract, user provided `params.gas amount` is passed as `gasLimit` for the execution in the destination chain.

This amount is then later checked in the `LzApp._checkGasLimit`, and it has to be greater than or equal to `minDstGasLookup[_dstChainId][_type]`. Even the non-blocking messaging channels can be blocked if the `minDstGasLookup` value is not sufficient enough.

This can happen if the transaction reverts with Out Of Gas error before reaching the non-blocking app try/catch (e.g. somewhere here). As a result, failed transaction will be stored in `Endpoint.storedPayload` instead of `LzNonblockingApp.failedMessages`.

Recommendation

Ensure that the `minDstGasLookup` value is set sufficiently high to prevent failed messages from blocking the channel.

Resolution

Abracadabra Team: Acknowledged.

L-17 | Lacking Constant Usage

Category	Severity	Location	Status
Code Best Practices	Low	BoundSpellCrosschainActions.sol: 88	Resolved

Description

In the `estimate` function the packet type provided to the `minDstGasLookup` function is hardcoded as the literal 1.

Recommendation

Consider using the `PT_SEND_AND_CALL` constant value to follow best practices and avoid magic numbers.

Resolution

Abracadabra Team: The issue was resolved in commit 29f7b10.

L-18 | Unused Error

Category	Severity	Location	Status
Code Best Practices	Low	MultiRewards.sol: 42	Resolved

Description

`ErrInvalidRewardHandler` error in the `MultiRewards` contract is defined but never used.

Recommendation

Consider removing the error, or alternatively perform an input check during the `setRewardHandler` function and use the defined error.

Resolution

Abracadabra Team: The issue was resolved in commit ceab273.

L-19 | Can't Send Native Token With CREATE3

Category	Severity	Location	Status
Configuration	Low	Create3Factory.sol	Resolved

Description

The `Create3Factory.deploy()` function accepts a `value` parameter to use when deploying the new contract. However, there are no `payable`, `receive` or `fallback` functions in the `Create3Factory` contract which makes it impossible to send native token for deployment.

Technically, it's still possible to send native token to the contract outside of the EVM, but anyone will be able to frontrun the deployment transaction and steal it by deploying their own contract and passing the balance of the `Create3Deployer` as `value`.

In result, deployments which require sending funds will always fail.

Recommendation

Make the `deploy` function `payable`.

Resolution

Abracadabra Team: The issue was resolved in commit 9ff6ca1.

L-20 | Late Redeem Is Incentivized

Category	Severity	Location	Status
Configuration	Low	TokenLocker.sol	Acknowledged

Description

When users redeem, they burn their issued tokens and wait a certain amount of time to receive back the underlying token. They must wait for the current epoch to end plus the `lockupPeriod`.

This means that the later they redeem their tokens in the given epoch, the less they will have to wait compared to users who redeemed earlier in the same epoch.

Recommendation

Consider documenting this behavior for users.

Resolution

Abracadabra Team: Acknowledged.

L-21 | DOS If Custom Params Are Disabled

Category	Severity	Location	Status
DoS	Low	MultiRewardsClaimingHandler.sol	Acknowledged

Description

`MultiRewardsClaimingHandler.notifyRewards()` populates the `adapterParams` field and then sends a message by calling `sendFrom()`.

The `sendFrom` function will revert if the OFT's `useCustomAdapterParams` variable is set to false and users won't be able to claim their rewards to another chains.

Recommendation

Be sure to never toggle the `useCustomAdapterParams`, as it's currently set to true.

Resolution

Abracadabra Team: Acknowledged.

L-22 | Reward Tokens Are Not Removable

Category	Severity	Location	Status
Configuration	Low	MultiRewards.sol	Acknowledged

Description

The `rewardTokens` added to the `MultiRewards` contract can only be added and not removed. This is dangerous because:
• the array may become too large and DOS the system.
• one of the tokens may start reverting causing DOS of the system

Recommendation

Be aware of the risk.

Resolution

Abracadabra Team: Acknowledged.

L-23 | Hardcoded LZ Params

Category	Severity	Location	Status
Configuration	Low	MultiRewardsClaimingHandler.sol: 88, BoundSpellCrosschainActions.sol: 96	Acknowledged

Description

As recommended by the `LayerZero` integration checklist point 5 & 6 (https://docs.layerzero.network/v1/developers/evm/evm-guides/advanced/relayer-adapter-parameters) the param for `zroPaymentAddress` and `useZro` should not be hardcoded.

This prevents the ability of using the ZRO token as a fee payment option in the future.

Recommendation

Consider passing in an input parameter for the `zroPaymentAddress` and `useZro` field.

Resolution

Abracadabra Team: Acknowledged.

L-24 | Hardcoded LOCAL_CHAIN_ID

Category	Severity	Location	Status
Configuration	Low	MultiRewardsClaimingHandler.sol	Acknowledged

Description

The `LOCAL_CHAIN_ID` for the current chain in `MultiRewardsClaimingHandler` is hardcoded to `0`. If the user specifies 0 as `dstChainId`, the reward tokens will be simply transferred to them on the current chain.

Typically, there is no reason for a user to go through that flow if they want to claim the reward tokens on the same chain, since they can do it by calling `MultiRewards.getRewards()` without passing any arguments.

If for some reason an external integrator always call `getRewards` with parameters, it makes sense that when they want to claim on the same chain, they will pass its layer zero chain id as `dstChainId` and not `0`.

This will not be caught by the current if statement and the transaction will probably revert because `minDstGasLookup` for the current chain will not be set in the `OFT`. Because of this, the user will not be able to claim their tokens.

Recommendation

Consider having the `LayerZero` id on each chain as `LOCAL_CHAIN_ID` instead of hardcoding it to 0. You can additionally check for 0 as well if it's needed for some other integrations.

Resolution

Abracadabra Team: Acknowledged.

L-25 | getRewardsFor Receiver Is Hardcoded

Category	Severity	Location	Status
Configuration	Low	MultiRewardsClaimingHandler.sol	Resolved

Description

When a user calls `_getRewardsFor()` with `RewardHandlerParams`, their rewards will be sent to the `MultiRewardsClaimingHandler` contract which will send them to the desired destination chain.

The receiver there will be the same address as the sender on the source chain. If the user doesn't have the same address on the other chain - because it's a smart contract or the other chain is not an EVM one, they will lose access to their rewards.

Recommendation

Consider letting the user pass a receiver address for the destination chain

Resolution

Abracadabra Team: The issue was resolved in commit 4bcb5df.

L-26 | Potential Revert On Zero Transfer

Category	Severity	Location	Status
Configuration	Low	MultiRewards.sol	Resolved

Description

The `MultiRewards._getRewardsFor()` function doesn't check if the current reward of the user for a given token is positive. This means a 0 transfer will be initiated for any `rewardToken` that hasn't accrued.

If one of these tokens revert on 0 amount transfers, the given user will not be able to claim any rewards using that function until they accrue these rewards.

Recommendation

Consider executing a `continue` if the accrued rewards are 0 in both `_getRewardsFor` and `MultiRewardsClaimingHandler.notifyRewards()`

Resolution

Abracadabra Team: The issue was resolved in commit 7563dfc.

L-27 | Ether May Be Stuck In Contract

Category	Severity	Location	Status
Logical Error	Low	MultiRewards.sol: 280	Resolved

Description

In `_getRewardsFor`, the `rewardHandler` contract is called as follows:

```
rewardHandler.notifyRewards{value: params.value}(user, _rewards, params.data)
```

However, `params.value` is user-defined and may differ from `msg.value`, leading to potential excess ETH left in the contract that cannot be retrieved. Furthermore, in `notifyRewards` if `msg.value > param.fee + param.gas` then excess ether is also stuck in contract.

Recommendation

Consider passing in `msg.value` instead of `params.value`:

```
rewardHandler.notifyRewards{value: msg.value}
```

Resolution

Abracadabra Team: The issue was resolved in commit 4be7a94.

L-28 | Layerzero Dust Issues

Category	Severity	Location	Status
Configuration	Low	MultiRewardsClaimingHandler.sol	Acknowledged

Description

When `MultiRewardsClaimingHandler` sends token to the destination chain, layerzero will not use all the tokens, but leave a dust amount of tokens in the contract, depending on the configured OFT decimals.

Since there is no refund mechanism once the tokens enter the `MultiRewardsClaimingHandler`, users will always be losing these dust amounts.

The same problem exists in `BoundSpellCrosschainActions._sendMintAndStakeBoundSpell` and `BoundSpellCrosschainActions._sendStakeBoundSpell`

Recommendation

Consider implementing a mapping with `dustAmounts` that the users can claim.

Resolution

Abracadabra Team: Acknowledged.

L-29 | Unclaimed Dust In MultiRewards Contract

Category	Severity	Location	Status
Code Best Practices	Low	MultiRewards.sol	Acknowledged

Description

The `MultiRewards` contract may experience a minor accumulation of unclaimed reward tokens (dust) due to the inherent precision loss in Solidity's integer arithmetic.

When calculating `rewardPerToken` and users' earned rewards, minimal precision loss occurs because of integer division truncating fractional values.

This precision loss results in a small amount of rewards not being distributed, thus remaining in the contract. Over time, these minimal unclaimed amounts can accumulate within the contract, remaining inaccessible to users.

Recommendation

Consider utilizing the `recover` function to retrieve any accumulated dust if it becomes significant for any of the reward tokens.

Resolution

Abracadabra Team: Acknowledged.

L-30 | Refund Address Should Be Configurable

Category	Severity	Location	Status
Logical Error	Low	MultiRewardsClaimingHandler.sol: 87	Resolved

Description

In `notifyRewards`, the user receiving rewards is also set as the `refundAddress`. However, if the user is a contract without a `receive` function then the refund mechanism will fail and prevent rewards from being claimed.

Furthermore, if `notifyRewards` was called directly by an Operator (who pays the gas and fees), then the Operator should receive the refund instead of the user.

Recommendation

Consider allowing a function input for `refundAddress` instead of assuming it should be `user`.

Resolution

Abracadabra Team: The issue was resolved in commit 147db99.

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.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits