



SMART CONTRACT SECURITY AUDIT

Clique

b3 Detailed Findings

CLIENT

Clique

LANGUAGE

Solidity

REVIEW

March 2025

DOCUMENT

Document information

Project	Clique, b3 Detailed Findings
Client	Clique
Platform	Not recorded in the original report
Language	Solidity
Repository	Not recorded in the original report
Commit	Not recorded in the original report
Methods	Not recorded in the original report
Report date	20 March 2025

Document history

VERSION	DATE	CHANGE
1.0	20 March 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
	M-01 Incorrect totalPieces calculation in CliqueLock:calculateClaimableAmount can leads to unevenly8 vesting amount distribution	
	M-02 Anyone can initiate claim of any stream by using just their streamId in CliqueLock:claim function	9
	M-03 CliqueDistributorManager:createDistributor should explicit check for signature non-malleability	10
	L-01 Excessive sent ether would not be refunded to the user in Distributor:claim function	10
	L-02 Missing fee validation checks in Distributor:setFee function	11
	L-03 Lack of multiple validations in CliqueLock:_createStream before creating stream leads to spam creations.	12
	I-01 Twice imports of ECDSA library in CliqueDistributorManager.sol	12
	I-02 Missing check for signature usage in CliqueDistributorManager:createDistributor can leads to same signature being used multiple times	13
	I-03 Consider using SafeERC20 for IERC20 to make the transfer more secure.	13
	I-04 Missing checks for address(0) when assigning values to address state variables	14
	I-05 Remove Unused Custom Error to save more gas	14
	Disclaimer	15

Executive summary

About Clique

These contracts form part of the Clique token distribution and management system: CliqueLock handles vesting streams, and Distributor and CliqueDistributorManager handle airdrop claims and distributor creation.

Summary of the review

This is the detailed issue write-up from Safe Edges' March 2025 review of Clique's token distribution and vesting contracts (CliqueLock, Distributor and CliqueDistributorManager). It sets out eleven issues, three Medium, three Low and five Informational, with the code and recommended fixes for each. It does not record resolution statuses. The fix statuses for the same eleven findings are recorded in the companion final report, Safe-Edges_Clique_b3-Distribution-and-Vesting_2025-03.

Findings by severity



Findings by status

STATUS	ISSUES
● Not recorded	11

Scope

Repository	Not recorded in the original report
------------	-------------------------------------

Commit	Not recorded in the original report
--------	-------------------------------------

Platform	Not recorded in the original report
----------	-------------------------------------

Language	Solidity
----------	----------

IN-SCOPE FILES

```
CliqueLock
Distributor
CliqueDistributorManager
```

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: Not recorded in the original report.

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
M-01	Incorrect totalPieces calculation in CliqueLock:calculateClaimableAmount can leads to unevenly vesting amount distribution	Medium	● Not recorded
M-02	Anyone can initiate claim of any stream by using just their streamId in CliqueLock:claim function	Medium	● Not recorded
M-03	CliqueDistributorManager:createDistributor should explicit check for signature non-malleability	Medium	● Not recorded
L-01	Excessive sent ether would not be refunded to the user in Distributor:claim function	Low	● Not recorded
L-02	Missing fee validation checks in Distributor:setFee function	Low	● Not recorded
L-03	Lack of multiple validations in CliqueLock:_createStream before creating stream leads to spam creations.	Low	● Not recorded
I-01	Twice imports of ECDSA library in CliqueDistributorManager.sol	Informational	● Not recorded
I-02	Missing check for signature usage in CliqueDistributorManager:createDistributor can leads to same signature being used multiple times	Informational	● Not recorded
I-03	Consider using SafeERC20 for IERC20 to make the transfer more secure.	Informational	● Not recorded
I-04	Missing checks for address(0) when assigning values to address state variables	Informational	● Not recorded
I-05	Remove Unused Custom Error to save more gas	Informational	● Not recorded

Detailed findings

M-01 Incorrect totalPieces calculation in CliqueLock:calculateClaimableAmount can leads to unevenly vesting amount distribution

Severity **Medium** Status ● Not recorded

DESCRIPTION

The current calculates the totalPieces like this:

```
uint256 totalPieces = vestingDuration / stream.pieceDuration + 1;
```

This works completely fine when the $\text{vestingDuration} \% \text{pieceDuration} \neq 0$. Like in scenario where let's say vestingDuration is 13 months and pieceDuration is 3 months, then this would return totalPieces as 5. However when vestingDuration is 12 months and pieceDuration is 3 months, then again this would create 5 totalPieces . Now the problem with such piece calculation is that it can create unevenly vesting amount, like in our case, it would allow to redeem 60% of vested amount for first 3 pieceDuration , and in the 4th piece (which is from 9 months -12 months), it will allow rest 40%. The way it is allowing the largest chunk in last piece, is because after 12 months, this condition will satisfy which will return the whole amount

```
if (block.timestamp >= stream.endTime) {  
    return stream.amount;  
}
```

IMPACT

This leads to vestedAmount to be distributed unevenly potentially forcing the last piece getting the larger chunk than other pieces.

RECOMMENDED MITIGATION

You can first check using modulus whether 1 should be added to totalPieces or not.

M-02 Anyone can initiate claim of any stream by using just their streamId in CliqueLock:claim function

Severity **Medium** Status ● Not recorded

DESCRIPTION

Claim takes only streamId as argument, and then fetch the recipient of the exact claimable amount by calling `_resolveRecipient` function. However, it does not ensure that the caller is the rightful receiver. This creates a scenario where one malicious user can initiate the claim of other streams by using just their streamId even if the receiver did not intend to do so.

IMPACT

Malicious user can initiate the claim of other streams by using their streamId.

RECOMMENDED MITIGATION

Consider implementing this check which will allow only the receiver to initiate the claim:

```
function claim(uint256 streamId) external {
    Stream memory stream = streams[streamId];
    address recipient = _resolveRecipient(streamId, stream);
+   if (recipient != msg.sender) {
+       revert InvalidRecipient();
+   }
    if (stream.amount == 0) {
        revert StreamNotFound(streamId);
    }
    ...}
```

M-03 CliqueDistributorManager:createDistributor should explicit check for signature non-malleability

Severity **Medium** Status ● Not recorded

DESCRIPTION

createDistributor function calls ECDSA.recover function to recover signer based on the hash, and signature. However, solady's ECDSA does not implement signature uniqueness check like Openzeppelin's ECDSA, which makes sure that the s field in the signature should be in the lower half order. Since during recovering s can potentially have two values, implementing such check prevent such scenarios.

IMPACT

this issue can potentially create malleable signature

RECOMMENDED MITIGATION

Consider either using Openzeppelin ECDSA library. Otherwise, integrating this check into the contract will ensure that signatures cannot be manipulated to create another valid signature from the original one.

```
if (uint256(s) > 0x7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff) {  
    return false;  
}
```

L-01 Excessive sent ether would not be refunded to the user in Distributor:claim function

Severity **Low** Status ● Not recorded

DESCRIPTION

For being able to claim the airdrop tokens, the user must first sent the fee. Now the current logic only checks whether the amount is greater than fee, even if the amount is equals to the fee the call would get reverted. Moreover, the excess ether would not be refunded to the user. Only the owner can withdraw funds from this contracts. This create a interesting scenario, where the user must have to pay more than the fee in order to recieve the airdrop token, knowinglly that their excess would be stuck there atleast for them.

```
function claim(
    bytes32[] calldata _proof,
    bytes calldata _signature,
    uint256 _amount,
    uint256 _index,
    address _onBehalfOf
) external payable whenActive {
    @> if (msg.value < fee) {
        revert InsufficientFee();
    }
```

IMPACT

Excess ether would not be refunded to the user leading to user's loss

RECOMMENDED MITIGATION

Consider implementing these changes:

```
+ error IncorrectFee();

function claim(
    bytes32[] calldata _proof,
    bytes calldata _signature,
    uint256 _amount,
    uint256 _index,
    address _onBehalfOf
) external payable whenActive {
+   if (msg.value != fee) {
+       revert IncorrectFee();
+   }
```

L-02 Missing fee validation checks in Distributor:setFee function

Severity **Low** Status ● Not recorded

DESCRIPTION

setFee can only be called by the deployer. However, it misses critical checks which puts restriction on the fee limit.

RECOMMENDED MITIGATION

Consider adding minFee and maxFee restriction limit in the setFee function.

L-03 Lack of multiple validations in CliqueLock:_createStream before creating stream leads to spam creations.

Severity **Low** Status ● Not recorded

DESCRIPTION

The _createStream function, creates a stream using the caller feed stream struct. However, during the creation lacks multiple validation checks related to the amount (>0) and current timestamp less than startTime, cliffTime and endTime. Lacks of such check can leads to creation of stream that just spam the system using its resources and does not actually be useful.

RECOMMENDED MITIGATION

consider adding a the validation check in the function:

```
+ error InvalidAmount();

function _createStream(StreamConfig calldata config) internal returns (uint256) {
    uint256 streamId = nextStreamId++;
+   if (config.amount == 0) {
+       revert InvalidAmount();
+   }
+   if (config.startTime > config.cliffTime || block.timestamp > config.startTime ||
config.cliffTime > config.endTime) {
        revert InvalidSchedule();
    }
}
```

I-01 Twice imports of ECDSA library in CliqueDistributorManager.sol

Severity **Informational** Status ● Not recorded

DESCRIPTION

Makes the code more redundant

RECOMMENDED MITIGATION

Remove one imports out of two:

```
import "solady/utils/MerkleProofLib.sol";
import "solady/utils/ECDSA.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "solady/utils/FixedPointMathLib.sol";
import {UUPSUpgradeable} from "solady/utils/UUPSUpgradeable.sol";
import {Initializable} from "solady/utils/Initializable.sol";
import {OwnableRoles} from "solady/auth/OwnableRoles.sol";
import {EIP712} from "solady/utils/EIP712.sol";
import {Distributor} from "../sub/Distributor.sol";
- import {ECDSA} from "solady/utils/ECDSA.sol";
```

I-02 Missing check for signature usage in CliqueDistributorManager:createDistributor can leads to same signature being used multiple times

Severity **Informational** Status ● Not recorded

DESCRIPTION

The current logic uses deadline as replay protection. However, it does not revert the call if the same signature is used multiple times to create Distributor within the signed deadline. This would create spam distributor creation. This can be avoided if you use nonce that keep updated once signature being used, making it more secure against replay attack scenarios.

```
bytes32 digest = _hashTypedData(
    keccak256(
        abi.encode(
            keccak256(
                @> "NewDistributor(address token, address vault, address signer, uint256 fee,
                uint256 deadline)"
            ),
            _token,
            _vault,
            _signer,
            _fee,
            _deadline
        )
    )
);
```

IMPACT

User can replay a signature multiple times withing the signed deadline

RECOMMENDED MITIGATION

Consider adding nonce in the signature hash.

I-03 Consider using SafeERC20 for IERC20 to make the transfer more secure.

Severity **Informational** Status ● Not recorded

DESCRIPTION

It is recommended to use OpenZeppelin's SafeERC20 library.

```
+ import {SafeERC20} from "openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol";
+ using SafeERC20 for IERC20;
```

I-04 Missing checks for address(0) when assigning values to address state variables

Severity **Informational** Status ● Not recorded

DESCRIPTION

Check for address(0) when assigning values to address state variables.

FOUND IN DISTRIBUTOR.SOL

```
function setSigner(address _signer) external onlyRoles(PROJECT_ADMIN) {}
```

FOUND IN DISTRIBUTOR.SOL

```
vault = _vault;
```

FOUND IN CLIQUEDISTRIBUTORMANAGER.SOL

```
function createDistributor(  
    address _token,  
    address _vault,  
    address _signer,  
    uint256 _fee,  
    uint256 _deadline,  
    bytes calldata _signature  
) external returns (address) {}
```

I-05 Remove Unused Custom Error to save more gas

Severity **Informational** Status ● Not recorded

DESCRIPTION

it is recommended to remove the custom error when it remain unused

FOUND IN CLIQUEDISTRIBUTORMANAGER.SOL

```
error InvalidSignature();
```

FOUND IN DISTRIBUTOR.SOL

```
error InvalidMerkleRootIndex();
```

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