GA GUARDIAN

Citrea

Token Review

Security Assessment

March 10th, 2026

Summary

Audit Firm Guardian

Prepared By Felipe Gomez, Roberto Reigada

Client Firm Chainway Labs

Final Report Date March 10, 2026

Audit Summary

Chainway Labs engaged Guardian to review the security of their Citrea Token. From the 27th of February to the 2nd of March, a team of 2 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the lack of High and Critical issues detected during the main review, Guardian assigns a Confidence Ranking of 5 to the protocol. Guardian advises the protocol to consider periodic review with future changes. For detailed understanding of the Guardian Confidence Ranking, please see the rubric on the following page.

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

Guardian Confidence Ranking

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. Low risk of latent critical issues.	0 High/Critical findings and few Low/Medium severity findings.
4: High Confidence	Code is clean, well-structured, and adheres to best practices. Only 1 Significant issue was uncovered per week. 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 issues. Varied Low severity findings.
3: Moderate Confidence	Medium-severity and occasional High-severity issues found. Code is functional, but there are concerning areas (e.g., weak modularity, risky patterns). No critical design flaws, though some patterns could lead to issues in edge cases. **Recommendation:** Address issues thoroughly and consider a targeted follow-up audit depending on code changes.	1-2 High/Critical findings per engagement week.
2: Low Confidence	Code shows frequent emergence of Critical/High vulnerabilities. Audit revealed recurring anti-patterns, weak test coverage, or unclear logic. These characteristics suggest a high likelihood of latent issues. **Recommendation:** Post-audit development and a second audit cycle are strongly advised.	2-4 High/Critical findings per engagement week. Or additional High/Critical findings uncovered in remediation review which have not been resolved and confirmed by Guardian.
1: Very Low Confidence	Code has systemic issues. Multiple High/Critical findings (≥5/week), poor security posture, and design flaws that introduce compounding risks. Safety cannot be assured. **Recommendation:** Halt deployment and seek a comprehensive re-audit after substantial refactoring.	≥5 High/Critical findings and overall systemic flaws.

Table of Contents

Project Overview

Project Summary

Project Name	Citrea Token
Language	Solidity
Codebase	https://github.com/chainwayxyz/citrea-token/
Commit(s)	Main Review commit: 0230630de9d4460937735cadd3f6cd1876534dc4 Remediation Review commit: 00060a0b9b4f48418dbfb40997bbcc3ebae4c2a3

Audit Summary

Delivery Date	March 10, 2026
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	0	0	0	0	0	0
Medium	0	0	0	0	0	0
Low	2	0	0	0	0	2
Info	7	0	0	2	0	5

Audit Scope & Methodology

```
contract,source,total,comment
citrea-token-okko-xctr-admin-params/src/CitreaToken.sol,21,26,1
citrea-token-okko-xctr-admin-params/src/GaugeVotes.sol,35,55,11
citrea-token-okko-xctr-admin-params/src/xCitreaToken.sol,260,446,105
citrea-token-okko-xctr-admin-params/script/01_DeployCTR.s.sol,25,37,4
citrea-token-okko-xctr-admin-params/script/02_DeployXCTR.s.sol,48,63,1
citrea-token-okko-xctr-admin-params/config/xCTRConfig.sol,7,10,5
source count: {
  total: 637,
  source: 396,
  comment: 127,
  single: 97,
  block: 30,
  mixed: 3,
  empty: 117,
  todo: 0,
  blockEmpty: 0,
  commentToSourceRatio: 0.3207070707070707
}
```

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
L-01	Orphaned Rewards Harm First Depositor	Rewards	Low	Resolved
L-02	maxWithdraw/maxRedeem Returns Non-Zero Values	Unexpected Behavior	Low	Resolved
I-01	Missing View Preview For Custom Withdrawals	Best Practices	Info	Resolved
I-02	RewardsAdded Emits Exiter As Sender	Events	Info	Resolved
I-03	Deposit(0) Succeeds Silently	Validation	Info	Acknowledged
I-04	Exit Penalty Rounds Down Favoring Exiter	Rounding	Info	Resolved
I-05	cancelExit Mints Zero Shares On Inflation	Validation	Info	Resolved
I-06	ERC4626 First Depositor Inflation Attack	Math	Info	Acknowledged
I-07	No Pause Mechanism For Emergency Response	Best Practices	Info	Resolved

L-01 | Orphaned Rewards Harm First Depositor

Category	Severity	Location	Status
Rewards	● Low	src/xCitreaToken.sol - addRewards() and totalAssets()	Resolved

Description

The `addRewards()` function does not check whether `totalSupply() > 0`. If rewards are added when there are no stakers, the rewards inflate `totalAssets()` without any shares to distribute them to.

When the reward epoch elapses and a user deposits:
1. `totalAssets()` = `orphanedRewards` (large value from streamed rewards)
2. `totalSupply()` = 0 (no stakers)
3. User's shares = deposit * (0 + 1) / (`orphanedRewards` + 1) → rounds to 0
4. User's CTR is transferred in but they receive 0 shares — permanent loss

No attacker required. Realistic scenarios:
• Governance adds rewards before the first staker joins
• All stakers exit while an automated reward distribution contract continues
• A delayed reward transaction lands after the last staker exits

PoC: Add 100 CTR rewards to empty vault, wait 14 days for full streaming, victim deposits 100 CTR → receives 0 shares, loses everything.

Recommendation

Add a supply check to addRewards:

```
function addRewards(uint256 amount) external {
require(totalSupply() > 0, 'xCTR: No stakers');
_addRewards(amount);
SafeERC20.safeTransferFrom(IERC20(asset()), msg.sender, address(this), amount);
}
```

Alternatively, mint dead shares in `initialize()` to prevent the zero-supply state from ever occurring.

Resolution

Chainway Labs Team: Pending.

L-02 | maxWithdraw/maxRedeem Returns Non-Zero Values

Category	Severity	Location	Status
Unexpected Behavior	Low	src/xCitreaToken.sol: 316-317 - maxWithdraw(), maxRedeem()	Resolved

Description

The withdraw() and redeem() functions are overridden to always revert with 'xCTR: Use startExit'. However, maxWithdraw() returns convertToAssets(maxRedeem(owner)) and maxRedeem() returns balanceOf(owner) — both non-zero for users with xCTR shares.

Per ERC4626 specification:

• maxWithdraw MUST return 0 if withdraw would revert for the given owner

• maxRedeem MUST return 0 if redeem would revert for the given owner

This breaks ERC4626 composability. Integrators (aggregators, routers, yield optimizers) querying these functions would believe withdrawal is possible, construct transactions accordingly, and have them revert.

Recommendation

Override both to return 0:

```
function maxWithdraw(address) public pure override returns (uint256) {
return 0;
}
function maxRedeem(address) public pure override returns (uint256) {
return 0;
}
```

Resolution

Chainway Labs Team: Pending.

I-01 | Missing View Preview For Custom Withdrawals

Category	Severity	Location	Status
Best Practices	Info	src/xCitreaToken.sol	Resolved

Description

The vault replaces standard ERC4626 withdrawal mechanics with a custom delayed-exit lifecycle through startExit, completeExit and cancelExit, but it does not provide a dedicated view function that returns expected net assets for a delayed withdrawal path at a chosen completion time.

Integrators can estimate the instant path using convertToAssets and getInstantExitPenalty, but the delayed path requires either executing a state-changing startExit first or re-implementing internal penalty math and per-exit snapshot logic off-chain.

This creates integration friction and increases the chance of quote mismatch between user interfaces, bots and on-chain behavior. Because delayed exits depend on snapshotted per-exit parameters and elapsed time, the absence of a canonical view method means every integrator must duplicate protocol logic to forecast outcomes.

In practice, duplicated logic tends to drift over time, especially after governance parameter changes or future upgrades, producing inaccurate UX and failed expectations.

Recommendation

Add explicit read-only quote functions for the custom exit model. A practical approach is to expose one view that previews delayed exit payout for a hypothetical new exit and one view that previews payout for an existing pending exit at an arbitrary timestamp.

These helpers should use the same internal math and snapshotted fields as execution paths so integrators can rely on canonical on-chain previews without state mutation.

Resolution

Chainway Labs Team: Pending.

I-02 | RewardsAdded Emits Exiter As Sender

Category	Severity	Location	Status
Events	Info	src/xCitreaToken.sol: 341 - _addRewards()	Resolved

Description

The _addRewards() function emits RewardsAdded(msg.sender, amount) regardless of call context. When called from addRewards() (L137), msg.sender is the actual external reward contributor who transfers tokens in — correct.

However, _addRewards() is also called from completeExit() (L175) and instantExit() (L211) for penalty redistribution. In these exit paths, msg.sender is the exiting user, not a reward contributor. No new tokens enter the contract — the penalty is simply reallocated from the exiting user's locked assets to the reward pool. Yet the event is identical to a genuine external reward addition.

Off-chain indexers, dashboards, and analytics tools that track RewardsAdded events will incorrectly attribute penalty reallocations as external reward contributions from exiting users. This leads to inflated reward contribution metrics and incorrect reward source accounting (internal penalty vs external deposit). No funds at risk — only emitted event metadata is misleading.

Recommendation

Emit a distinct event for penalty-sourced rewards:

```
event PenaltyRedistributed(address indexed exiter, uint256 amount);
function _addRewards(uint256 amount) internal {
_addRewardsInternal(amount);
emit RewardsAdded(msg.sender, amount);
}
function _addPenaltyRewards(uint256 amount) internal {
_addRewardsInternal(amount);
emit PenaltyRedistributed(msg.sender, amount);
}
```

Resolution

Chainway Labs Team: Pending.

I-03 | Deposit(0) Succeeds Silently

Category	Severity	Location	Status
Validation	Info	src/xCitreaToken.sol - inherited deposit()	Acknowledged

Description

Calling `deposit(0, receiver)` succeeds without reverting. It mints 0 shares and emits a misleading `Deposit(sender, receiver, 0, 0)` event. The inherited OZ `ERC4626` implementation does not guard against zero-amount deposits.

Recommendation

Override deposit to reject zero amounts:

```
function deposit(uint256 assets, address receiver) public override returns (uint256) {
require(assets > 0, 'xCTR: Zero deposit');
return super.deposit(assets, receiver);
}
```

Resolution

Chainway Labs Team: Pending.

I-04 | Exit Penalty Rounds Down Favoring Exiter

Category	Severity	Location	Status
Rounding	Info	src/xCitreaToken.sol: 266 - getExitPenalty()	Resolved

Description

The exit penalty calculation in `getExitPenalty()` uses `Math.mulDiv` with default floor rounding:

```
return Math.mulDiv(assets, scaledPenalty, BPS * durationRange);
```

This rounds the penalty DOWN, meaning the exiting user pays slightly less than they should. In DeFi protocols, fees and penalties should round UP (in favor of the protocol/remaining stakers) to prevent value leakage.

Impact is 1 wei per exit. Cumulative impact is negligible even with millions of exits.

Recommendation

Use ceiling rounding for the penalty calculation:

```
return Math.mulDiv(assets, scaledPenalty, BPS * durationRange, Math.Rounding.Ceil);
```

Resolution

Chainway Labs Team: Pending.

I-05 | cancelExit Mints Zero Shares On Inflation

Category	Severity	Location	Status
Validation	Info	src/xCitreaToken.sol: 199 - cancelExit()	Resolved

Description

When a user cancels a pending exit via `cancelExit()`, the function converts locked assets back to shares at the current exchange rate without checking that the resulting shares are non-zero:

```
uint256 shares = convertToShares(pendingExit.assets);
// No require(shares > 0) check
_mint(msg.sender, shares);
```

If the share price has been inflated (via direct CTR donation or reward injection to empty vault), `convertToShares` rounds down to 0. The user's pending exit assets are returned to the pool (increasing `totalAssets` for other stakers) but the user receives 0 shares back — effectively losing their entire position.

This is a secondary impact of the first-depositor inflation vector. It requires the share price to already be massively inflated, which itself requires an economically irrational griefing attack.

Recommendation

Add a minimum shares check in `cancelExit`:

```
uint256 shares = convertToShares(pendingExit.assets);
require(shares > 0, 'xCTR: Zero shares');
_mint(msg.sender, shares);
```

Alternatively, store the original shares burned in `startExit` within the `PendingExit` struct and restore that exact amount on cancel, avoiding exchange rate dependency entirely.

Resolution

Chainway Labs Team: Pending.

I-06 | ERC4626 First Depositor Inflation Attack

Category	Severity	Location	Status
Math	Info	src/xCitreaToken.sol - deposit() (inherited from ERC4626Upgradeable), totalAssets()	Acknowledged

Description

The `xCitreaToken` contract does not override `_decimalsOffset()`, which defaults to 0 in `OpenZeppelin`'s `ERC4626` implementation. This means the contract relies solely on the +1 virtual share/asset protection, which while making the attack non-profitable for the attacker (OZ's documented mitigation), still allows griefing of early depositors.

An attacker can: (1) Deposit 1 wei CTR to get 1 share, (2) Transfer large CTR directly to xCTR to inflate `totalAssets`, (3) Subsequent depositors receive 0 shares due to inflated exchange rate.

PoC shows: attacker deposits 1 wei + donates 100 CTR. Victim deposits 50 CTR and gets 0 shares (100% loss). However, attacker spends 100 CTR and can only redeem ~75 CTR, losing ~25 CTR net. With the 50% instant exit penalty, the attacker's total loss is ~62.5 CTR to grief a 50 CTR deposit.

Additionally, `xCTR` shares are non-transferable, preventing the attacker from selling inflated shares. OZ documents this as an accepted trade-off with offset=0: 'the default offset (0) makes it non-profitable even if an attacker is able to capture value from multiple user deposits.'

Recommendation

1. Override `_decimalsOffset()` to return 3-6, making inflation attacks orders of magnitude more expensive than profitable.
2. As defense-in-depth, add `require(shares > 0)` in deposit to revert instead of silently minting 0 shares (note: this converts the attack from silent loss to DoS on small deposits, so it's secondary to the offset fix).
3. Alternatively, mint dead shares in `initialize()` to establish a minimum share price floor.

Resolution

Chainway Labs Team: Pending.

I-08 | No Pause Mechanism For Emergency Response

Category	Severity	Location	Status
Best Practices	Info	src/xCitreaToken.sol - contract-wide	Resolved

Description

The `xCitreaToken` vault does not implement any pause functionality (e.g., `OpenZeppelin PausableUpgradeable`). If a critical vulnerability is discovered post-deployment, there is no way to temporarily halt deposits, exits, or reward additions while a fix is prepared and deployed via the upgrade mechanism.

The only emergency response available is upgrading the proxy implementation, which requires deploying new code, going through the proxy admin flow, and potentially waiting for any timelock delays. During this window, the vulnerable functions remain fully operational.

Given that the contract is upgradeable (`TransparentUpgradeableProxy`), adding pausability would complement the upgrade pattern by providing an immediate circuit breaker while a proper fix is developed and audited.

Recommendation

Inherit `PausableUpgradeable` and add `whenNotPaused` modifiers to key state-changing functions:
• deposit/mint
• `addRewards`
• `startExit`
• `instantExit`

Ensure `completeExit` and `cancelExit` are NOT paused — users should always be able to withdraw assets already in the exit pipeline. This provides an instant circuit breaker that buys time for a proper upgrade fix.

Resolution

Chainway Labs Team: Pending.

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
I-01	Zero-Supply Breaks Cancels And Deposits	Unexpected Behavior	Info	Acknowledged
I-02	Misleading ERC4626 Deposit And Mint Limits	Unexpected Behavior	Info	Acknowledged
I-03	Pausing Can Lock Users Mid-Exit	Unexpected Behavior	Info	Acknowledged

I-01 | Zero-Supply Breaks Cancels And Deposits

Category	Severity	Location	Status
Unexpected Behavior	Info	xCitreaToken.sol	Acknowledged

Description

`xCitreaToken` allows the last staker to burn the final real shares through `startExit()`, while the vault can still retain economic value in several forms. Residual assets can remain because ERC4626 conversions use virtual assets and virtual shares. Unreleased rewards can keep streaming into `totalAssets()` after supply has already reached zero. Direct CTR transfers to the vault also increase its raw backing. Once that happens, the contract enters a broken state where it still has value, but there are no live shares anchoring the exchange rate.

Recommendation

The vault should never be allowed to reach `totalSupply() == 0` while any economically meaningful underlying value remains. A robust mitigation is to mint permanent dead shares during initialization so share supply can never fully disappear. Another acceptable approach is to add explicit zero-supply unwind logic that sweeps or neutralizes residual value when last live shares are burned.

`cancelExit()` should also stop depending on a fresh ERC4626 conversion. The contract already knows how many shares were burned when the exit began, so that quantity should be stored in `PendingExit` and restored directly on cancel:

```
struct PendingExit {
    uint256 startTime;
    uint256 shares;
    uint256 assets;
    // ...
}
```

In addition, deposit() and mint() should revert whenever the previewed share amount is zero. The required invariant is that no user action can leave the vault with zero supply and positive backing while any later user can still interact with ERC4626 entry or cancel paths.

Resolution

Chainway Labs Team: Acknowledged.

I-02 | Misleading ERC4626 Deposit And Mint Limits

Category	Severity	Location	Status
Unexpected Behavior	Info	xCitreaToken.sol	Acknowledged

Description

xCitreaToken overrides maxDeposit() and maxMint() to return the absolute ERC20VotesUpgradeable._maxSupply(). That value is only the global cap used by the votes extension. It is not the amount of additional shares that can still be minted from the current vault state, and it does not reflect whether the vault is paused. As a result, the two ERC4626 limit functions no longer satisfy the normal integrator expectation that they represent a safe upper bound for a successful call.

This mismatch matters because ERC4626 routers and frontends routinely query maxDeposit() or maxMint() before deciding whether to submit a transaction. In this implementation, those functions can return a large nonzero value even when totalSupply() is already near the votes cap or while deposit() and mint() are disabled by whenNotPaused. A caller can therefore observe an apparently valid limit, submit a transaction within that limit, and still revert later when _mint() hits the votes cap or when the pause check rejects the operation.

This issue breaks composability with tooling that relies on ERC4626 limit introspection and can cause unnecessary reverts.

Recommendation

maxMint() should return only the remaining votes headroom rather than the absolute cap, and maxDeposit() should be derived from that remaining headroom under the current conversion rate. Both functions should also return zero while the vault is paused so that off-chain callers do not get an obviously misleading admission signal.

A minimal correction is:

```
function maxMint(address) public view override returns (uint256) {
    if (paused()) return 0;
    return ERC20VotesUpgradeable._maxSupply() - totalSupply();
}
```

maxDeposit() can then convert that remaining share headroom into assets using the live ERC4626 exchange rate. The required invariant is that a call within the reported maxDeposit() or maxMint() bound should not fail for reasons that are already knowable from current contract state.

Resolution

Chainway Labs Team: Acknowledged.

I-03 | Pausing Can Lock Users Mid-Exit

Category	Severity	Location	Status
Unexpected Behavior	Info	src/xCitreaToken.sol:210,240	Acknowledged

Description

The remediation introduces `PausableUpgradeable` and applies `whenNotPaused` broadly. However, `completeExit()` and `cancelExit()` are also guarded by `whenNotPaused` (see src/xCitreaToken.sol:210,240). This means the owner can call `pause()` and block users from finalizing already-started exits or cancelling exits within the intended cancellation window.

Pausability is typically intended to block new inbound actions (`deposit()`/`mint()`/`startExit()`/`addRewards()`) while still allowing users to complete withdrawals/egress. As implemented, pausing can temporarily lock user funds mid-exit and makes users dependent on the owner calling `unpause()` before they can retrieve assets.

Recommendation

Keep pausability, but scope it to block new inbound actions while allowing fund egress.

Suggested approach:

- Keep `whenNotPaused` on: `deposit()`, `mint()`, `addRewards()`, `startExit()` (and optionally `instantExit()`)
- Remove `whenNotPaused` from: `completeExit()` and `cancelExit()` so users can always finalize an exit/cancel even during an incident.

If the protocol explicitly intends to pause exits too, document this governance/ops trust assumption and consider a separate, more strictly governed `withdrawalsPaused` flag (timelock/multisig) rather than reusing `pause()`.

Resolution

Chainway Labs 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

Founded in 2022 by DeFi experts, Guardian is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian 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