Neosoul
Audit Report

Tue Aug 25 2026

contact@bitslab.xyz https://twitter.com/scalebit_

ScaleBit

Neosoul Audit Report

1 Executive Summary

1.1 Project Information

Description	NeoSoul is the largest AI agent infrastructure platform in the BNB Chain and 0G ecosystems, built to accelerate the AI economy. It enables agents to collaborate, compete and create value through harness engineering, which separates AI judgment from capital control
Type	AI, Gaming Data Protocol
Auditors	Alex,Bear Two,Nebu1a
Timeline	Tue Aug 18 2026 - Tue Aug 25 2026
Languages	Solidity
Platform	BSC,0G
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/NeoSoul-AI/evoevo-contracts/
Commits	a1a171486d77115fa04f65958a5b7a25ebd71d77 4b045819409498cac1c93cdaf30cf65409f46171 9308aa329b7fda7667cf2c4f482d61b056611a8e d8c81580cb3fe786248d6b666d712ff81b8f4adf

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
EPR	src/EvoPredictionRegistry.sol	65e1bae49ac3533864a71d9989200496731ab746
EER	src/EvoEvolutionRegistry.sol	5bf64353f3e18e2fe1243a2bcbdbf053facfaabf
EUAR	src/EvoUserActionRouter.sol	8bfbb7c47c10bebcaa027cf2972b50e092068e4f
EBR	src/EvoBindingRegistry.sol	b8ae1bb725cc98d0b4ef263c2716bcd9e9bdce90
ECO	src/EvoCommitteeOracle.sol	21c251817acac7053f80ae2ac7d1d8f035a9259a

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	12	11	1
Critical	0	0	0
Major	4	3	1
Medium	5	5	0
Minor	2	2	0
Informational	1	1	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by NeoSoul to identify any potential issues and vulnerabilities in the source code of the Neosoul smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 12 issues of varying severity, listed below.

ID	Title	Severity	Status
ECO-1	Unbounded Iteration in `_eligibleJurors()` Enables Gas-Exhaustion Denial of Service on Committee Selection	Medium	Fixed
ECO-2	Refund Challenge Bond and Introduce Incentive Reward Upon Successful Challenge	Medium	Fixed
ECO-3	Challenger Funds Permanently Locked Due to Missing ETH Withdrawal Function in Contract	Major	Fixed
ECO-4	The Signature can be Shared between the Two Functions	Minor	Fixed
ECO-5	Proposal Hash Fragmentation Causes Inaccurate Tallying and Potential Stalling	Medium	Fixed
EBR-8	Permanently Disabled Self-Hosted Registration Leaves Binding Entry Points Uncallable	Major	Fixed

ECO-9	Protocol Configuration Can Be Arbitrarily Changed by a Single Governor Role	Medium	Fixed
ECO-10	Missing State Change Check in Juror Activation Causes Redundant Writes and Events	Minor	Fixed
ECO-11	Redundant Juror Activation Entry Points May Cause Inconsistent Behavior	Informational	Fixed
EBR-12	Missing Access Control in `initializeV2` Allows Arbitrary Identity Registry Injection	Medium	Fixed
ECO-13	OutcomeKey Grouping Ignores Proposal Content, Enabling False Consensus	Major	Fixed
ECO-14	Transfer Approval Misused as Voting Authorization	Major	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the Neosoul Smart Contract :

Initializer

- Initializer can initialize contracts once and assign initial roles through the `initialize()` function.
- Initializer can enable dual-registry support once through the `initializeV2()` function.

Admin

- Admin can configure supported identity registries through the `setIdentityRegistrySupported()` function.
- Admin can configure trusted routers through the `setTrustedRouter()` function.
- Admin can configure router targets through the `setBindingRegistry()` , `setEvolutionRegistry()` , and `setOwnerJudgementRegistry()` functions.
- Admin can manage prediction roles through the `setManager()` , `setResultPublisher()` , and `setSnapshotPublisher()` functions.
- Admin can pause or unpause user actions through the `setPaused()` function.
- Admin can force-remove bindings through the `forceUnbind()` or `forceUnbindV2()` function.
- Admin can upgrade implementations through the UUPS upgrade flow.

Manager

- Manager can open predictions through the `openPrediction()` function.
- Manager can link opinions to predictions through the `setPredictionOpinionLink()` or `setPredictionOpinionLinks()` function.

User

- User can bind an existing agent through the `bindExistingAgentV2()` function.
- User can unbind an agent through the `unbindV2()` function.

- User can submit reasoning commitments through the `intakeReasoningV2()` function.
- User can judge predictions through the `judgeV2()` function.
- User can bind finalized prediction settlement data through the `bindPredictionSettlementSummary()` function.

Juror

- Juror can register through the `registerJuror()` function.
- Juror can activate or deactivate with approval through the `activateJurorWithSig()` or `setJurorActiveWithSig()` function.
- Juror can submit committee resolutions through the `submitJurorResolution()` function.

Automation Role

- Automation role can request committee selection through the `requestSelectionForPrediction()` function.
- Automation role can retry or finalize selection through the `retrySelectionForPrediction()` or `finalizeSelectionForPrediction()` function.
- Automation role can finalize pending results through the `finalizePendingResult()` function.
- Automation role can mark stalled predictions through the `markPredictionStalled()` function.

Result Publisher

- Result publisher can publish oracle result summaries through the `publishOracleResultSummary()` function

Snapshot Publisher

- Snapshot publisher can publish judgement snapshots through the `publishJudgementSnapshot()` function.

Challenge Operator

- Challenge operator can challenge pending results through the `challengePendingResult()` function.

Emergency Guardian

- Emergency guardian can resolve blocked predictions through the `emergencyResolvePrediction()` function.

4 Findings

ECO-1 Unbounded Iteration in `_eligibleJurors()` Enables Gas-Exhaustion Denial of Service on Committee Selection

Severity: Medium

Status: Fixed

Code Location:

src/EvoCommitteeOracle.sol#801-815

Descriptions:

In the `_drawJurors()` function, the protocol calls `_eligibleJurors()`, which loops through all registered jurors to obtain the eligible ones.

```
function _eligibleJurors() internal view returns (uint256[] memory out) {
    out = new uint256[](_jurorTokenIds.length);
    uint256 count;
    for (uint256 i = 0; i < _jurorTokenIds.length; i++) {
        uint256 tokenId = _jurorTokenIds[i];
        Juror storage juror = _jurors[tokenId];
        if (juror.registered && juror.active && juror.cooldownUntil <= block.timestamp) {
            out[count] = tokenId;
            count++;
        }
    }
    assembly {
        mstore(out, count)
    }
}
```

However, if a malicious attacker registers a large number of jurors using many addresses, the `_jurorTokenIds` array will become very large. This could cause the loop to run out of gas, preventing `finalizeSelectionForPrediction` from succeeding.

Suggestion:

It is recommended to limit the number of jurors retrieved per call.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ECO-2 Refund Challenge Bond and Introduce Incentive Reward Upon Successful Challenge

Severity: Medium

Status: Fixed

Code Location:

src/EvoCommitteeOracle.sol#735-755

Descriptions:

The purpose of `_finalizePendingResult` is to officially confirm a proposal that has reached quorum as the final result after the challenge window closes, and to complete the settlement process for the prediction.

```
function _finalizePendingResult(uint256 predictionId, address finalizer) internal {
    PredictionAssignment storage assignment = _getAssignmentStorage(predictionId,
PredictionStatus.PendingFinality);
    if (block.timestamp <= assignment.challengeDeadline) {
      revert ChallengeWindowStillOpen(predictionId, block.timestamp,
assignment.challengeDeadline);
    }

    bytes32 proposalHash = assignment.pendingProposalHash;
    ProposalTally storage tally = _proposalTallies[predictionId][proposalHash];
    _storeResult(predictionId, proposalHash, tally, finalizer, false);
    assignment.status = PredictionStatus.Finalized;

    emit PredictionResolvedByCommittee(
      predictionId,
      proposalHash,
      tally.evidenceBundleHash,
      tally.resolutionKind,
      tally.winningOptionIndex,
      block.timestamp,
      finalizer
```

```
        );
    }
```

However, if there is a challenger in between and the challenge succeeds, the protocol does not refund the bond to the challenger.

Suggestion:

It is recommended to refund the challenge bond to the challenger upon a successful challenge and also provide a certain incentive.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ECO-3 Challenger Funds Permanently Locked Due to Missing ETH Withdrawal Function in Contract

Severity: Major

Status: Fixed

Code Location:

src/EvoCommitteeOracle.sol#505-531

Descriptions:

The challenger deposits ETH into the contract when submitting a challenge, but the contract lacks a withdrawal function for ETH, causing the challenger's deposited funds to be stuck in the contract indefinitely with no way to be withdrawn.

```
 function challengePendingResult(uint256 predictionId, bytes32
challengerEvidenceBundleHash, bytes32 reasonCode)
      external
      payable
      onlyRole(CHALLENGE_ROLE)
  {
      PredictionAssignment storage assignment = _getAssignmentStorage(predictionId,
PredictionStatus.PendingFinality);
      if (block.timestamp > assignment.challengeDeadline) {
          revert ChallengeWindowClosed(predictionId, block.timestamp,
assignment.challengeDeadline);
      }
      if (_challenges[predictionId].exists) revert AlreadyChallenged(predictionId);
      if (challengerEvidenceBundleHash == bytes32(0)) revert ZeroEvidenceBundleHash();
      if (msg.value < protocolConfig.challengeBond) {
          revert InsufficientChallengeBond(protocolConfig.challengeBond, msg.value);
      }

      _challenges[predictionId] = Challenge({
          exists: true,
          challenger: msg.sender,
          evidenceBundleHash: challengerEvidenceBundleHash,
```

```
            reasonCode: reasonCode,
            bondAmount: uint96(msg.value),
            openedAt: uint64(block.timestamp)
        });
        assignment.status = PredictionStatus.Challenged;

        emit PredictionChallenged(predictionId, msg.sender,
challengerEvidenceBundleHash, reasonCode, uint96(msg.value));
    }
```

Suggestion:

It is recommended to transfer the ETH to a designated address, or to implement a function for withdrawing ETH.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ECO-4 The Signature can be Shared between the Two Functions

Severity: Minor

Status: Fixed

Code Location:

src/EvoCommitteeOracle.sol#333-343

Descriptions:

Both `activateJurorWithSig()` and `setJurorActiveWithSig()` use signatures to perform business operations.

```
    function setJurorActiveWithSig(
        uint256 memberTokenId,
        bool active,
        uint256 nonce,
        uint256 deadline,
        bytes calldata signature
    ) external {
        address owner = _assertStrictPredictionTokenOwner(msg.sender, memberTokenId);
        _consumeSetJurorActiveApproval(owner, memberTokenId, active, nonce, deadline,
signature);
        _setJurorActive(memberTokenId, active);
    }
```

However, if both functions are only used to set the juror's active status to true, then the signatures for these two functions could be shared. It is worth considering whether they need to be kept separate.

Suggestion:

It is recommended to refactor the function and consolidate the duplicated business logic.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ECO-5 Proposal Hash Fragmentation Causes Inaccurate Tallying and Potential Stalling

Severity: Medium

Status: Fixed

Code Location:

src/EvoCommitteeOracle.sol#441-504

Descriptions:

The `proposalHash` is computed using all input parameters, all of which are provided by the user.

```
bytes32 proposalHash = hashCommitteeResolution(
    predictionId, resolutionKind, winningOptionIndex, optionVotes, evidenceBundleHash
);
```

As a result, there can be cases where two jurors select the same winning option, but if any of the following parameters differ the `optionVotes` array the `evidenceBundleHash` and the `resolutionKind` then their resulting `proposalHash` values will be different, causing their submissions to be tallied into separate `ProposalTally` buckets. This fragmentation leads to inaccurate aggregation of support for the same outcome, and eventually triggers `markPredictionStalled`, putting the prediction into the `Stalled` state, where it can only be resolved through emergency intervention.

Suggestion:

It is recommended to take measures to mitigate this issue.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

EBR-8 Permanently Disabled Self-Hosted Registration Leaves Binding Entry Points Uncallable

Severity: Major

Status: Fixed

Code Location:

src/EvoBindingRegistry.sol#143

Descriptions:

In `EvoBindingRegistry`, `setSelfHostedRegistrationEnabled()` reverts whenever enabled is true and always forces `selfHostedRegistrationEnabled` to `false`. The `initialize()` function also hard-codes the flag to false. Because both `registerAndBind()` and `registerAndBindFor()` require `selfHostedRegistrationEnabled` to be true before proceeding, they always revert with `SelfHostedRegistrationDisabled`. These registration entry points are permanently unreachable dead code. If self-hosted registration is meant to be available in any deployment, the feature cannot be used at all; otherwise the dead code should be removed to avoid misleading interfaces.

Suggestion:

It is recommended to either remove the permanently disabled registration functions, or make the flag configurable through an admin-only setter with a valid enable path, and clearly document which deployment mode each entry point serves.

Resolution:

The team fixed the issue by deleting `registerAndBind / registerAndBindFor / setSelfHostedRegistrationEnabled` and Router side entry, which can be found at commit ce2c29cde49f2bb7dc828bfdc885933edc458127.

ECO-9 Protocol Configuration Can Be Arbitrarily Changed by a Single Governor Role

Severity: Medium

Status: Fixed

Code Location:

src/EvoCommitteeOracle.sol#293

Descriptions:

In `EvoCommitteeOracle`, `setProtocolConfig()` lets any account with `GOVERNOR_ROLE` overwrite the whole committee configuration at any time. The validation in `_setProtocolConfig()` only checks that several values are non-zero and that quorum does not exceed `primaryCount` plus `reserveCount`. It sets no upper bounds for `primaryCount`, `reserveCount`, `challengeBond`, `graceWindow`, or the window durations. A single `GOVERNOR_ROLE` holder can set quorum and primaryCount to 1 so one juror submission finalizes a prediction, or set very large member counts to make `_drawJurors()` too expensive to execute. Already assigned predictions keep the snapshot copied into `PredictionAssignment`, so in-flight voting is mostly protected, but every new selection request and retry immediately uses the changed configuration.

Suggestion:

It is recommended to move configuration changes behind a timelock or multi-signature governance process, enforce strict minimum and maximum bounds for every field, and prevent changes from affecting predictions that have already been opened.

Resolution:

The team partially fixed the issue by adding upper and lower bounds for all fields and primary strict majority validation to the contract, which can be found at commit ce2c29cde49f2bb7dc828bfdc885933edc458127.

However, regarding the issue of "preventing changes from affecting predictions that have already been opened", the project team did not fix it.
Their response was that "`_setProtocolConfig` currently does not check the assignment status of any predictions. Verification revealed a very narrow risk boundary: the `quorum/voteWindow/graceWindow/challengeWindow` of predictions already in `Voting/PendingFinality` are snapshotted into `PredictionAssignment` upon selection completion and are unaffected by subsequent `setProtocolConfig` calls. Only two points are truly affected: `finalizeSelectionForPrediction` reads real-time `primaryCount` / `reserveCount` during the `SelectionPending` queue, and `challengePendingResult` reads real-time `challengeBond` during the `PendingFinality` window. Triggering this issue requires a proactive modification of the `GOVERNOR_ROLE` configuration that happens to coincide with these two narrow time windows. It is not something an external attacker could trigger; the consequence is unexpected process behavior rather than financial loss or result forgery".

ECO-10 Missing State Change Check in Juror Activation Causes Redundant Writes and Events

Severity: Minor

Status: Fixed

Code Location:

src/EvoCommitteeOracle.sol#943

Descriptions:

In `EvoCommitteeOracle`, `_setJurorActive()` sets `juror.active` and emits `JurorActiveUpdated` without checking whether the current value already equals the requested value. The entry points `activateJuror()`, `setJurorActive()`, `activateJurorWithSig()`, and `setJurorActiveWithSig()` can therefore repeat the same no-op update. Each repeated call writes the same storage value again, emits a duplicate event, and the signature-based entry points also consume another jurorActivationNonces nonce. The impact is limited to wasted gas, event noise, and nonce consumption. No funds or access control are affected.

Suggestion:

It is recommended to skip the storage write and the event emission when `juror.active` already equals the requested active value.

Resolution:

The team adopted our advice and fixed the issue by skipping the storage write and the event emission when `juror.active` already equals the requested active value, which can be found at commit ce2c29cde49f2bb7dc828bfdc885933edc458127.

ECO-11 Redundant Juror Activation Entry Points May Cause Inconsistent Behavior

Severity: Informational

Status: Fixed

Code Location:

src/EvoCommitteeOracle.sol#317

Descriptions:

In `EvoCommitteeOracle`, `activateJuror()` and `setJurorActive()` are both callable by `JUROR_MANAGER_ROLE` and both call `_setJurorActive()`. `activateJuror()` is a hard-coded special case that always passes true, so it is equivalent to `setJurorActive(memberTokenId, true)`. Because `_setJurorActive()` does not check whether juror.active already equals the requested value, calling `activateJuror()` on an active juror performs a redundant storage write and emits a duplicate `JurorActiveUpdated` event. The two entry points also create a maintenance risk because a future fix applied to one function may be missed in the other. The impact is limited to redundant writes, event noise, and inconsistent behavior. No funds or access control are affected.

Suggestion:

It is recommended to remove `activateJuror()` or turn it into a documented wrapper of `setJurorActive()`, and add a state-change check inside `_setJurorActive()` so repeated calls are skipped.

Resolution:

The team adopted our advice and fixed the issue by removing `activateJuror()` and `activateJurorWithSig`, which can be found at commit ce2c29cde49f2bb7dc828bfdc885933edc458127.

EBR-12 Missing Access Control in `initializeV2` Allows Arbitrary Identity Registry Injection

Severity: Medium

Status: Fixed

Code Location:

src/EvoBindingRegistry.sol#105

Descriptions:

In `EvoBindingRegistry`, `initializeV2()` is guarded only by `reinitializer(2)`. The `reinitializer()` modifier prevents the function from running more than once, but it does not check who calls it. The fresh deployment script only invokes the V1 `initialize()` function, so version 2 stays unused after deployment and leaves a public window. In that window, any attacker can deploy a registry that has code and call `initializeV2()` with the attacker's registry address. The malicious registry is then permanently added to `supportedIdentityRegistries`, the `publicIdentityRegistry` value is poisoned, and version 2 is consumed so the admin can no longer initialize it correctly. The malicious registry can fake the return values of `ownerOf()`, `getApproved()`, and `isApprovedForAll()`, which lets the attacker forge agent bindings and V2 judgements.

Suggestion:

It is recommended to add an access control modifier such as `onlyRole(ADMIN_ROLE)` to `initializeV2()`, and to complete the V2 initialization atomically inside the deployment transaction instead of relying on a later public transaction.

Resolution:

The team adopted our advice and fixed the issue by adding an access control modifier such as `onlyRole(ADMIN_ROLE)` to `initializeV2()`, which can be found at commit ce2c29cde49f2bb7dc828bfdc885933edc458127.

ECO-13 OutcomeKey Grouping Ignores Proposal Content, Enabling False Consensus

Severity: Major

Status: Fixed

Code Location:

src/EvoCommitteeOracle.sol#543

Descriptions:

The contract uses `outcomeKey` to tally votes (i.e., to determine whether quorum has been reached), but once quorum is met, it takes the `proposalHash` of the first submitter under that grouping as the final result.

```
 bytes32 outcomeKey = _outcomeKey(predictionId, resolutionKind, winningOptionIndex,
optionVotes.length);
        if (_outcomeCanonicalProposal[predictionId][outcomeKey] == bytes32(0)) {
            _outcomeCanonicalProposal[predictionId][outcomeKey] = proposalHash;
        }
        uint16 outcomeCount = _outcomeApprovalCounts[predictionId][outcomeKey] + 1;
        _outcomeApprovalCounts[predictionId][outcomeKey] = outcomeCount;

        emit JurorResolutionSubmitted(
            predictionId, memberTokenId, proposalHash, evidenceBundleHash,
tally.approvalCount, reserveMember
        );

        if (outcomeCount >= assignment.quorum) {
            bytes32 canonicalHash = _outcomeCanonicalProposal[predictionId][outcomeKey];
            ProposalTally storage canonicalTally = _proposalTallies[predictionId]
[canonicalHash];
            assignment.status = PredictionStatus.PendingFinality;
            assignment.pendingProposalHash = canonicalHash;
            assignment.pendingOpenedAt = uint64(block.timestamp);
            assignment.challengeDeadline = uint64(block.timestamp +
assignment.challengeWindow);
```

```
            emit PendingFinalityOpened(
                predictionId, canonicalHash, canonicalTally.evidenceBundleHash,
assignment.challengeDeadline
            );
        }
```

Suppose a prediction requires 3 jurors (quorum = 3) to pass.

- Juror A submits: winningOptionIndex = 1, optionVotes = `[100, 0]`, evidenceBundleHash = `0x123`.
- Juror B submits: winningOptionIndex = 1, optionVotes = `[50, 50]`, evidenceBundleHash = `0x456`.
- Juror C submits: winningOptionIndex = 1, optionVotes = `[0, 100]`, evidenceBundleHash = `0x789`.

All three have different `proposalHash` values (because the array contents and evidence differ). However, their `outcomeKey` values are exactly the same. The contract increments the tally for that `outcomeKey` to 3 (>= quorum), which triggers finality. The contract then retrieves the first proposal stored under that `outcomeKey` (i.e., Juror A's `[100, 0]`) and sets it as the final resolution.

Thus, the final resolution becomes A's `[100, 0]`, even though A received only 1 vote.

Suggestion:

It is recommended to take measures to mitigate this issue.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

ECO-14 Transfer Approval Misused as Voting Authorization

Severity: Major

Status: Acknowledged

Code Location:

src/EvoCommitteeOracle.sol#949-954

Descriptions:

The `_assertPredictionTokenOwner()` check allows not only the NFT owner but also approved addresses to pass.

```
  function _assertPredictionTokenOwner(address operator, uint256 tokenId) internal
view {
      address tokenOwner = nft.ownerOf(tokenId);
      if (operator != tokenOwner && nft.getApproved(tokenId) != operator &&
!nft.isApprovedForAll(tokenOwner, operator)) {
          revert Unauthorized();
      }
  }
```

If the NFT is listed on a secondary marketplace such as OpenSea or an aggregator, and the owner has granted approval to that platform, then a third-party approved operator can call submitJurorResolution() to cast a vote on behalf of the owner. As a result, the operator can submit a resolution that goes against the owner's true intention.

Suggestion:

It is recommended to take measures to mitigate this issue.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't pose any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.

