QuillAudits

AUDIT REPORT

September 2025

For

CLIQUE

Table of Content

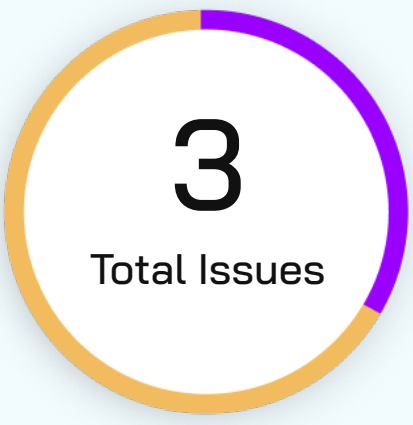
Executive Summary

Project Name	Clique
Project URL	https://www.clique.tech/
Overview	Clique Lock Sui is a decentralised airdrop distribution system built on the Sui blockchain that enables projects to distribute tokens to users with customizable fee structures and cryptographic verification. The system supports both simple batch-based distributions and merkle tree-based proofs, allowing administrators to configure fees per batch while users pay fees in designated tokens to claim their allocated airdrops with ECDSA signature validation
Audit Scope	The Scope of the Audit is to analyse the Code Quality ,Security and Correctness of Clique Smart Contract
Branch	Main
Commit Hash	9bd812e02334a3f652bbc5b8fd5567531960e0f6
Contracts in Scope	base_distributor.move bytes32.move fee_mode.move roles.move simple_dispersal.move simple_distributor.move
Language	Move
Blockchain	Sui
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	18th September 2025 - 24th September 2025
Updated Code Received	28th September 2025
Review 2	29th September 2025
Fixed In	e150b3540ee7ecf9a198be0b5c771f08c908cb20

Verify the Authenticity of Report on QuillAudits Leaderboard:

https://www.quillaudits.com/leaderboard

Number of Issues per Severity



Critical	0 (0.0%)
High	0 (0.0%)
Medium	0 (0.0%)
Low	2 (66.6%)
Informational	1 (33.3%)

Severity

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

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Missing fee rate validation allows excessive percentage fees exceeding the claimed amount value	Low	Acknowledged
2	Missing batch configuration validation enables unintended fee structures in claim processing	Low	Resolved
3	Inconsistent capability parameter positioning reduces authorization clarity and reading efficiency	Informational	Resolved

Checked Vulnerabilities

- Transaction-ordering dependence
- Timestamp dependence
- Denial of service / logical oversights
- Timestamp dependence
- Access control
- Code clones, functionality duplication
- Witness Type
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Business logic contradicting the specification
- Number of rounding errors
- Gas usage
- Unchecked CALL Return Values
- Centralization of power

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
- 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.

Types of Severity

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

Critical: Immediate and Catastrophic Impact

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

High (H): Significant Risk of Major Loss or Compromise

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

Medium (M): Potential for Moderate Harm Under Specific Circumstances

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

Low (L): Minor Imperfections with Limited Repercussions

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

Informational (I): Opportunities for Improvement, Not Immediate Risks

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.

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.

Severity Matrix

Impact

Likelihood	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

Low Severity Issues

Missing fee rate validation allows excessive percentage fees exceeding the claimed amount value

Acknowledged

Path

sources/base_distributor.move#L193-205

Function Name

`set_fee_mode`

Description

The set_fee_mode function enables project administrators to configure fee modes for specific batch identifiers without validating the fee_rate parameter in Singer Tier fee modes. The function directly stores the provided FeeMode enum variant into the fee_modes table without checking whether the fee_rate exceeds the FEE_RATE_DENOMINATOR constant defined as 1,000,000,000,000,000,000 in sources/fee_mode.move#L16.

However, the fee calculation logic in sources/fee_mode.move#L18-31 computes fees using the formula (amount * fee_rate) / FEE_RATE_DENOMINATOR, where fee_rate values equal to or exceeding FEE_RATE_DENOMINATOR result in percentage fees of 100% or higher. When administrators accidentally or intentionally set fee_rate values above FEE_RATE_DENOMINATOR, users end up paying fees that equal or exceed the value of tokens they claim, creating an economically irrational fee structure where claiming becomes unprofitable.

Consequently, this validation gap enables scenarios where users pay more in fees than the value of airdrop tokens they receive, effectively making the airdrop claiming process economically disadvantageous. Administrative errors such as entering fee rates in incorrect units or adding extra zeros could accidentally create fee rates of 200%, 1000%, or higher, causing users to lose significant value when claiming their allocated tokens.

Recommendation

We recommend adding validation in the set_fee_mode function to ensure that fee_rate values in SingerTier fee modes do not exceed FEE_RATE_DENOMINATOR, preventing fee percentages above 100% and maintaining economically reasonable fee structures for airdrop claiming operations.

Clique Team's Comment

The fee token and the airdrop token is not necessarily the same token, so it's possible that fee rate is >= 100%. For example, if the airdrop token's decimal is very large, and we collect fee in SUI (whose decimal is 9), it's highly likely that the fee rate is > 100% (and maybe even >10000000%)

Missing batch configuration validation enables unintended fee structures in claim processing

Resolved

Path

sources/base_distributor.move#L274-278

Description

The claim_check function calculates expected fees by calling self.required_fee(batch_id, amount) without validating whether the provided batch_id has been explicitly configured via set_fee_mode. The function processes claims for any batch_id value provided by users, relying on the required_fee function in lines 215-222, which returns the fee value to 0 if the administrator of the projects forgets to configure a default fee or fee mode for a given batch_id.

This design enables users to successfully claim airdrops using valid signatures while paying fees that may not reflect the intended economic structure for that particular batch, which could be zero if the batch is not configured properly.

Consequently, administrative oversight in fee configuration can lead to unintended economic outcomes where users pay zero fees for specific airdrop batches.

Recommendation

We recommend adding explicit validation in the claim_check function to verify that the batch_id has been configured via set_fee_mode before processing claims, ensuring that only intentionally configured batch identifiers can be used for airdrop distribution and preventing accidental fee structure deviations.

Clique Team's Comment

If changing the code in this way, the default fee mode will be not used, but we want there to be a default fee mode.

So instead, when creating a distributor, the default fee mode is set to a very large fixed amount, ensuring nobody could claim before administrator sets a fee explicitly

Informational Issues

Inconsistent capability parameter positioning reduces authorization clarity and reading efficiency

Resolved

Description

In sources/base_distributor.move#L96-101, the new_base_distributor function places the ProjectAdminCap parameter as the third argument after signer_pubkey and vault, while similar patterns appear in sources/simple_dispersal.move#L17 and sources/simple_distributor.move#L17, where capability parameters are positioned mid-way through parameter lists rather than as the first argument. Similarly, in other configuration functions like set_fee_mode, set_signer_pubkey etc.

However, this positioning violates Sui Move best practices where authorization capabilities should be placed first to make function authorization requirements immediately apparent. When capabilities are buried within parameter lists, developers and readers of the code must scan through arguments to identify authorization checks, increasing cognitive load and reducing code clarity.

Consequently, this inconsistent positioning can lead to authorization oversights during development and auditing, as security-critical requirements become less obvious when capabilities are not prominently placed.

Recommendation

We recommend repositioning all capability parameters as the first argument in function signatures throughout the codebase, following Sui Move best practices to make authorization requirements immediately visible and maintain consistency with Sui framework conventions for enhanced code clarity and security audit efficiency.

Clique Team's Comment

We've moved the capability parameters to the first (if there's no "self" object) or second (if there's "self" object)

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

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

Two issues of low and one issue of informational severity were found. One of the issues was acknowleged and others were resolved.

Disclaimer

QuillAudits 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. QuillAudits 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 Clique. 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.

QuillAudits 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 QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With seven years of expertise, we've secured over 1400 projects globally, averting over $3 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.

7+ Years of Expertise	**1M+** Lines of Code Audited
50+ Chains Supported	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

September 2025

For

CLIQUE

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com