



SMART CONTRACT SECURITY AUDIT

Clique

Lock Program

CLIENT

Clique

PLATFORM

Solana

LANGUAGE

Rust (Anchor)

REVIEW

June 2025

DOCUMENT

Document information

Project	Clique, Lock Program
Client	Clique
Platform	Solana
Language	Rust (Anchor)
Repository	Not recorded in the original report
Commit	Not recorded in the original report
Methods	Not recorded in the original report
Report date	June 2025 (original says 2025; date taken from the PDF)

Document history

VERSION	DATE	CHANGE
1.0	June 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 User Balance Is Not Recorded During Deposit	8
	M-02 Unauthorized User Can Front-Run Vault Initialization	9
	M-03 Unauthorized User Can Front-Run Vesting Initialization	9
	M-04 Invalid Schedule Can Bypass Vesting Time Constraints	10
	L-01 Missing Default Address Validation	11
	Disclaimer	12

Executive summary

About Clique

Clique's lock program on Solana holds SOL deposits in a vault operated by an authority and relayers, and creates token vesting schedules for beneficiaries with a start time, cliff and end time.

Summary of the review

Safe Edges reviewed Clique's Solana lock program, covering the deposit vault and the vesting module, in 2025. Five issues were identified: four Medium and one Low. The Medium issues covered deposits that are never recorded against the user, vault and vesting initialization that any account can front-run, and vesting schedules that can start in the past. Clique acknowledged all five.

Findings by severity



Findings by status

STATUS	ISSUES
● Acknowledged	5

Scope

Repository	Not recorded in the original report
Commit	Not recorded in the original report
Platform	Solana
Language	Rust (Anchor)

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	User Balance Is Not Recorded During Deposit	Medium	● Acknowledged
M-02	Unauthorized User Can Front-Run Vault Initialization	Medium	● Acknowledged
M-03	Unauthorized User Can Front-Run Vesting Initialization	Medium	● Acknowledged
M-04	Invalid Schedule Can Bypass Vesting Time Constraints	Medium	● Acknowledged
L-01	Missing Default Address Valdiation	Low	● Acknowledged

Detailed findings

M-01 User Balance Is Not Recorded During Deposit

Severity **Medium** Status ● Acknowledged

DESCRIPTION

The `deposit` function transfers SOL from the user to the vault but does not update any on-chain record of the user's deposited amount. Without tracking user balances, the system cannot verify ownership or enforce limits during future withdrawals or interactions, leading to loss of accounting integrity.

```
pub fn deposit(ctx: Context<Deposit>, amount: u64, recipient: Vec<u8>) -> Result<()>
{
    let vault = &mut ctx.accounts.vault;
    let user = &mut ctx.accounts.user;

    // Transfer SOL from user to vault
    let cpi_context = CpiContext::new(
        ctx.accounts.system_program.to_account_info(),
        anchor_lang::system_program::Transfer {
            from: user.to_account_info(),
            to: vault.to_account_info(),
        },
    );
    @> anchor_lang::system_program::transfer(cpi_context, amount)?;

    msg!("Deposited {} lamports to {}", amount, hex::encode(&recipient));
    Ok(())
}
```

IMPACT

Users who deposit SOL have no on-chain proof of deposit, making refunding or withdrawal logic unreliable or insecure.

RECOMMENDATION

Record the deposited amount in a user balance field after the transfer.

```
let user = &mut ctx.accounts.user;
+ user.balance += amount;
```

TEAM RESPONSE

Acknowledge

M-02 Unauthorized User Can Front-Run Vault Initialization

Severity **Medium** Status ● Acknowledged

DESCRIPTION

The `initialize` function sets the `vault.authority` and assigns relayers without verifying that the caller is an authorized signer. Although it configures sensitive roles like relayers and the authority, it lacks any `signer` constraint on the `authority` account. This allows any user to front-run the initialize call and assign themselves or malicious actors as relayers and authority, especially in scenarios where the vault PDA is publicly derivable.

IMPACT

An attacker can initialize the vault ahead of the legitimate user and seize control by setting themselves as authority and relayers, compromising the integrity of subsequent operations.

RECOMMENDATION

Add a signer validation with the deployer public address and change them if required.

TEAM RESPONSE

N/A

M-03 Unauthorized User Can Front-Run Vesting Initialization

Severity **Medium** Status ● Acknowledged

DESCRIPTION

The `initialize_vesting` function sets critical state such as `vesting.creator` and `beneficiary` but does not require the creator to be a `signer`. As a result, a malicious actor can front-run the intended initializer and configure vesting parameters for arbitrary recipients, assuming they have sufficient token balance for the transfer.

IMPACT

An attacker can seize control over a vesting contract by initializing it before the legitimate user, redirecting funds or vesting rights.

RECOMMENDATION

Require deployer to be a signer to ensure only the intended party can initialize the vesting contract.

TEAM RESPONSE

Acknowledge

M-04 Invalid Schedule Can Bypass Vesting Time Constraints

Severity **Medium** Status ● Acknowledged

DESCRIPTION

The `initialize_vesting` function verifies that `end_ts > start_ts`, but does not ensure that `start_ts` lies in the future relative to the current block time. This allows initialization of vesting schedules with `start_ts` in the past, effectively bypassing the intended lock-in period and possibly enabling immediate or retroactive vesting.

```
pub fn initialize_vesting(  
    ctx: Context<InitializeVesting>,  
    amount: u64,  
    start_ts: i64,  
    cliff_amount: u64,  
    end_ts: i64,  
) -> Result<()> {  
    let vesting = &mut ctx.accounts.vesting;  
  
    require!(end_ts > start_ts, VestingError::InvalidSchedule);  
    require!(amount > 0, VestingError::InvalidAmount);  
  
    ...  
}
```

IMPACT

A vesting contract can be initialized with a schedule that appears already elapsed, causing premature token release and defeating time-lock assumptions.

RECOMMENDATION

Add a constraint to validate that `start_ts` is greater than the current Unix timestamp.

TEAM RESPONSE

Acknowledge

L-01 Missing Default Address Validation

Severity Low Status ● Acknowledged

DESCRIPTION

In the `InitializeVesting` struct, `beneficiary` account is not checked for being the default zero address. This allows initializing a vesting schedule with an invalid or unclaimable beneficiary.

IMPACT

Tokens could be locked in a vesting schedule with no way for anyone to claim them.

RECOMMENDATION

Add a check to ensure the `beneficiary` is not the default address.

TEAM RESPONSE

Acknowledge

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