

ABRACADABRA MONEY · SECURITY REVIEW

Abracadabra Money OFT.

A security review of OFT.

CLIENT	PROJECT	AUDITORS	AUDIT WINDOW	FINAL REPORT
Abracadabra Money	OFT	Guardian	November 12 – December 14, 2024	December 14, 2024

§ 00

Contents.

59 pages: overview, confidence ranking, scope, 48 findings, disclaimer, and about Guardian.

§ 01	Overview	P. 03
§ 02	Confidence Ranking	P. 04
§ 03	Scope & Classifications	P. 05
	Files in Scope	P. 06
§ 04	Findings	P. 07
	Main Review · 39 findings	P. 10
	Remediation Review · 9 findings	P. 49
§ 05	Disclaimer	P. 58
§ 06	About Guardian	P. 59

§ 01

Overview.

Guardian reviewed OFT across 2 review rounds. The final report includes 4 High, 5 Medium, 39 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	MAIN REVIEW COMMIT
OFT	1e8b997f6187
PREPARED BY	
Guardian	
AUDIT WINDOW	
November 12 – December 14, 2024	
FINAL REPORT	
December 14, 2024	

1.1 Confidence Ranking

3 / 5

Moderate Confidence

Guardian assigns a Confidence Ranking of 3 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	4	0	0	0	4
■ Medium	5	0	0	0	5
■ Low	39	0	17	0	22
■ Informational	0	0	0	0	0
All severities	48	0	17	0	31

§ 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	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 ASSIGNED	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 OFT in <https://github.com/Abracadabra-money/abracadabra-money-contracts>. The scope includes 1,360 source lines across 6 files.

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.

Files in Scope.

FILE	NSLOC	TOTAL
MAIN REVIEW		
src/mixins/Create3Factory.sol	15	21
src/periphery/BoundSpellCrosschainActions.sol	198	284
src/periphery/MultiRewardsClaimingHandler.sol	82	116
src/periphery/TokenLocker.sol	231	323
src/staking/MultiRewards.sol	241	333
src/staking/SpellPowerStaking.sol	44	67
REMEDIATION REVIEW		
src/mixins/Create3Factory.sol	15	21
src/periphery/TokenLocker.sol	233	327
src/staking/MultiRewards.sol	253	347
src/staking/SpellPowerStaking.sol	48	72

§ 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					
H-01	stakeFor Function Is Not Overridden	Logical Error	HIGH	RESOLVED	p. 10
H-02	SpellPowerStaking Does Not Override The MultiRewards.stakeFor Function	Validation	HIGH	RESOLVED	p. 11
H-03	Lockup period can be avoided	Protocol	HIGH	RESOLVED	p. 12
M-01	Potential Revert On Zero Transfer	ERC20	MEDIUM	RESOLVED	p. 13
M-02	getRewardsFor receiver is hardcoded	Crosschain	MEDIUM	RESOLVED	p. 14
M-03	Paused staking is possible	Pause	MEDIUM	RESOLVED	p. 15
L-01	Funds Held Hostage With Lockup Period	Configuration	LOW	ACKNOWLEDGED	p. 16
L-02	Potential Revert On Zero Transfer	Configuration	LOW	RESOLVED	p. 17
L-03	Centralization Risks	Centralization	LOW	ACKNOWLEDGED	p. 18
L-04	MultiRewards Contract Only Supports 18 Decimal Tokens	Logical Error	LOW	RESOLVED	p. 19
L-05	Incompatibility With Tokens That Revert On Zero Value Transfers	Logical Error	LOW	RESOLVED	p. 20
L-06	BoundSpellActionSender.send Allows Sending a Zero Amount	Code Best Practices	LOW	RESOLVED	p. 21
L-07	lockingDeadline Parameter Can Be Removed	Code Best Practices	LOW	ACKNOWLEDGED	p. 22
L-08	Missing Require Check in _updateInstantRedeemParams Function	Configuration	LOW	RESOLVED	p. 23
L-09	TokenLocker._redeemFor Function Claims to the to Address Before Redeeming	Validation	LOW	ACKNOWLEDGED	p. 24
L-10	Use of an Outdated Version Of solady's safeTransferLib	Code Best Practices	LOW	RESOLVED	p. 25
L-11	Floating Pragma	Code Best Practices	LOW	ACKNOWLEDGED	p. 26
L-12	Unused Imports	Code Best Practices	LOW	RESOLVED	p. 27
L-13	Lack of a Double Step TransferOwnership Pattern	Code Best Practices	LOW	ACKNOWLEDGED	p. 28

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
L - 14	Incompatibility With Fee-On-Transfer Tokens	DoS	LOW	ACKNOWLEDGED	p. 29
L - 15	Unclaimed Dust in MultiRewards Contract	Code Best Practices	LOW	ACKNOWLEDGED	p. 30
L - 16	Unused Error	Code Best Practices	LOW	RESOLVED	p. 31
L - 17	Hardcoded LZ Params	Configuration	LOW	ACKNOWLEDGED	p. 32
L - 18	Late Redeem Is Incentivized	Configuration	LOW	ACKNOWLEDGED	p. 33
L - 19	Unnecessary Parameter Data	Code Best Practices	LOW	RESOLVED	p. 34
L - 20	Lacking Constant Usage	Code Best Practices	LOW	RESOLVED	p. 35
L - 21	DOS if custom params are disabled	DoS	LOW	ACKNOWLEDGED	p. 36
L - 22	TokenLocker._redeemFor Function Claims to the to Address Before Redeeming	Configuration	LOW	ACKNOWLEDGED	p. 37
L - 23	Uncleared lastLockIndex	Logical Error	LOW	RESOLVED	p. 38
L - 24	Layerzero dust issues	Configuration	LOW	ACKNOWLEDGED	p. 39
L - 25	Ether May Be Stuck In Contract	Logical Error	LOW	RESOLVED	p. 40
L - 26	Can't send native token with CREATE3	Configuration	LOW	RESOLVED	p. 41
L - 27	Unclear Process for Handling Failed Messages on Destination Chain	Configuration	LOW	ACKNOWLEDGED	p. 42
L - 28	Refund Address Should Be Configurable	Logical Error	LOW	RESOLVED	p. 43
L - 29	Reward tokens are not removable	Configuration	LOW	ACKNOWLEDGED	p. 44
L - 30	getRewardsFor receiver is hardcoded	Configuration	LOW	RESOLVED	p. 45
L - 31	MultiRewards Contract Only Supports 18 Decimal Tokens	Logical Error	LOW	RESOLVED	p. 46
L - 32	Warning About minDstGasLookup	Code Best Practices	LOW	ACKNOWLEDGED	p. 47
L - 33	Hardcoded LOCAL_CHAIN_ID	Configuration	LOW	ACKNOWLEDGED	p. 48

REMEDIATION REVIEW

H-01	_getRewardsFor DOS	DOS	HIGH	RESOLVED	p. 49
M-01	Get Rewards DoS	DoS	MEDIUM	RESOLVED	p. 50
M-02	DoS Invalid Address(0) Reward Entry in _getRewardsFor	DoS	MEDIUM	RESOLVED	p. 51
L - 01	Callback Token Reentrancy	Reentrancy	LOW	RESOLVED	p. 52
L - 02	Outdated Comment	Documentation	LOW	RESOLVED	p. 53

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
L-03	Unnecessary Import	Optimization	LOW	RESOLVED	p. 54
L-04	exit Function Only Transfers Rewards To to	Warning	LOW	RESOLVED	p. 55
L-05	Unused Error	Informational	LOW	RESOLVED	p. 56
L-06	Unused Import And Comment	Informational	LOW	RESOLVED	p. 57

HIGH **H-01**

RESOLVED

stakeFor Function Is Not Overridden

CATEGORY	Logical Error
LOCATION	SpellPowerStaking.sol
REVIEW	Main Review

— 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

Status: Resolved.

<https://github.com/Abracadabra-money/abracadabra-money-contracts/commit/0a684a740fc5e592f3a943ce13a40fc311b815a1>

HIGH **H-02**

RESOLVED

SpellPowerStaking Does Not Override The MultiRewards.stakeFor Function

CATEGORY	Validation
LOCATION	SpellPowerStaking.sol
REVIEW	Main Review

— DESCRIPTION

The `BoundSpellActionReceiver` contract interacts with the `SpellPowerStaking` contract via its `stakeFor` function. Specifically, when cross-chain actions are triggered through LayerZero, the `_handleMintBoundSpellAndStake` and `_handleStakeBoundSpell` functions call `spellPowerStaking.stakeFor`.

However, the `SpellPowerStaking` contract enforces a lockup period for withdrawals via the `lastAdded` mapping, which tracks the last timestamp at which a user staked tokens. This lockup mechanism is implemented in the stake function of `SpellPowerStaking`, however, the `stakeFor` function in the `MultiRewards` contract (from which `SpellPowerStaking` inherits) is not overridden in `SpellPowerStaking`.

As a result, when staking occurs through the `BoundSpellActionReceiver` using `stakeFor`, the `lastAdded` mapping is not updated. This allows users to bypass the lockup mechanism as they can stake without the `lastAdded` mapping being updated.

— RECOMMENDATION

Override the `stakeFor` function in the `SpellPowerStaking` contract as shown below:

```
``solidity
function stakeFor(address user, uint256 amount) public virtual override {
    super.stakeFor(user, amount);
    lastAdded[user] = block.timestamp;
}
``
```

— RESOLUTION

Status: Resolved.

HIGH **H-03**

RESOLVED

Lockup period can be avoided

CATEGORY	Protocol
LOCATION	SpellPowerStaking
REVIEW	Main Review

— DESCRIPTION

There is a `lockupPeriod` that must pass after a stake before the user is able to withdraw in the `SpellPowerStaking` contract. The `lastAdded` mapping is updated on each stake, allowing the code to check whether the lockup period has passed on a withdrawal trial. However, `stakeFor` doesn't update that mapping. Any user can use the `BoundSpellActionSender` contract to send a stake message and make the `BoundSpellActionReceiver` call the `stakeFor` function. By doing so, they will successfully stake and avoid the lockup period.

— RECOMMENDATION

Update the lockup period on `stakeFor()` as well.

— RESOLUTION

Status: Resolved.

MEDIUM **M-01**

RESOLVED

Potential Revert On Zero Transfer

CATEGORY ERC20
LOCATION MultiRewards.sol
REVIEW Main Review

— 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

Status: Resolved.

MEDIUM **M-02**

RESOLVED

getRewardsFor receiver is hardcoded

CATEGORY	Crosschain
LOCATION	MultiRewardsClaimingHandler
REVIEW	Main Review

— 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

Status: Resolved.

MEDIUM **M-03**

RESOLVED

Paused staking is possible

CATEGORY Pause
LOCATION MultiRewards
REVIEW Main Review

DESCRIPTION

The `MultiRewards` contract can be paused by its owner. When it's paused, no stakes are allowed. However, users may bypass that restriction by initiating a cross chain stake. The `BoundSpellCrosschainAction` will call the `stakeFor` function which doesn't check if the contract is paused or not. This will allow users to stake even in a paused state and potentially benefit from the current situation.

RECOMMENDATION

Consider disallowing `stakeFor` when the contract is paused.

RESOLUTION

Status: Resolved.

LOW **L-01**

ACKNOWLEDGED

Funds Held Hostage With Lockup Period

CATEGORY Configuration
LOCATION SpellPowerStaking.sol: 49
REVIEW Main Review

— 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

Status: Acknowledged.

LOW **L-02**

RESOLVED

Potential Revert On Zero Transfer

CATEGORY Configuration
LOCATION MultiRewards.sol
REVIEW Main Review

— 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

Status: Resolved.

<https://github.com/Abracadabra-money/abracadabra-money-contracts/commit/7563dfcb95f226809d7c2154c3ce837c5a508df7>

LOW **L-03**

ACKNOWLEDGED

Centralization Risks

CATEGORY	Centralization
LOCATION	MultiRewards.sol, SpellPowerStaking.sol, TokenLocker.sol
REVIEW	Main Review

— 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

Status: Acknowledged.

LOW **L-04**

RESOLVED

MultiRewards Contract Only Supports 18 Decimal Tokens

CATEGORY	Logical Error
LOCATION	MultiRewards.sol: 114, MultiRewards.sol: 122
REVIEW	Main Review

DESCRIPTION

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

```
```solidity
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 `rewardToken` added has 18 decimals.

RESOLUTION

Status: Resolved.

LOW L-05

RESOLVED

Incompatibility With Tokens That Revert On Zero Value Transfers

| | |
|----------|--|
| CATEGORY | Logical Error |
| LOCATION | MultiRewardsClaimingHandler.sol 78, MultiRewards.sol 271, TokenLocker.sol 237, TokenLocker.sol 238 |
| REVIEW | Main Review |

DESCRIPTION

Several functions in the contracts do not check for zero value transfers before attempting a token transfer. This can lead to failed transactions if a token reverts on zero value transfers which is a behavior exhibited by some custom ERC20 tokens. The functions affected are:

- `MultiRewardsClaimingHandler.notifyRewards()`
- `MultiRewards._getRewardsFor(address user, RewardHandlerParams memory params)`
- `TokenLocker._instantRedeemFor()`

Moreover, in the `MultiRewardsClaimingHandler.notifyRewards` function, when there are no pending rewards for a specific `rewardToken`, the LayerZero's `sendFrom` method is still invoked to create a cross-chain message. This should not be done when the reward amount is zero.

RECOMMENDATION

Before any token transfer, add a check to ensure that the amount transferred is greater than zero. On the other hand, ensure that `sendFrom` is only called when the rewards `amount` is higher than zero.

RESOLUTION

Status: Resolved.

LOW **L-06**

RESOLVED

BoundSpellActionSender.send Allows Sending a Zero Amount

CATEGORY Code Best Practices
LOCATION BoundSpellCrosschainActions.sol 105
REVIEW Main Review

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

Status: Resolved.

<https://github.com/Abracadabra-money/abracadabra-money-contracts/commit/4f505890a0c7b757a4ca2705174d87585189c2b9>

LOW **L-07**

ACKNOWLEDGED

lockingDeadline Parameter Can Be Removed

CATEGORY Code Best Practices

LOCATION TokenLocker.sol

REVIEW Main Review

— 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

Status: Acknowledged.

LOW **L-08**

RESOLVED

Missing Require Check in _updateInstantRedeemParams Function

CATEGORY Configuration
LOCATION TokenLocker.sol 309
REVIEW Main Review

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

Status: Resolved.

<https://github.com/Abracadabra-money/abracadabra-money-contracts/commit/05c994fccc2134a6f4510cc0225aa7069d9a3ac0>

LOW **L-09**

ACKNOWLEDGED

TokenLocker._redeemFor Function Claims to the to Address Before Redeeming

CATEGORY Validation
LOCATION TokenLocker.sol 208
REVIEW Main Review

— 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

Status: Acknowledged.

LOW **L-10**

RESOLVED

Use of an Outdated Version Of solady's safeTransferLib

| | |
|----------|---------------------|
| CATEGORY | Code Best Practices |
| LOCATION | Global |
| REVIEW | Main Review |

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:

```
``solidity
[[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] (<https://github.com/Vectorized/solady/pull/1128/files>), 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](<https://github.com/Vectorized/solady/releases/tag/v0.0.267>).

RESOLUTION

Status: Resolved.

<https://github.com/Abracadabra-money/abracadabra-money-contracts/commit/2a7cde101616e03f760c478d0bff3831b284b422>

LOW **L-11**

ACKNOWLEDGED

Floating Pragma

CATEGORY Code Best Practices

LOCATION Global

REVIEW Main Review

— 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

Status: Acknowledged.

LOW **L-12**

RESOLVED

Unused Imports

| | |
|----------|---------------------|
| CATEGORY | Code Best Practices |
| LOCATION | TokenLocker.sol 11 |
| REVIEW | Main Review |

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

Status: Resolved.

<https://github.com/Abracadabra-money/abracadabra-money-contracts/commit/2d0e09efabccff710d65c5de92986422d27d6ef1d>

LOW **L-13**

ACKNOWLEDGED

Lack of a Double Step TransferOwnership Pattern

CATEGORY Code Best Practices

LOCATION MultiRewardsClaimingHandler.sol 15, TokenLocker.sol 16, BoundSpellCrosschainActions.sol 36, BoundSpellCrosschainActions.sol 186

REVIEW Main Review

DESCRIPTION

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

```
``solidity
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] (<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol>) contract.

RESOLUTION

Status: Acknowledged.

LOW **L-14**

ACKNOWLEDGED

Incompatibility With Fee-On-Transfer Tokens

| | |
|----------|-------------|
| CATEGORY | DoS |
| LOCATION | Global |
| REVIEW | Main Review |

— 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

Status: Acknowledged.

LOW **L-15**

ACKNOWLEDGED

Unclaimed Dust in MultiRewards Contract

CATEGORY Code Best Practices

LOCATION MultiRewards.sol

REVIEW Main Review

— 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

Status: Acknowledged.

LOW **L-16**

RESOLVED

Unused Error

CATEGORY Code Best Practices
LOCATION MultiRewards.sol: 42
REVIEW Main Review

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

Status: Resolved.

<https://github.com/Abracadabra-money/abracadabra-money-contracts/commit/ceab273e2d08e2b1a72c813f0836430bb61322b7>

LOW **L-17**

ACKNOWLEDGED

Hardcoded LZ Params

CATEGORY Configuration

LOCATION MultiRewardsClaimingHandler.sol: 88, BoundSpellCrosschainActions.sol: 96

REVIEW Main Review

— 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

Status: Acknowledged.

LOW **L-18**

ACKNOWLEDGED

Late Redeem Is Incentivized

CATEGORY Configuration
LOCATION TokenLocker.sol
REVIEW Main Review

— 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

Status: Acknowledged.

LOW **L-19**

RESOLVED

Unnecessary Parameter Data

| | |
|----------|---------------------------------|
| CATEGORY | Code Best Practices |
| LOCATION | BoundSpellCrossChainActions.sol |
| REVIEW | Main Review |

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

Status: Resolved.

<https://github.com/Abracadabra-money/abracadabra-money-contracts/commit/6a9e62ecf96677657d87291588a9730277809a48>

LOW **L-20**

RESOLVED

Lacking Constant Usage

CATEGORY Code Best Practices
LOCATION BoundSpellCrosschainActions.sol: 88
REVIEW Main Review

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

Status: Resolved.

<https://github.com/Abracadabra-money/abracadabra-money-contracts/commit/29f7b1054a3d549b36ac15adf14df0330f81b446>

LOW L-21

ACKNOWLEDGED

DOS if custom params are disabled

| | |
|----------|-----------------------------|
| CATEGORY | DoS |
| LOCATION | MultiRewardsClaimingHandler |
| REVIEW | Main Review |

— 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

Status: Acknowledged.

LOW L-22

ACKNOWLEDGED

TokenLocker._redeemFor Function Claims to the to Address Before Redeeming

CATEGORY Configuration
LOCATION TokenLocker.sol 208
REVIEW Main Review

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

Status: Acknowledged.

LOW L-23

RESOLVED

Uncleared lastLockIndex

CATEGORY Logical Error
LOCATION TokenLocker.sol:252-271
REVIEW Main Review

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](https://github.com/GuardianAudits/abra-of-1/blob/1e8b997f6187d87e104a33a4f161fd7f6120d385/src/periphery/TokenLocker.sol#L266C1-L268C22) 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

Status: Resolved.

<https://github.com/Abracadabra-money/abracadabra-money-contracts/commit/74f1b9d0c9e3200dddcff552425a4c826a624623>

LOW **L-24**

ACKNOWLEDGED

Layerzero dust issues

| | |
|----------|-----------------------------|
| CATEGORY | Configuration |
| LOCATION | MultiRewardsClaimingHandler |
| REVIEW | Main Review |

— 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

Status: Acknowledged.

LOW **L-25**

RESOLVED

Ether May Be Stuck In Contract

CATEGORY Logical Error
LOCATION MultiRewards.sol: 280
REVIEW Main Review

— 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`:

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

— RESOLUTION

Status: Resolved.

<https://github.com/Abracadabra-money/abracadabra-money-contracts/commit/4be7a94f4e299ae2b4ba7f9d34a9b138d2711989>

LOW L-26

RESOLVED

Can't send native token with CREATE3

CATEGORY Configuration
LOCATION Create3Factory
REVIEW Main Review

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

Status: Resolved.

<https://github.com/Abracadabra-money/abracadabra-money-contracts/commit/9ff6ca1f1c3c934ed632a011c754b667b65bee4f>

LOW **L-27**

ACKNOWLEDGED

Unclear Process for Handling Failed Messages on Destination Chain

CATEGORY Configuration

LOCATION MultiRewardsClaimingHandler.sol, BoundSpellCrosschainActions.sol

REVIEW Main Review

— 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

Status: Acknowledged.

LOW **L-28**

RESOLVED

Refund Address Should Be Configurable

CATEGORY Logical Error
LOCATION MultiRewardsClaimingHandler.sol: 87
REVIEW Main Review

— 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

Status: Resolved.

<https://github.com/Abacadabra-money/abracadabra-money-contracts/commit/147db99b584c45bb7e80fad3e785f4faec7d8171>

LOW **L-29**

ACKNOWLEDGED

Reward tokens are not removable

CATEGORY Configuration

LOCATION MultiRewards

REVIEW Main Review

— 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

Status: Acknowledged.

LOW **L-30**

RESOLVED

getRewardsFor receiver is hardcoded

| | |
|----------|-----------------------------|
| CATEGORY | Configuration |
| LOCATION | MultiRewardsClaimingHandler |
| REVIEW | Main Review |

— 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

Status: Resolved.

<https://github.com/Abracadabra-money/abracadabra-money-contracts/commit/4bcb5dfbcc45459c8a882f60b203286a0e752693>

LOW **L-31**

RESOLVED

MultiRewards Contract Only Supports 18 Decimal Tokens

| | |
|----------|--|
| CATEGORY | Logical Error |
| LOCATION | MultiRewards.sol: 114, MultiRewards.sol: 122 |
| REVIEW | Main Review |

DESCRIPTION

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

```
```solidity
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

Status: Resolved.

<https://github.com/Abracadabra-money/abracadabra-money-contracts/commit/e4194075c92564791a37c2cc9605a2ff3d34d18c>

LOW **L-32**

ACKNOWLEDGED

Warning About minDstGasLookup

CATEGORY Code Best Practices

LOCATION Global

REVIEW Main Review

— 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](https://github.com/Abracadabra-money/abracadabra-oftv2-contracts/blob/4b56ba05cf081bfb1796d1a68fcfe6cf5997551f/src/LzApp.sol#L40C1-L48C11)). 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

Status: Acknowledged.

LOW **L-33**

ACKNOWLEDGED

Hardcoded LOCAL_CHAIN_ID

| | |
|----------|-----------------------------|
| CATEGORY | Configuration |
| LOCATION | MultiRewardsClaimingHandler |
| REVIEW | Main Review |

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

Status: Acknowledged.

HIGH **H-01**

RESOLVED

_getRewardsFor DOS

| | |
|----------|-----------------------------|
| CATEGORY | DOS |
| LOCATION | MultiRewardsClaimingHandler |
| REVIEW | Remediation Review |

— DESCRIPTION

[L-26](https://www.notion.so/guardianaudits/L-26-1438bda5828c80b6915cc243e4b0f87a?pvs=4) shows how in some cases a transfer of 0 amount can happen. It recommends to skip the current loop iteration if the current reward amount is 0 in both `*MultiRewards._getRewardsFor()` and `*MultiRewardsClaimingHandler.notifyRewards()`. The current implemented fix skips the iteration only in `MultiRewards._getRewardsFor()`.

This fix introduces a more serious problem. In `MultiRewards` if the current reward is 0, the entry in the `_rewards` array won't be populated, i.e its `token` and `amount` will both be 0. The whole array will be passed to `rewardHandler.notifyRewards()`. Inside that function, there are two possible scenarios:

A) transfer to the same chain

B) transfer to a different chain

Both of these cases will lead to revert if the current token is `address(0)`. For A, `safeTransfer()` will be called on `address(0)`. And for B, the oft will be retrieved as `tokenOfs[address(0)]` which will result in `address(0)` as well and will revert when `sendFrom()` is called.

In result, core contract functionalities will be blocked for the given user until they accumulate rewards in each token listed.

— RECOMMENDATION

Skip the iteration in `MultiRewardsClaimingHandler.notifyRewards()` if the current amount is 0.

— RESOLUTION

Status: Resolved.

MEDIUM **M-01**

RESOLVED

Get Rewards DoS

| | |
|----------|--------------------|
| CATEGORY | DoS |
| LOCATION | MultiRewards.sol |
| REVIEW | Remediation Review |

DESCRIPTION

In the `_getRewardsFor` function the `_rewards` array is now unpopulated when the reward amount is zero. As a result the `rewardToken` stored in the rewards entry will be `address(0)`. This will lead to a DoS in the `notifyRewards` function in all cases.

Firstly in the `param.dstChainId == LOCAL_CHAIN_ID` case will attempt to call the transfer function on the zero address which will revert.

And in the `param.dstChainId != LOCAL_CHAIN_ID` case the `transferFrom` function is attempted to be called on the `oft` that corresponds to the `token`. The corresponding `oft` will also be the zero address and thus this call will subsequently revert.

RECOMMENDATION

Add early continue cases for both the `token == address(0)` and `amount == 0` for robust compatibility with all operators and tokens which might revert on zero transfer.

RESOLUTION

Status: Resolved.

MEDIUM **M-02**

RESOLVED

DoS Invalid Address(0) Reward Entry in _getRewardsFor

| | |
|----------|--------------------|
| CATEGORY | DoS |
| LOCATION | MultiRewards.sol |
| REVIEW | Remediation Review |

— DESCRIPTION

In the `_getRewardsFor` function, if a particular reward amount is zero, the corresponding `_rewards[i]` entry remains unchanged and effectively defaults to `address(0)` and `amount = 0`. When the `notifyRewards` function subsequently processes the returned `_rewards` array, it attempts to perform a token transfer for that entry, causing the call to revert due to the invalid token address. This will disrupt normal reward distribution and prevent users from successfully receiving other valid rewards in the same batch.

— RECOMMENDATION

Update the `notifyRewards` function to check for `address(0)` before attempting any token transfers. If the token address is `address(0)`, simply continue to the next iteration.

— RESOLUTION

Status: Resolved.

LOW **L-01**

RESOLVED

Callback Token Reentrancy

CATEGORY Reentrancy
LOCATION SpellPowerStaking.sol: 27, 32
REVIEW Remediation Review

DESCRIPTION

In the `stake` and `stakeFor` functions the `lastAdded` mapping is updated after the `super.stake` and `super.stakeFor` calls.

As a result, if the staking token allows any callback hooks upon `transferFrom`, then it may be possible for the user to re-enter into the `SpellPowerStaking` system and withdraw their stake immediately without waiting the lockup period.

There is no identified incentive to do this, and the planned staking tokens do not have callbacks, however as a best practice the `lastAdded` should be updated before the call to `super.stake` and `super.stakeFor`.

RECOMMENDATION

Consider updating the `lastAdded` mapping before the calls to `super.stake` and `super.stakeFor`.

RESOLUTION

Status: Resolved.

LOW **L-02**

RESOLVED

Outdated Comment

CATEGORY Documentation
LOCATION BoundSpellCrossChainActions.sol: 236
REVIEW Remediation Review

— DESCRIPTION

In the documentation for the `BoundSpellActionReceiver` `onOFTReceived` function the comment for the `_payload` parameter still references the `rewardHandlerParams`, however this value has been removed from the `MintBoundSpellAndStakeParams` struct.

— RECOMMENDATION

Consider removing the comment that references the `rewardHandlerParams`.

— RESOLUTION

Status: Resolved.

LOW **L-03**

RESOLVED

Unnecessary Import

| | |
|----------|------------------------------------|
| CATEGORY | Optimization |
| LOCATION | BoundSpellCrossChainActions.sol: 8 |
| REVIEW | Remediation Review |

DESCRIPTION

The `BoundSpellCrossChainActions` file imports the `RewardHandlerParams` however these are not used in either contract in this file.

RECOMMENDATION

Remove the `RewardHandlerParams` import.

RESOLUTION

Status: Resolved.

LOW **L-04**

RESOLVED

exit Function Only Transfers Rewards To to

| | |
|----------|--------------------|
| CATEGORY | Warning |
| LOCATION | MultiRewards.sol |
| REVIEW | Remediation Review |

DESCRIPTION

For the fix of L-25, the `_getRewardsFor` function now accepts a `to` address, allowing the reward recipient on the destination chain to be different from the user address on the source chain. The `_exitFor` and `exit` functions also accept a `to` address as a parameter to achieve this functionality.

``plain text

```
function _exitFor(address user, address to, RewardHandlerParams memory params) internal {
    _withdrawFor(user, balanceOf[user]);
    _getRewardsFor(user, to, params);
}
```

During the call, the `stakingToken` is withdrawn to the `user` address, and rewards are transferred to `to` address. This behavior is intended as part of the fix for the previous L-25 issue.

However, the same behavior is observed when the user calls the regular `exit(address to, RewardHandlerParams memory params)` function. This `exit` function gives the impression that all balances, including the `stakingToken` balance, will be transferred to the `to` address. However, only the rewards are transferred to the `to` address, while the `stakingToken` is still transferred to the `user`` address.

RECOMMENDATION

If this behavior is intended, consider documenting it to avoid misunderstandings. Otherwise, transfer the `stakingToken` to the `to` address.

RESOLUTION

Status: Resolved.

LOW **L-05**

RESOLVED

Unused Error

| | |
|----------|-------------------------------------|
| CATEGORY | Informational |
| LOCATION | MultiRewardsClaimingHandler.sol: 23 |
| REVIEW | Remediation Review |

DESCRIPTION

`ErrZeroAddress` error in `MultiRewardsClaimingHandler` contract is defined but never used.

RECOMMENDATION

Consider removing unused error.

RESOLUTION

Status: Resolved.

LOW **L-06**

RESOLVED

Unused Import And Comment

| | |
|----------|---------------------------------|
| CATEGORY | Informational |
| LOCATION | BoundSpellCrossChainActions.sol |
| REVIEW | Remediation Review |

DESCRIPTION

`rewardHandlerParams` was removed from the `MintBoundSpellAndStakeParams` and is no longer used anywhere in the contract.

However, the `rewardHandlerParams` import statement was not removed, and comments in line 237 still reference it.

RECOMMENDATION

Remove the `rewardHandlerParams` import statement and delete the comment that mentions `rewardHandlerParams`.

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.