Safe Edges

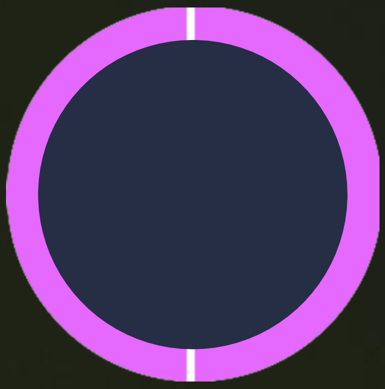
Audit Report March, 2025

Table of Content

Executive Summary

Project Name	Clique
Project URL	https://github.com/clique-external/b3-contracts/tree/main
Overview	These contracts form part of the Clique token distribution and management system, providing secure and efficient mechanisms for token distribution and vesting.
Audit Scope	https://github.com/clique-external/b3-contracts/tree/main
Commit	-
Language	Solidity
Blockchain	Eth
Method	Manual Analysis, Functional Testing, Automated Testing
First Review	17 st March 2025 - 20th March 2025
Updated Code Received	20th March 2025
Second Review	20th March 2025

Number of Security Issues per Severity



🟥 High 🟪 Medium

🟩 Low 🟨 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	2	2	2
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	1	2	2

Checked Vulnerabilities

- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Computations Correctness
- ✓ Race conditions/front running
- ✓ Re-entrancy
- ✓ Malicious libraries
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Missing Zero Address Validation
- ✓ Revert/require functions
- ✓ Upgradeable safety
- ✓ Using inline assembly
- ✓ Style guide violation
- ✓ Parallel Execution safety
- ✓ UTXO Model Verification
- ✓ FuelVM Opcodes
- ✓ Cross-Chain Interactions
- ✓ Modular Design
- ✓ Access Control Vulnerabilities
- ✓ Denial of Service (DoS)
- ✓ Oracle Manipulation
- ✓ Signature Replay Attacks
- ✓ Improper Handling of External Calls
- ✓ Proxy Storage Collision
- ✓ Use of Deprecated Functions

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistic analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract’s performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Issue Details

[M-01] Incorrect totalPieces calculation in CliqueLock:calculateClaimableAmount can lead to uneven vesting amount distribution

Description

The function calculates totalPieces as follows:

This works correctly when vestingDuration % pieceDuration != 0. However, when vestingDuration is exactly divisible by pieceDuration, an extra piece is added unnecessarily, causing uneven vesting distribution. For example:

- vestingDuration = 13 months, pieceDuration = 3 months → totalPieces = 5 (correct)
- vestingDuration = 12 months, pieceDuration = 3 months → totalPieces = 5 (incorrect, should be 4)

As a result, the last vesting period may receive a disproportionately large share of the vested amount.

Impact

Leads to uneven vesting, potentially delaying or accelerating fund distribution unfairly.

Recommended Mitigation

Before adding 1 to totalPieces, check if vestingDuration % pieceDuration == 0.

Status - Fixed

[M-02] Anyone can initiate claims for any stream using just the streamId in CliqueLock:claim

Description

The claim function takes only streamId as an argument and fetches the recipient dynamically:

However, it does not check if the caller (msg.sender) is the intended recipient. This allows a malicious actor to claim another user's stream.

Impact

Attackers can steal vested tokens by providing another user's streamId.

Recommended Mitigation

Ensure that only the rightful recipient can initiate the claim:

Status - Acknowledge

[M-03] CliqueDistributorManager:createDistributor should explicitly check for signature non-malleability

Description

The function recovers the signer from a hash and signature but does not ensure that the s field is in the lower half order. This allows signature malleability.

Impact

Attackers can manipulate valid signatures to create alternate valid signatures.

Recommended Mitigation

Either use OpenZeppelin’s ECDSA library or add a check to enforce s being in the lower half:

Status - Acknowledge

[L-01] Excessive Ether sent to Distributor:claim is not refunded

Description

Users must send a fee to claim tokens. If they send excess Ether, it is not refunded.

Impact

Users may lose excess Ether when claiming tokens.

Recommended Mitigation

Ensure the exact fee is sent:

Status - Acknowledge

[L-02] Missing fee validation checks in Distributor:setFee

Description

The function lacks constraints on the fee range.

Recommended Mitigation

Define minFee and maxFee limits.

Status - Fixed

[L-03] CliqueLock:_createStream lacks multiple validations before creating a stream

Description

The function does not validate amount > 0, startTime < cliffTime < endTime, or block.timestamp < startTime.

Impact

Allows creation of invalid or spam streams.

Recommended Mitigation

consider adding a the validation check in the function

Status - **Fixed**

[I-01] Duplicate import of ECDSA in CliqueDistributorManager.sol

Description

The ECDSA library is imported twice, making the code redundant.

Recommended Mitigation

Remove one import

Status - **Fixed**

[I-02] Missing check for signature usage in CliqueDistributorManager:createDistributor

Description

The function relies on deadline for replay protection but does not prevent multiple uses of the same signature within that timeframe.

Impact

Allows signature reuse, leading to spam distributor creation.

Recommended Mitigation

Include a nonce in the signature hash to prevent replay attacks.

Status - **Acknowledge**

[I-03] Use SafeERC20 for secure token transfers

Description

It is recommended to use OpenZeppelin’s SafeERC20 library for secure transfers.

Status - Acknowledge

[I-04] Missing address(0) checks for critical state variables

Description

Certain functions do not check for address(0), which could introduce errors.

- Found in Distributor.sol
- Found in CliqueDistributorManager.sol

Recommended Mitigation

Validate addresses before assignment.

Status - Acknowledge

[I-05] Remove unused custom errors to optimize gas usage

Description

Unused custom errors increase contract size and gas costs.

- Found in CliqueDistributorManager.sol
- Found in Distributor.sol

Recommended Mitigation

Remove these errors if they are not used anywhere in the contract

Status - Fixed

Closing Summary

In this report, we have considered the security of Clique. We performed our audit according to the procedure described above.

Some issues of informational severity were found,which the Clique Team has Fixed.

Disclaimer

SafeEdges Smart contract security audit provides services to help identify and mitigate potential security risks in Clique. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. SafeEdges audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Hyperlane. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

SafeEdges cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of Clique to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About SAFE EDGES

SafeEdges is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

500+
Audits Completed

$3B
Secured

600k+
Lines of Code Audited

Audit Report
March, 2025

https://safeedges.in
info@safeedges.in

Safe Edges