GUARDIAN AUDITS

SMART CONTRACT SECURITY AUDIT OF

Abracadabra Money

Summary

Audit Firm Guardian

Prepared By Daniel Gelfand, Owen Thurm, Waffle, Priyam, blnk, 10ambear

Client Firm Abracadabra Money

Final Report Date February 6, 2024

Audit Summary

Abracadabra engaged Guardian to review the security of its staking rewards contract. From the 29th of January to the 1st of February, a team of 6 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Notice that the examined smart contracts are not resistant to internal exploit. For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **Arbitrum**

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

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/AbraPoCs

Table of Contents

Project Information

Smart Contract Risk Assessment

Addendum

Project Overview

Project Summary

Project Name	Abracadabra Money
Language	Solidity
Codebase	https://github.com/Abracadabra-money/abracadabra-money-contracts
Commit(s)	Initial Commit: 20e6b724c4d561d84375060d46042ba4a1d28525 Final Commit: ff019c0232447a8c05a22c0d0b17f14057a0cd0c

Audit Summary

Delivery Date	February 6, 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	3	0	0	0	0	3
● Low	11	0	0	4	0	7

Audit Scope & Methodology

<u>Vulnerability Classifications</u>

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	● Critical	● High	● Medium
Likelihood: *Medium*	● High	● Medium	● Low
Likelihood: *Low*	● Medium	● Low	● Low

<u>Impact</u>

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.

<u>Likelihood</u>

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.

Invariants Assessed

During Guardian's review of Abracadabra's staking contract, fuzz-testing with [Foundry](#) was performed on the protocol's main functions. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 50,000+ runs up to a depth of 250 with a prepared Foundry fuzzing suite.

ID	Description	Initial	Remediated	Run Count
LMR-01	User Locks Do Not Exceed Max Locks	✅	✅	50,000+
LMR-02	Earned Rewards Do Not Exceed Reward Token Balance	✅	✅	50,000+
LMR-03	Sum of Users Locked Balance = Total Locked	✅	✅	50,000+
LMR-04	Sum of Users Unlocked Balance = Total Unlocked	✅	✅	50,000+
LMR-05	Sum of Users Balance = Total Supply	✅	✅	50,000+
LMR-06	Last Reward Time Applicable Is Not Less Than The Reward Period's Last Update Time	✅	✅	50,000+
LMR-07	Latest Lock Has The Latest Unlock Time	❌	✅	50,000+
LMR-08	Total Supply Accurately Increased After Staking	✅	✅	50,000+
LMR-09	User's Staking Contract Balance Accurately Increased After Staking	✅	✅	50,000+
LMR-10	User's Token Balance Decremented By Staked Amount	✅	✅	50,000+
LMR-11	Locking Accurately Increases Total Supply	✅	✅	50,000+

Invariants Assessed

ID	Description	Tested	Passed	Run Count
LMR-12	User's Staking Contract Balance Accurately Increased After Locking	✅	✅	50,000+
LMR-13	User's Token Balance Unchanged After Locking	✅	✅	50,000+
LMR-14	Withdraw Decreases Unlocked Balance By Withdrawn Amount	✅	✅	50,000+
LMR-15	User's Staking Contract Balance Decreased By Withdrawn Amount	✅	✅	50,000+
LMR-16	User's Token Balance Increased By Withdrawn Amount	✅	✅	50,000+
LMR-17	User's Token Balance Increased By Reward Amount	✅	✅	50,000+
LMR-18	User Rewards After Getting Rewards Is 0	✅	✅	50,000+
LMR-19	Total Supply Unchanged After Getting Rewards	✅	✅	50,000+

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	Compounding Rewards Dilutes Rewards For Others	Gaming	● High	Resolved
M-01	Potential System DoS	DoS	● Medium	Resolved
M-02	Trapped MIM Rewards	Trapped Funds	● Medium	Resolved
M-03	_createLocks DoS	DoS	● Medium	Resolved
L-01	LogLockIndexChanged Invalid Index Update	Events	● Low	Resolved
L-02	Lacking rewardToken Validation	Validation	● Low	Resolved
L-03	notifyRewardAmount Does Not Validate rewardToken	Validation	● Low	Resolved
L-04	Potentially Invalid maxLocks Value	Validation	● Low	Resolved
L-05	Invalid Withdraw Function Documentation	Documentation	● Low	Resolved
L-06	Malicious Operator May Reduce Rewards	Centralization	● Low	Resolved
L-07	Rewards Accrue For Expired Locks	Unexpected Behavior	● Low	Acknowledged
L-08	Lacking Constructor Validation	Validation	● Low	Resolved
L-09	Rewards Are Granted Immediately After Locking	Unexpected Behavior	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-10	Invalid Lock Indexes May Cause Out-of-Bounds	Unexpected Behavior	● Low	Acknowledged
L-11	Sequencer Outage Prevents Important Actions	Warning	● Low	Acknowledged

H-01 | Compounding Rewards Dilutes Rewards For Others

Category	Severity	Location	Status
Gaming	● High	LockingMultiReward.sol	Resolved

Description [PoC](PoC)

In the rewards system, rewards are not locked and can be claimed at any point in time, even if the user's staking tokens are locked for the 13 week period.

A user with a large portion of the totalSupply may continuously claim their rewards and lock those tokens to gain an even greater portion, whether it be directly staking if the reward token matches the staking token, or with a swap to the staking token from the reward token.

This will ultimately dilute the rewards for other users in the reward period while massively swaying the rewards towards themselves.

The change in distribution is at the expense of other staked users, as there is a set amount to be distributed and the final rewards are dramatically skewed towards the user due to their continuous compounding while other staked users have their rewards fractionalized.

Recommendation

Consider if this is the expected behavior. If this is expected then clearly document it for users so they have a chance to collect an appropriate amount of rewards. If this is not expected, consider requiring that rewards can only be claimed at the end of the lock period.

Resolution

Abracadabra Team: The issue was resolved in commit [3081d38](3081d38).

Guardian Team: Because reward periods do not necessarily align with epochs, rewards may still be compounded within a reward period. Consider forcing reward notifications to adhere exactly to epoch time windows, otherwise be aware of this behavior and clearly document it for users.

Abracadabra Team: Reward periods are now tied directly to epochs in commit [78b7cfa](78b7cfa).

M-01 | Potential System DoS

Category	Severity	Location	Status
DoS	● Medium	LockingMultiRewards.sol: 433, 482	Resolved

Description

In the _updateRewards function the rewardTokens are iterated over to _updateRewardsGlobal for each token. However there is no explicit bound on the length of the rewardTokens array.

As a result it is possible for the rewardTokens array to become so long that updating the rewards for each token requires more gas than the block gas limit allows.

In such a scenario the _updateRewards function and all functions relying on it would be DoS'd. Similarly in the _getRewards function all rewardTokens are iterated over to pay the user's rewards.

Therefore the owner may DoS the claiming of rewards by adding many rewardTokens with the addReward function.

Recommendation

Consider adding a limit to the amount of rewardTokens that may be supported in the addReward function.

Resolution

Abracadabra Team: The issue was resolved in commit 0bdf6d5.

M-02 | Trapped MIM Rewards

Category	Severity	Location	Status
Trapped Funds	● Medium	LockingMultiReward.sol	Resolved

Description

In the `_rewardPerToken` function if the totalSupply is 0 the `rewardPerTokenStored` is not advanced, meaning that rewards are not accrued until a user stakes.

This behavior is fine when a user does stake at some point during the reward period, however in the case that no users stake for an entire reward period, the rewards for that period will not be distributed and will have to be claimed using the recover function.

In the case where MIM is used as a rewardToken, the undistributed rewards will not be recoverable as the stakingToken cannot be recovered. Therefore any MIM rewards that go undistributed will be locked.

Recommendation

If the `totalSupply` is 0 before the end of a reward period be sure to stake a trivial amount in order to capture any undistributed MIM rewards before calling the `notifyRewardAmount` function.

Resolution

Abracadabra Team: The issue was resolved in commit [710f511](710f511).

Guardian Team: Upon recovery the reward rate is not updated to reflect the new reward token balance in the staking contract. Consider adjusting the reward rate, otherwise be careful to not remove tokens that are necessary for user rewards.

M-03 | _createLocks DoS

Category	Severity	Location	Status
DoS	● Medium	LockingMultiReward.sol: 312	Resolved

Description

In the processExpiredLocks function there is no requirement that the operator must process the oldest expired lock first. Therefore if an operator neglects to process expired locks for a user directly when they unlock it becomes possible for an operator to errantly or maliciously process the lock at the lastLockIndex before processing the other locks.

Depending on where the lastLockIndex is relative to the user's locks array this can yield different outcomes:

In the following example lock A expired before lock B and lock B expired before lock C, lock C is the true last lock:

• lastLockIndex is 2
• The user's locks array is [A, B, C]
• All locks are expired, the operator processes lock C first
• The user's locks array is now [A, B], the lastLockIndex is still 2

However, 2 is an invalid index for the array and now the user cannot create a new lock as the _createLock function will revert with an array index out of bounds error.

Recommendation

Consider requiring that the lock at the lastLockIndex may only be processed when there is only 1 item left in the locks array. Additionally, be sure to carefully process the correct locks on time.

Resolution

Abracadabra Team: The issue was resolved in commit a312ff3.

L-01 | LogLockIndexChanged Invalid Index Update

Category	Severity	Location	Status
Events	● Low	LockingMultiRewards.sol: 354	Resolved

Description

In the processExpiredLocks function when the lock being processed is the final lock in the array, the LogLockIndexChanged event will still emit with a fromIndex of the last index and a toIndex of the last index.

In this case no lock index changed, therefore the LogLockIndexChanged event may be misleading.

Recommendation

Consider only emitting the LogLockIndexChanged event when the index != lastIndex.

Resolution

Abracadabra Team: The issue was resolved in commit 00cc23a.

L-02 | Lacking rewardToken Validation

Category	Severity	Location	Status
Validation	● Low	LockingMultiReward.sol: 256	Resolved

Description

In the addReward function there is no validation that the rewardToken is not already present in the rewardTokens list. There is no significant issue with a duplicate rewardToken, however it would cause unnecessary gas expenditure upon updating and claiming rewards.

Recommendation

Consider using an EnumerableSet for the rewardTokens and validating that the rewardToken is not already present in the set before adding it in the addReward function.

Resolution

Abracadabra Team: The issue was resolved in commit bcd8d9a.

L-03 | notifyRewardAmount Does Not Validate rewardToken

Category	Severity	Location	Status
Validation	● Low	LockingMultiReward.sol: 256, 292	Resolved

Description

The function notifyRewardAmount is used by operators to update the distribution rate and period of a reward token.

The issue is that there is another function addReward which actually adds the reward token to the rewardTokens array state variable.

If the operator uses notifyRewardAmount to distribute a reward token that does not exist in the array, these rewards will not be accounted for as most reward-related functions will loop through the reward token arrays.

Recommendation

Consider adding a check to notifyRewardAmount to verify that the reward token was already added by the owner.

Resolution

Abracadabra Team: The issue was resolved in commit 1c20437.

L-04 | Potentially Invalid maxLocks Value

Category	Severity	Location	Status
Validation	● Low	LockingMultiReward.sol: 100	Resolved

Description

In the constructor the maxLocks is assigned to the result of _lockDuration / _rewardsDuration, this result is valid when the _lockDuration is an exact multiple of the _rewardsDuration, but in any case where the _lockDuration is not an exact multiple of the _rewardsDuration the maxLocks is 1 less than it should be due to truncation.

For example:
• _lockDuration = 5 weeks
• _rewardsDuration = 2 weeks
• maxLocks = 5 / 2 = 2
• However 3 distinct reward periods will intersect the 5 week lock period, therefore a user may have a maximum of 3 locks.

It is unlikely that the _lockDuration would not be a perfect multiple of the _rewardsDuration, however this edge case ought to be either handled or validated against in the constructor.

Recommendation

Either validate that _lockDuration % _rewardsDuration == 0 or assign maxLocks to (_lockDuration + _rewardsDuration • 1) / _rewardsDuration.

Resolution

Abracadabra Team: The issue was resolved in commit b84ba2f.

L-05 | Invalid Withdraw Function Documentation

Category	Severity	Location	Status
Documentation	● Low	LockingMultiReward.sol: 141	Resolved

Description

The documentation for the withdraw function suggests that the function will "iterate through the locks to find expired locks, prunning them and cumulate the amounts to withdraw", however the withdraw function does not implement this behavior.

Recommendation

Consider implementing this behavior for the withdraw function or remove the comment on this behavior.

Resolution

Abracadabra Team: The issue was resolved in commit 1d99a3d.

L-06 | Malicious Operator May Reduce Rewards

Category	Severity	Location	Status
Centralization	● Low	LockingMultiReward.sol: 292	Resolved

Description

In the notifyRewardAmount function the rewardRate is assigned to the result of amount / rewardsDuration, which will include precision loss up to 604800 when the rewardsDuration is 1 week long.

This amount of precision loss will be negligible for tokens with 18 decimals, however if a token such as USDC, with 6 decimals, is used as the rewardToken then this amount of precision loss may compound to be nontrivial over time.

Additionally, a malicious operator could leverage this precision loss to cause significant loss of assets for users (if a 6 decimal token is used). The operator could donate a 0 amount to the notifyRewardAmount function and cause reward periods to end when they have only just begun in order to update the rewardRate and cause precision loss on the total amount to be distributed.

The malicious operator could donate a 0 amount in a loop this way to ultimately cause a significant reduction in the rewardRate due to precision loss.

Recommendation

There is already a recover function to rescue any funds lost due to precision. To address the potential gaming by a malicious operator, consider requiring that a non-trivial amount of the token is donated or that the previous period is fully finished before starting a new one.

Resolution

Abracadabra Team: The issue was resolved in commit bcd8d9a.

L-07 | Rewards Accrue For Expired Locks

Category	Severity	Location	Status
Unexpected Behavior	● Low	LockingMultiReward.sol	Acknowledged

Description

The purpose of the processExpiredLocks function is to transfer the locked Balance to unlocked Balance.

The rewards accrued by the user is calculated based on the balanceOf(user), which is calculated in the following way: bal.unlocked + ((bal.locked * lockingBoostMultiplerInBips) / BIPS).

If there is a delay in calling processExpiredLocks then the users will still accrue the rewards for their expired locks.

Recommendation

Consider if this is the expected behavior, otherwise be sure to process expired locks on time.

Resolution

Abracadabra Team: Acknowledged.

L-08 | Lacking Constructor Validation

Category	Severity	Location	Status
Validation	● Low	LockingMultiReward.sol	Resolved

Description

In the constructor there is no validation that the _lockDuration and _rewardsDuration are not too short nor too long.

Additionally there is no validation that the lockingBoostMultiplier is greater than the minimum basis points divisor, otherwise users would be punished for locking rather than being rewarded.

Recommendation

Consider adding validation in the constructor for the _lockDuration, _rewardsDuration, and lockingBoostMultiplier

Resolution

Abracadabra Team: The issue was resolved in commit a8f386a.

Guardian Team: The MIN_BOOST_MULTIPLIER validation does not ensure locking provides a boost. Consider reverting if _lockingBoostMultiplierInBips <= BIPS instead to ensure there is a boost for locking.

Abracadabra Team: The validation was updated in commit c34decd.

L-09 | Rewards Are Granted Immediately After Locking

Category	Severity	Location	Status
Unexpected Behavior	● Low	LockingMultiReward.sol	Acknowledged

Description

User's locked balances are updated immediately upon locking, even though technically their locking period does not begin until the end of the week they locked within.

This means that user's will begin receiving the locking rewards outside of the 13 week locking period, which can reduce the rewards earned by current lockers and may be unexpected by the protocol.

Recommendation

Consider if this is expected behavior, if it is not then only allow users to accrue locking rewards once their 13 week lock period has begun.

Resolution

Abracadabra Team: Acknowledged.

L-10 | Invalid Lock Indexes May Cause Out-of-Bounds

Category	Severity	Location	Status
Unexpected Behavior	● Low	LockingMultiReward.sol	Acknowledged

Description

The operator must pass the lockIndexes in consideration of an element's placement after the length decrements from the prior indexes.

Consider the following scenario:

• Bob has 3 expired locks [0, 1, 2]

• Operator passes lockIndexes = [0, 1, 2]

• After processing indexes 0 and 1, the lock array's length will only be 1. However, the current index to process is 2.

• Consequently, processExpiredLocks reverts if (locks[index].unlockTime > block.timestamp) due to out-of-bounds.

Recommendation

Consider documenting this scenario and ensure operators are able to track the to-be indexes if needed.

Resolution

Abracadabra Team: We will ensure our gelato task processing locks does this correctly.

L-11 | Sequencer Outage Prevents Important Actions

Category	Severity	Location	Status
Warning	● Low	LockingMultiRewards.sol	Acknowledged

Description

When the Arbitrum sequencer is down, transactions to the LockingMultiRewards contract can still be recorded using the DelayedInbox.

This allows users to access the contract but will not allow operators and admin to call the restricted functions through DelayedInbox due to the way address aliasing works.

From the Arbitrum Docs, "when these messages are executed on L2, the sender's address —i.e., that which is returned by msg.sender — will not simply be the L1 address that sent the message; rather it will be the address's "L2 Alias." An address's L2 alias is its value increased by the hex value 0x1111000000000000000000000000000000001111"

L2_Alias = L1_Contract_Address + 0x1111000000000000000000000000000000001111

As a result the owner will not be able to pause or unpause the contract when the sequencer is down, and operators will not be able to process locks nor notify rewards.

Recommendation

Be aware of this scenario and have a contingency plan in the event that the sequencer is down for an extended period.

Resolution

Abracadabra Team: Acknowledged.

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