

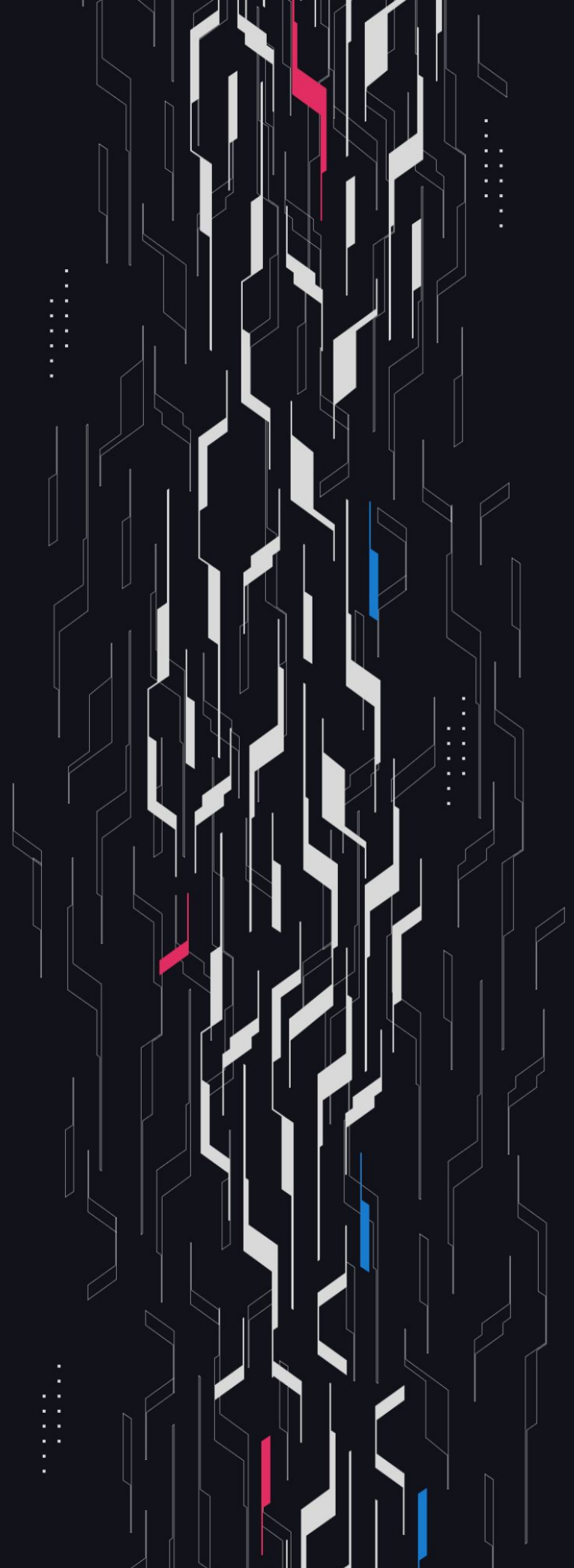
GA GUARDIAN

Yellow

Nitewatch

Security Assessment

March 8th, 2026



Summary

Audit Firm Guardian

Prepared By Osman Ozdemir, Ali Shehab

Client Firm Yellow

Final Report Date 03/08/2026

Audit Summary

Yellow engaged Guardian to review the security of its Nitewatch Contracts. From March 2nd through March 6th, 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 Information

Project Overview 5

Audit Scope & Methodology 6

Smart Contract Risk Assessment

Findings & Resolutions 9

Addendum

Disclaimer 22

About Guardian 23

Project Overview

Project Summary

Project Name	Yellow
Codebase	https://github.com/layer-3/nitewatch
Commit(s)	Main Review Commit: 29c9d97c8951652f9128e5f0032618b874804d4c

Audit Summary

Delivery Date	3/8/2026
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	3	0	0	3	0	0
● Info	9	0	0	9	0	0

Audit Scope & Methodology

```
contract,source,total,comment
nitewatch/contracts/evm/src/ThresholdCustody.sol,159,210,2
source count: {
  total: 210,
  source: 159,
  comment: 2,
  single: 2,
  block: 0,
  mixed: 0,
  empty: 49,
  todo: 0,
  blockEmpty: 0,
  commentToSourceRatio: 0.012578616352201259
}
```

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: <i>High</i>	Impact: <i>Medium</i>	Impact: <i>Low</i>
Likelihood: <i>High</i>	● Critical	● High	● Medium
Likelihood: <i>Medium</i>	● High	● Medium	● Low
Likelihood: <i>Low</i>	● 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	RemoveSingners Can Fail Due To Old Threshold	Logical Error	● Low	Acknowledged
L-02	Signer Front-Runs Removal To Finalize Withdrawal	Frontrunning	● Low	Acknowledged
L-03	Re-added Signer's Stale Approval Is Resurrected	Validation	● Low	Acknowledged
I-01	Phantom Withdrawals Corrupt Rate Limits (OOS)	Logical Error	● Info	Acknowledged
I-02	Unused Error In ThresholdCustody	Informational	● Info	Acknowledged
I-03	Worker Cannot Reject Before Expiry (OOS)	Logical Error	● Info	Acknowledged
I-04	startWithdraw Does Not Auto-approve For Caller	Best Practices	● Info	Acknowledged
I-05	Contract Bypasses Off-Chain Rate Limits (OOS)	Logical Error	● Info	Acknowledged
I-06	Type Inconsistency In ThresholdCustody	Informational	● Info	Acknowledged
I-07	Informational Note For Users	Informational	● Info	Acknowledged
I-08	Misleading README In Nitewatch	Documentation	● Info	Acknowledged
I-09	CEI Pattern Not Followed	Best Practices	● Info	Acknowledged

L-01 | RemoveSingners Can Fail Due To Old Threshold

Category	Severity	Location	Status
Logical Error	● Low	ThresholdCustody.sol: 121-123	Acknowledged.

Description

The removeSigners function removes the signers first and then sets the new threshold.

```
_removeSigners(signersToRemove.toAddressByteArray());  
_setThreshold(newThreshold);
```

However, the internal MultiSignerERC7913._removeSigners function always performs the _validateReachableThreshold check. Since the threshold has not yet been updated at this stage, the validation is executed against the previous threshold value.

As a result, any attempt to remove enough signers that the old threshold becomes unattainable fails even if the caller passes a smaller newThreshold

Recommendation

Set the new threshold first and then remove the signers.

Resolution

Yellow Team: Acknowledged.

L-02 | Signer Front-Runs Removal To Finalize Withdrawal

Category	Severity	Location	Status
Frontrunning	● Low	ThresholdCustody.sol: 103-123	Acknowledged.

Description

A signer about to be removed can observe the pending `removeSigners` transaction in the mempool and front-run it by submitting a `finalizeWithdraw` call with higher gas. Because the removal has not yet been executed, the signer remains in the active set and passes the `onlySigner` modifier.

When a pending withdrawal is one approval short of the threshold, this front-run pushes the approval count over, triggering `_executeWithdrawal` and transferring the funds. The `removeSigners` transaction confirms afterward — the signer is removed, but the funds are already gone.

The remaining signers have no way to prevent this. Even if they no longer want the withdrawal to proceed — whether due to changed circumstances, a revised signer configuration, or simply reconsidering the request — `rejectWithdraw` is only callable after `OPERATION_EXPIRY` (1 hour). There is no on-chain cancellation mechanism during the active window.

Recommendation

Consider acknowledging this behavior so that signers are aware of it.

Resolution

Yellow Team: Acknowledged.

L-03 | Re-added Signer's Stale Approval Is Resurrected

Category	Severity	Location	Status
Validation	● Low	ThresholdCustody.sol: 85	Acknowledged.

Description

In ThresholdCustody.sol, the `_countValidApprovals` function iterates all current signers and checks `withdrawalApprovals[withdrawalId][s]`. When a signer is removed via `removeSigners`, their entry in `withdrawalApprovals` mapping is NOT cleared, it remains `true`.

If that same address is later re-added via `addSigners` within the OPERATION_EXPIRY window of a pending withdrawal, the old approval is counted again without the re-added signer explicitly calling `finalizeWithdraw`.

Recommendation

Consider storing and validating the timestamp when the signer was added and validating it against the request creation date in `_countValidApprovals`, or acknowledging this behavior if it is intended.

Resolution

Yellow Team: Acknowledged.

I-01 | Phantom Withdrawals Corrupt Rate Limits (OOS)

Category	Severity	Location	Status
Informational	● Info	ThresholdCustody.sol: 35	Acknowledged.

Description

InvalidThreshold error is defined in ThresholdCustody contract but never used.

Recommendation

Remove unused error.

Resolution

Yellow Team: Acknowledged.

I-02 | Unused Error In ThresholdCustody

Category	Severity	Location	Status
Logical Error	● Info	service.go: 258-296	Acknowledged.

Description

In service.go, the processWithdrawal function treats a successful finalizeWithdraw transaction as a completed withdrawal. However, in the ThresholdCustody logic, a successful finalizeWithdraw call does not necessarily mean the withdrawal was executed. The function records the signer’s approval, but executes the withdrawal only when validApprovals >= request.requiredThreshold. If the threshold is not yet reached, the transaction succeeds, records an additional approval and the withdrawal remains pending. Despite this, the worker records the withdrawal in the database as approved and finalized.

This creates a mismatch between the off-chain accounting state and the actual contract state. In particular, receipt.Status == 1 only confirms that the transaction did not revert - it does not guarantee that funds were transferred or that the withdrawal was finalized. Because the database is later used to calculate hourly and daily withdrawal totals, pending withdrawals may be counted as completed withdrawals even though no assets have actually left custody. As a result, the rate-limiting mechanism can be manipulated or poisoned, causing legitimate withdrawals to be blocked based on phantom usage.

Recommendation

Update the worker so that it records a withdrawal only after the contract has actually finalized it on-chain. processWithdrawal should stop using receipt.Status == 1 as the completion condition and instead verify that a matching WithdrawFinalized event (with success == true) was emitted. If finalizeWithdraw succeeds but no such event is observed, the request should remain pending and must not be included in the withdrawal totals used for rate limiting.

Resolution

Yellow Team: Acknowledged.

I-03 | Re-added Signer's Stale Approval Is Resurrected

Category	Severity	Location	Status
Best Practices	● Info	ThresholdCustody.sol: 138-162	Acknowledged.

Description

The `startWithdraw` function creates a withdrawal request but does not count the calling signer as an approver. The signer who initiates the withdrawal must call `finalizeWithdraw` separately to register their own approval.

The caller already passed the `onlySigner` check, so the contract knows they are a valid signer, but their approval is not recorded. For a threshold of 1 this means two transactions are needed where one should be enough. For higher thresholds it adds an extra round-trip for the initiator.

Recommendation

Record the caller's approval inside `startWithdraw` so the initiating signer does not need a second transaction.

Resolution

Yellow Team: Acknowledged.

I-04 | startWithdraw Does Not Auto-approve For Caller

Category	Severity	Location	Status
Logical Error	● Info	ThresholdCustody.sol: 196-201	Acknowledged.

Description

In processWithdrawal, the worker attempts to call rejectWithdraw immediately when a withdrawal fails the off-chain policy checks. However, the contract allows rejection only after the request has expired:

```
require(block.timestamp > request.createdAt + OPERATION_EXPIRY, WithdrawalNotExpired());
```

As a result, when the worker receives a fresh WithdrawStarted event and immediately tries to reject it, the call will typically revert because the expiry window has not yet passed. This means the worker cannot actually reject policy-violating withdrawals at the moment they are detected. Once the rejection attempt fails, the service records the event as an error and treats it as processed, which prevents the same request from being retried later when it does become eligible for rejection. Consequently, this leaves policy-violating withdrawals unresolved in a pending state and may allow them to be finalized by other signers before expiry.

Recommendation

If immediate off-chain policy veto is required, add a dedicated on-chain path. Introduce a separate function that is callable only by a dedicated Nitewatch role operated by the off-chain worker, not by arbitrary signers. If ThresholdCustody is intended to remain committee-driven, do not use the current worker with ThresholdCustody.

Resolution

Yellow Team: Acknowledged.

I-05 | Contract Bypasses Off-Chain Rate Limits (OOS)

Category	Severity	Location	Status
Informational	● Info	ThresholdCustody.sol: 14-19	Acknowledged.

Description

newThreshold is defined as uint256 in all hardcoded type hashes (e.g., SET_THRESHOLD_TYPEHASH), while the corresponding function parameters use uint64 newThreshold. Although this does not affect the resulting hash, aligning the types would reduce the risk of confusion or incorrect signing by off-chain tooling.

Recommendation

Ensure the newThreshold type is consistent between the function parameters and the hardcoded type hashes.

Resolution

Yellow Team: Acknowledged.

I-06 | Type Inconsistency In ThresholdCustody

Category	Severity	Location	Status
Logical Error	● Info	global	Acknowledged.

Description

The withdrawal rate-limiting logic exists only in the off-chain worker and is not enforced by ThresholdCustody itself. This is especially problematic when the threshold is set to 1, because startWithdraw immediately executes the withdrawal in that case. As a result, funds can leave custody before the worker has any opportunity to apply the configured policy checks, allowing the off-chain rate limits to be bypassed.

Recommendation

If withdrawal rate limiting is intended security control, it should be enforced on-chain before a withdrawal can be executed. The contract should validate the limits during the withdrawal flow, rather than relying exclusively on the off-chain worker. If it is intended to keep policy evaluation off-chain, the contract should be adjusted so that execution cannot occur without an explicit approval from the off-chain component. As a short-term mitigation, avoid configurations with threshold = 1, since that setup allows withdrawals to be completed before any off-chain checks can run.

Resolution

Yellow Team: Acknowledged.

I-07 | Informational Note For Users

Category	Severity	Location	Status
Informational	● Info	Global/Nitewatch	Acknowledged.

Description

The protocol does not maintain on-chain accounting. All accounting is performed off-chain based on on-chain events. Users should be aware that they cannot track or view their balances directly on-chain and must use the protocol UI.

Recommendation

This is an informational issue intended to highlight the protocol design for readers and users. No changes are required.

Resolution

Yellow Team: Acknowledged.

I-08 | Misleading README In Nitewatch

Category	Severity	Location	Status
Documentation	● Info	ThresholdCustody.sol: 192	Acknowledged.

Description

The README in the Nitewatch repository states: " The **Event Daemon** waits for the outcome (WithdrawFinalized or WithdrawRejected), fires an internal event, and NeoDAX debits the balance upon successful confirmation."

However, the custody contract only emits the WithdrawFinalized event with a bool success parameter. It emits true for successful withdrawals and false for rejected ones, and does not emit a WithdrawRejected event.

Recommendation

Consider updating the README to reflect the actual contract behavior. Also, ensure that off-chain listeners do not rely on a WithdrawRejected event and instead correctly differentiate WithdrawFinalized events based on the boolean success value.

Resolution

Yellow Team: Acknowledged.

I-09 | CEI Pattern Not Followed

Category	Severity	Location	Status
Best Practices	● Info	ThresholdCustody.sol: 213-215	Acknowledged.

Description

The `_executeWithdrawal` function clears storage variables after performing an unsafe external call. While all important functions use the `nonReentrant` modifier and the `request.finalized` flag is set to true before the external call, preventing possible reentrancy, it is recommended to update all state variables before making an external call to an arbitrary address.

Recommendation

Update state variables before making external calls as a best practice.

Resolution

Yellow 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>