



GRISOCO AI

Smart-Contract Security Assessment

SAMPLE REPORT - ILLUSTRATIVE FINDINGS ON A TEST CONTRACT

GENERATED	2026-10-10 11:55 UTC
AUDIT ROUND	1
ASSESSMENT MODE	Automated agentic audit (AI review, independent verification, and deterministic tools)

57/100

CRITICAL RISK

3 confirmed findings (1 critical, 1 high, 1 low).

ISSUED

AUD-8FB086F3A1AD014F

2026-10-10 · HMAC-SHA256 · fingerprint 8fb086f3a1ad014f

Attests that this document is an unedited Grisoco AI report. It is not a certificate of the contract's security.

Confidential — prepared for engineering review. Distribute only to parties authorized by the contract owner.

How This Score Was Calculated

The score starts at 100 and subtracts points for each finding in this report's main section. Weights are fixed: critical 25, high 15, medium 7, low 3, informational 1. Repeats of the same detector decay rather than stacking — the second time one detector raises the same title it counts half, the third a third, and so on — because one pattern to fix is one risk, not nine. Deterministic tool findings, dependency-file findings and code-quality notes are excluded entirely, which is why the sections at the end of this report do not move this number. A critical or high finding sets the risk grade on its own, whatever the score.

Severity	Weight	Repeat	Charged	Finding
CRITICAL	25	1	25.00	Session-key spend limit is charged for the first call of a batch only
HIGH	15	1	15.00	A session key can grant itself a new, unlimited key through the unchecked batch tail
LOW	3	1	3.00	A session key registered with validUntil = 0 never expires
			43.00	Total penalty
			57	Score = 100 - 43.00

Executive Summary

Grisoco AI analyzed BatchSessionKeyAccount and recorded 3 finding(s) across 1 critical, 1 high, 0 medium, 1 low, and 0 informational severity.

This report is the output of an automated, multi-model AI audit. No human security engineer reviewed the audited code. It is decision-support evidence for a qualified engineering review — it is not a certification, warranty, or guarantee that the audited contract is free of vulnerabilities, nor proof that every exploitable condition has been identified. Automated and AI-assisted analysis carries an inherent false-negative risk: the absence of a finding is not evidence of the absence of risk, and additional audit rounds lower that risk without removing it. A qualified human reviewer must independently verify exploitability and the adequacy of any remediation — including for code paths and properties outside this report's scope — before the audited contract is relied upon to hold real value.

Our recommendation is not changed by a clean result: commission an independent manual review and a full human audit before the contract is deployed to a network holding real value. A contract that has completed this audit has been examined by several independent models and agent roles rather than by a single pass, and — where the findings recorded here have been remediated and re-checked in a later round — carries fewer known defects than it did on entry. That is a statement about this process, not a measure of the contract's total risk.

Severity	Count
CRITICAL	1
HIGH	1
MEDIUM	0
LOW	1
INFORMATIONAL	0

Audit Scope

Project / company	SAMPLE REPORT - illustrative findings on a test contract
Contracts audited	1
Dependency files compiled	0
EIP-170 limit	24,576 bytes
Delegated spending authority	Detected (ERC-4337 account, session keys)
Delegation review	Included: spend limits across validation and execution, per-key permission storage, delegation narrowing, expiry and revocation, and the reach of each granted scope.

Contracts Audited

The contracts selected for this audit. The findings, the finding total, and the security score in this report cover these files.

File	Contract	License	Source size	Runtime bytecode
BatchSessionKeyAccount.sol	BatchSessionKey Account	MIT	4,127 bytes (4.03 KiB)	Unavailable

Files Analyzed

- BatchSessionKeyAccount.sol — BatchSessionKeyAccount (4,127 bytes (4.03 KiB)) · License: MIT · Runtime bytecode: Unavailable

Privileged Access Map

- OWNER → addSessionKey(address key, address token, uint256 limit, uint48 validUntil), removeSessionKey(address key)

Roles listed above are the ones this scan recognises: an `onlyOwner`` modifier, OpenZeppelin's `onlyRole(...)`` and `hasRole(...)``, and a call to `owner()`. A guard written any other way — a custom modifier, an inline caller check, or an internal guard function — is not shown here, so treat this map as a floor rather than the complete picture.

Emergency Control Review

Pause	Not detected
Unpause	Not detected
Asset recovery	Not detected
Upgrade controls	Not detected
Circuit breaker	Not detected

Methodology and Tool Coverage

This audit was performed by Grisoco AI's audit engine rather than by a single model answering a single prompt. Every candidate finding is independently verified before it is reported, and a confirmed finding can still be demoted. Deterministic tools run alongside the review, and their output is triaged rather than accepted on sight.

No single opinion becomes a finding on its own: independent verification is the quality gate.

Stage	Role in the audit
AI review	Deep review of the source
Verification	Independent verification of every candidate
Final review	A last review of every confirmed finding
Tool triage	Gating of deterministic tool findings
Synthesis	Deduplication, regression comparison, and report drafting

Deterministic tool coverage for this run:

Tool	Status	Detail
static-analysis	Completed	0 finding(s)
pattern-analysis	Completed	0 finding(s)
symbolic-analysis	Completed	0 finding(s)
lint	Completed	0 finding(s)
property-fuzzing	Completed	0 finding(s)

AI-sourced findings below were not accepted on a single opinion. Each candidate was independently verified, and a confirmed finding can still be demoted. See Verification under each such finding.

Findings Index

#	Severity	Title	Location
1	CRITICAL	Session-key spend limit is charged for the first call of a batch only	BatchSessionKeyAccount.sol:73
2	HIGH	A session key can grant itself a new, unlimited key through the unchecked batch tail	BatchSessionKeyAccount.sol:48
3	LOW	A session key registered with validUntil = 0 never expires	BatchSessionKeyAccount.sol:52

Findings, Remedy, and Safe Local PoC Plan

CRITICAL

1. Session-key spend limit is charged for the first call of a batch only

Category: delegation_scope_escape · Location: BatchSessionKeyAccount.sol:73 · Evidence source: AI review

What was observed: validateUserOp decodes the whole batch but inspects and charges only calls[0] against the key's remaining allowance (lines 73-78). executeBatch then performs every call in the batch (line 88). A key limited to 100 tokens submits [transfer 100, transfer 100, ...] and is charged once; calls after the first can also name any target, value and calldata. Whoever holds the session key, an AI agent or an attacker who took its key, can move the account's entire balance.

Recommended remedy: Validate and charge every call in the batch, not the first: require each target to be the session token, each value to be zero, each selector to be transfer, and charge the sum of the amounts against the allowance. Reject batches the loop cannot fully check.

Safe local PoC plan: On a disposable local fork with an EntryPoint deployed, grant a session key or delegation exactly as the finding describes, then sign and submit a user operation whose calldata goes beyond the stated intent. Record whether validation accepts it and what the account's balances look like afterwards. Use only a disposable local fork or test deployment. Start at `BatchSessionKeyAccount.sol:73`, exercise the flagged path, and capture the transaction result, events, state difference, and trace. Never use production keys or broadcast this test.

Verification: confirmed

Fix validation: Add a regression test for the intended invariant, re-run the audit, then replay the relevant flow on a disposable fork.

HIGH

2. A session key can grant itself a new, unlimited key through the unchecked batch tail

Category: delegation_scope_escape · Location: BatchSessionKeyAccount.sol:48 · Evidence source: AI review

What was observed: onlyOwner admits msg.sender == address(this) (line 48), and executeBatch calls each target from the account itself. Because only calls[0] is checked, a later call can target the account and invoke addSessionKey with any limit and expiry, turning a temporary, bounded key into a permanent one the owner never granted.

Recommended remedy: Never let a session-key batch call the account itself: reject any call whose target is address(this), and keep administrative functions reachable by the owner's signature only.

Safe local PoC plan: On a disposable local fork with an EntryPoint deployed, grant a session key or delegation exactly as the finding describes, then sign and submit a user operation whose calldata goes beyond the stated intent. Record whether validation accepts it and what the account's balances look like afterwards. Use only a disposable local fork or test deployment. Start at `BatchSessionKeyAccount.sol:48`, exercise the flagged path, and capture the transaction result, events, state difference, and trace. Never use production keys or broadcast this test.

Verification: confirmed

Fix validation: Add a regression test for the intended invariant, re-run the audit, then replay the relevant flow on a disposable fork.

LOW

3. A session key registered with validUntil = 0 never expires

Category: session_key_lifecycle · Location: BatchSessionKeyAccount.sol:52 · Evidence source: AI review

What was observed: addSessionKey stores validUntil as given, and validation packs it into validationData (line 79). Under ERC-4337 a validUntil of 0 means no expiry, so a key added without one keeps its authority until the owner removes it.

Recommended remedy: Require validUntil to be non-zero and within a maximum lifetime when adding a key.

Safe local PoC plan: On a disposable local fork with an EntryPoint deployed, grant a session key, then revoke, rotate or let it expire as the finding describes. Sign a user operation with the old key and record whether validation still accepts it. Use only a disposable local fork or test deployment. Start at `BatchSessionKeyAccount.sol:52`, exercise the flagged path, and capture the transaction result, events, state difference, and trace. Never use production keys or broadcast this test.

Verification: confirmed

Fix validation: Add a regression test for the intended invariant, re-run the audit, then replay the relevant flow on a disposable fork.

Platform Attestation

This assessment was generated by Grisoco AI using its automated audit workflow.

Attestation ID: AUD-8FB086F3A1AD014F

Report fingerprint (SHA-256): 8fb086f3a1ad014f2ad869009c852ab698a4638497588a42a8a3e8ee5eb3c2f9

Digital signature (HMAC-SHA256): d29bbab213a7b9e356617ba4145d6c5d2bd3d451c5408f80bafd21560bf23ee2

Meaning: the fingerprint binds this report's contents and timestamp to the Grisoco AI workflow. The signature is generated with a private key held only by Grisoco AI's server — a copy of this report with edited findings, score, or scope will not carry a signature Grisoco AI would recompute for that content. Neither is a personal auditor's signature or a legal warranty.

GRISOCO AI AUDIT RECORD
AUD-8FB086F3A1AD014F

