



SMART CONTRACT SECURITY AUDIT

Clique

EIP-7702 Delegate and Penalty Hook

CLIENT

Clique

LANGUAGE

Solidity

REVIEW

November 2025

DOCUMENT

Document information

Project	Clique, EIP-7702 Delegate and Penalty Hook
Client	Clique
Platform	Not recorded in the original report
Language	Solidity
Repository	CliqueOfficial/cliquestock: src/hook/LinearPenaltyHook.sol
Commit	Not recorded in the original report
Methods	Manual code review, static analysis, and best practice verification
Assessed	November 2025
Report date	4 November 2025

Document history

VERSION	DATE	CHANGE
1.0	4 November 2025	Original report issued View original report
1.1	October 2026	Reformatted into the Safe Edges standard template. Findings unchanged

This report was reformatted into the Safe Edges standard template in October 2026. Findings, severities, recommendations and code references are unchanged from the original report, which is preserved unchanged in the [Safe Edges Audit Portfolio on GitHub](#).

CONTENTS

Contents

01	Executive summary	3
02	Scope	4
03	Methodology	5
04	Severity and status	6
05	Findings summary	7
06	Detailed findings	8
	L-01 Weak distributor access assumption in LinearPenaltyHook.execute	8
	L-02 Weak replay protection in DelegateMultiExec.multiExec	8
	L-03 Use rounding up in LinearPenaltyHook.getPenalty to better discourage early claimers	9
	I-01 Potential Gas grieving risk in DelegateMultiExec.multiExec	9
	G-01 Avoid zero assignment in for loops counters for gas efficiency	10
	Disclaimer	11

Executive summary

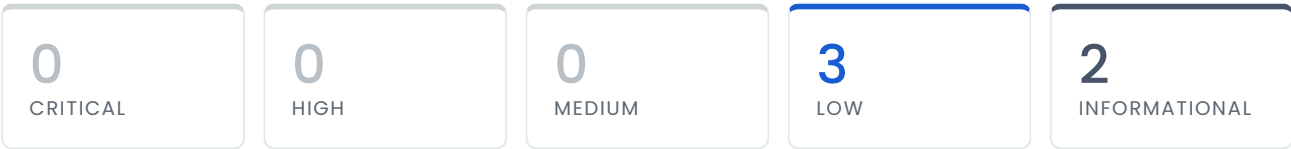
About Clique

Clique’s EIP-7702 work adds DelegateMultiExec, which executes signed batches of calls, and LinearPenaltyHook, which charges a time-based penalty on early claims from vesting allocations.

Summary of the review

Safe Edges reviewed Clique's LinearPenaltyHook and DelegateMultiExec contracts, built around EIP-7702 delegation, in November 2025. Five issues were identified: three Low and two Informational, one of which is a gas optimisation. The Low issues cover the hook’s reliance on being called only by the distributor, a signed-batch nonce that is never checked on-chain, and penalty rounding that slightly favours early claimers. At the time of the report the findings were under review with fixes pending.

Findings by severity



Findings by status

STATUS	ISSUES
● Open	5

Scope

Repository	CliqueOfficial/cliQUE-lock: src/hook/LinearPenaltyHook.sol
Commit	Not recorded in the original report
Platform	Not recorded in the original report
Language	Solidity

IN-SCOPE FILES

LinearPenaltyHook.sol
DelegateMultiExec (reviewed in the findings, not listed in the original scope)

Only the code at the commit above was reviewed. Changes made after this commit are not covered by this report.

Methodology

Methods recorded for this engagement: Manual code review, static analysis, and best practice verification.

Manual review

Auditors read the in-scope code line by line, tracing control flow, state changes and trust boundaries against the intended behaviour described by the client.

Threat modelling

We map privileged roles, external dependencies and value flows, then test the assumptions the code makes about each of them.

Functional and automated testing

Where tooling exists for the language and chain, we run static analysis and targeted tests to confirm suspected issues and check edge cases.

Fix review

After the client addresses the findings, we review the updated code and record the final status of each issue.

Severity and status

Severity levels

SEVERITY	DEFINITION
Critical	Direct loss or permanent lock of funds, or full compromise of the protocol, with few or no preconditions. Must be fixed before deployment.
High	Significant loss of funds, broken core functionality or manipulation of critical state under realistic conditions. Should be fixed before deployment.
Medium	Limited loss or degraded functionality, or issues that need specific conditions or privileged access to exploit. Should be fixed.
Low	Minor impact, low likelihood, or deviations from best practice. Fix when practical.
Informational	Code quality, documentation, gas and style observations with no direct security impact.

Status definitions

STATUS	DEFINITION
Resolved	The issue was fixed and the fix was verified.
Partially resolved	The risk was reduced but not fully removed.
Acknowledged	The client accepted the issue without changing the code.
Open	The issue was not addressed at the time of the report.
Not recorded	The original report did not record a resolution status for this issue.

Findings summary

ID	TITLE	SEVERITY	STATUS
L-01	Weak distributor access assumption in LinearPenaltyHook.execute	Low	● Open
L-02	Weak replay protection in DelegateMultiExec.multiExec	Low	● Open
L-03	Use rounding up in LinearPenaltyHook.getPenalty to better discourage early claimers	Low	● Open
I-01	Potential Gas griefing risk in DelegateMultiExec.multiExec	Informational	● Open
G-01	Avoid zero assignment in for loops counters for gas efficiency	Informational	● Open

Detailed findings

L-01 Weak distributor access assumption in LinearPenaltyHook.execute

Severity **Low** Status ● Open

DESCRIPTION

The execute function in LinearPenaltyHook computes and returns a penalty based on the provided allocation amount and timestamp.

However, the logic implicitly assumes that only the authorized distributor contract will invoke this function, as it derives the `_configurationId` via `_root`. In practice, there are no access control checks enforcing that only the distributor can call execute.

```
function execute(
    address,
    address _recipient,
    bytes32 _root,
    uint256 _allocatedamount,
    bytes memory _extra
) external returns (uint256 _consumed) {
    ClaimExtra memory claimExtra = abi.decode(_extra, (ClaimExtra));
    @>> bytes32 _configurationId = _distributor().configurationId(_root);
```

Anyone can call this function to bypass the implicit `_distributor()` caller check and preview the penalty. Since, the function does not perform any state changes, this does not directly damage the protocol.

IMPACT

While this does not lead to any exploit or unauthorized state modification, it represents a weak trust assumption regarding the source of calls and could expose internal logic to unintended external queries.

RECOMMENDED MITIGATION

If this function is intended to be used exclusively by the distributor, explicitly restrict its access.

L-02 Weak replay protection in DelegateMultiExec.multiExec

Severity **Low** Status ● Open

DESCRIPTION

`multiExec` constructs an ECDSA-signed digest that includes a nonce field, but the contract does not persist or check any on-chain state for that nonce. Because of that, a previously observed valid signed message can be re-submitted repeatedly and will pass verification each time, allowing the same batch of calls to be executed multiple times.

It is possible that the nonce is being managed off-chain; however, if that is not the case, it should be incremented or updated on-chain after each successful execution to prevent reuse.

IMPACT

A signed batch can be replayed using previously valid call data if the nonce is not properly updated or tracked.

RECOMMENDED MITIGATION

implement an on-chain sequential nonce that increments after each successful execution to ensure every signed batch can be used only once.

L-03 Use rounding up in LinearPenaltyHook.getPenalty to better discourage early claimers

Severity Low Status ● Open

DESCRIPTION

The getPenalty function calculates the claim penalty proportionally based on the remaining vesting duration:

```
uint256 penalty = (_allocatedamount * restTime) / totalTime;
```

By default, Solidity's integer division rounds down, meaning the computed penalty may be slightly lower than the mathematically precise ratio.

Since the purpose of the penalty mechanism is to discourage early claims, this rounding direction slightly favors early claimers.

IMPACT

Minor underestimation of penalties for early claimers due to rounding down.

RECOMMENDED MITIGATION

Use rounding up to ensure the penalty fully reflects the intended ratio, for example :

```
penalty = FixedPointMathLib.mulDivUp(_allocatedamount, restTime, totalTime);
```

I-01 Potential Gas griefing risk in DelegateMultiExec.multiExec

Severity Informational Status ● Open

DESCRIPTION

multiExec iterates over an array of user-provided calls and executes each via:

```
(bool success, bytes memory returnData) = callEntry.target.call{ value: callEntry.value }(callEntry.data);
```

Because call is invoked without an explicit gas: stipend, it forwards all remaining gas to the target. A malicious or poorly implemented target (for example, a contract fallback or receive() that runs an expensive loop or may return large returnData) can consume most or all gas for the transaction. If callEntry.allowFailure is false, such consumption can cause the whole multiExec to revert; if allowFailure is true, it still wastes the relayers gas and can cause subsequent calls to fail or be starved of gas.

IMPACT

This issue can cause a denial-of-service condition, allowing a single malicious target to halt or disrupt the execution of the entire batch.

RECOMMENDED MITIGATION

Avoid storing or returning large returnData values in memory to minimize gas overhead from memory copying. Additionally, if feasible, restrict multiExec execution to trusted or whitelisted target contracts whenever possible.

G-01 Avoid zero assignment in for loops counters for gas efficiency

Severity **Informational** Status ● Open Category Gas

DESCRIPTION

To optimize gas efficiency in a for loop, it is recommended to avoid explicitly setting the counter to zero, as it is already initialized to zero by default. Removing unnecessary variable assignments will further enhance the contract's gas efficiency.

```
- for (uint256 i = 0; i < ...; ) {...}
+ for (uint256 i; i < ...; ) {...}
```

Disclaimer

This report is based on the code at the commit listed in the Scope section and on information the client provided during the review. It reflects Safe Edges' assessment at the time of the review and is not a guarantee that the code is free of vulnerabilities.

Changes made after the reviewed commit, deployment configuration, off-chain components and third-party dependencies are outside the scope of this report unless stated otherwise.

Security is ongoing. We recommend continued testing, monitoring and bug bounty coverage alongside any audit.

This report is not investment advice and is not an endorsement of the project, its team or any token.



Safe Edges is a Web3 security firm and official security partner for the Fuel, ENI and XI EcoChain ecosystems. We work across smart contract audits, Rust and WASM reviews, ZK circuit review, AI and MCP security, and penetration testing.

200+

PROJECTS AUDITED

\$4B+

TVL SECURED

0

POST-AUDIT HACKS