



SMART CONTRACT SECURITY AUDIT

Clique

SVM V2 Lock Hook and Streams

CLIENT

Clique

PLATFORM

SVM

LANGUAGE

Rust (Anchor)

REVIEW

January 2026

DOCUMENT

Document information

Project	Clique, SVM V2 Lock Hook and Streams
Client	Clique
Platform	SVM
Language	Rust (Anchor)
Repository	Not recorded in the original report
Commit	Not recorded in the original report
Methods	Not recorded in the original report
Assessed	January 2026

Document history

VERSION	DATE	CHANGE
1.0	January 2026	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 Attacker Can Block Token Distribution Recipients	8
	M-02 Accepting Mints With Freeze Authority Enables Post-Claim Freezes	8
	M-03 Operator Can Break Claim Flow Recipient	10
	L-01 Revocation Forfeits Vested but Unclaimed Tokens to Revoke Authority	11
	L-02 Dynamic Recipient Toggle Bricks Pending Claims	11
	L-03 Vault Parameter Should Use SystemAccount Type	12
	L-04 Base Account Constraint Prevents Protocol Integration	13
	L-05 Insufficient Timestamp Validation Allows Vesting Bypass via Instant or Pre-Vested Streams	14
	L-06 Missing Chain Identifier Enables Cross-Chain Signature Replay Attacks	14
	L-07 Caller Can Redirect NFT Mint To Incorrect Token Account	16
	I-01 Mutable Mint Allows Token Swap After Root Publication	17
	I-02 No Fee Caps Allow Unbounded Fee Extraction	17
	I-03 set_mint Lacks Mint Account Validation and Legacy Token Enforcement	18
	I-04 Frontrunning can pre-create stream ATA and DoS create_stream and create_nft_bound_stream	18
	I-05 Misspelled Helper Function receover_signer Hurts Clarity	19
	Disclaimer	21

Executive summary

About Clique

Version 2 of Clique's SVM programs adds a lock hook that turns claims into vesting streams, including NFT-bound streams minted with Token-2022.

Summary of the review

Safe Edges reviewed version 2 of Clique's SVM programs, covering the merkle distributor, the lock hook and vesting streams, in January 2026. Fifteen issues were identified: three Medium, seven Low and five Informational. The Medium issues concern pre-funded accounts blocking claim initialisation, mints with a freeze authority, and a missing account in the ATA creation CPI that breaks the claim flow. The original report does not record a resolution status for any finding.

Findings by severity



Findings by status

STATUS	ISSUES
● Not recorded	15

Scope

Repository	Not recorded in the original report
Commit	Not recorded in the original report
Platform	SVM
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	Attacker Can Block Token Distribution Recipients	Medium	● Not recorded
M-02	Accepting Mints With Freeze Authority Enables Post-Claim Freezes	Medium	● Not recorded
M-03	Operator Can Break Claim Flow Recipient	Medium	● Not recorded
L-01	Revocation Forfeits Vested but Unclaimed Tokens to Revoke Authority	Low	● Not recorded
L-02	Dynamic Recipient Toggle Bricks Pending Claims	Low	● Not recorded
L-03	Vault Parameter Should Use SystemAccount Type	Low	● Not recorded
L-04	Base Account Constraint Prevents Protocol Integration	Low	● Not recorded
L-05	Insufficient Timestamp Validation Allows Vesting Bypass via Instant or Pre-Vested Streams	Low	● Not recorded
L-06	Missing Chain Identifier Enables Cross-Chain Signature Replay Attacks	Low	● Not recorded
L-07	Caller Can Redirect NFT Mint To Incorrect Token Account	Low	● Not recorded
I-01	Mutable Mint Allows Token Swap After Root Publication	Informational	● Not recorded
I-02	No Fee Caps Allow Unbounded Fee Extraction	Informational	● Not recorded
I-03	set_mint Lacks Mint Account Validation and Legacy Token Enforcement	Informational	● Not recorded
I-04	Frontrunning can pre-create stream ATA and DoS create_stream and create_nft_bound_stream	Informational	● Not recorded
I-05	Misspelled Helper Function receover_signer Hurts Clarity	Informational	● Not recorded

Detailed findings

M-01 Attacker Can Block Token Distribution Recipients

Severity **Medium** Status ● Not recorded

DESCRIPTION

The LockHook.execute flow initializes several on-chain accounts as part of processing a claim, including creating an NFT mint via a direct SystemProgram.create_account call and invoking the clique-lock program to create deterministic PDAs such as streams and NFT bindings. These accounts are expected to be uninitialized at the time of execution. The hook unconditionally calls create_account for the mint and relies on downstream CPI calls to initialize PDAs, assuming the destination accounts hold zero lamports.

However, the System Program rejects create_account when the destination already has lamports, returning AccountAlreadyInUse. Because the mint address or the PDAs derived for clique-lock can be predictable, an attacker can pre-fund these addresses and cause the initialization path to revert.

IMPACT

An attacker can persistently deny service to legitimate claimants by preventing NFT mint creation or stream initialization, blocking token distribution without directly stealing funds. Affected claims cannot be completed until manual cleanup or code changes are applied, potentially halting distributions at scale and causing operational disruption, user complaints, and reputational damage.

RECOMMENDED MITIGATION

Handle pre-funded destination accounts safely by avoiding unconditional create_account calls and supporting an allocate-and-assign fallback when lamports are already present.

M-02 Accepting Mints With Freeze Authority Enables Post-Claim Freezes

Severity **Medium** Status ● Not recorded

DESCRIPTION

If the distributor uses a mint with a configured freeze authority, that authority can freeze recipients token accounts (including ATAs) after claims, creating a denial-of-service and custody risk outside the programs control. The program does not restrict or validate the mints freeze_authority at initialization, and set_mint accepts a raw Pubkey without on-chain validation.

```
// new_distributor.rs
pub mint: InterfaceAccount<'info, Mint>,
distributor.mint = ctx.accounts.mint.key();

pub fn set_mint<'info>(ctx: Context<SetMint>, mint: Pubkey) -> Result<()> {
    ctx.accounts.distributor.mint = mint;
    emit!(MintUpdatedEvent {
        distributor: ctx.accounts.distributor.key(),
        mint,
    });
    Ok(())
}
```

IMPACT

Recipients ATAs can be frozen post-claim, enabling external censorship/DoS by the freeze authority.

RECOMMENDED MITIGATION

Enforce no freeze authority at initialization and require validation when updating the mint. Accept the full Mint account in set_mint and verify freeze_authority.is_none() before assignment. Document that distributions must use mints without a freeze authority; if retention is necessary, make the distributor PDA the freeze authority and clearly disclose the risks.

```
// new_distributor.rs
require!(
    ctx.accounts.mint.freeze_authority.is_none(),
    ErrorCode::MintHasFreezeAuthority
);

pub fn set_mint<'info>(ctx: Context<SetMint>, mint: Pubkey) -> Result<()> {
    require!(
        ctx.accounts.mint.freeze_authority.is_none(),
        ErrorCode::MintHasFreezeAuthority
    );
    ctx.accounts.distributor.mint = mint;
    emit!(MintUpdatedEvent {
        distributor: ctx.accounts.distributor.key(),
        mint,
    });
    Ok(())
}
```

M-03 Operator Can Break Claim Flow Recipient

Severity **Medium** Status ● Not recorded

DESCRIPTION

The lock hook creates the recipients NFT associated token account by building a `create_associated_token_account_idempotent` instruction and invoking it directly. This instruction requires the associated token program account to be present in the CPI account list. While the code correctly validates and stores `associated_token_program` in `LockHookAccounts`, the subsequent `invoke` call for the ATA creation omits this account and only includes the `Token-2022` program.

As a result, the CPI does not satisfy the instructions expected accounts, causing the runtime to reject the invocation. This issue is purely functional and will consistently trigger when the ATA does not already exist.

IMPACT

All claim executions that require creating the recipients NFT associated token account will revert at runtime, permanently breaking the claim and lock flow for affected users. This prevents NFT minting and downstream stream creation, halting token distribution and requiring a code fix and redeploy to restore functionality.

RECOMMENDED MITIGATION

Include the associated token program account in the CPI account slice for the ATA creation, or use Anchors associated token helpers to ensure required accounts are always provided.

```
anchor_lang::solana_program::program::invoke(
    &ix_create_ata,
    &[
        ctx.payer.clone(),
        accounts.nft_token_account.to_account_info(),
        ctx.claimer.clone(),
        accounts.nft_mint.to_account_info(),
        accounts.system_program.to_account_info(),
        accounts.token_2022_program.to_account_info(),
        accounts.associated_token_program.to_account_info(),
    ],
    +
)?;
```

L-01 Revocation Forfeits Vested but Unclaimed Tokens to Revoke Authority

Severity Low Status ● Not recorded

DESCRIPTION

The stream revocation logic returns all tokens that have not yet been claimed back to the revoke authority, even if a portion of those tokens are already vested and claimable by the recipient. In the revoke instruction, the remaining amount is calculated as `stream.amount - stream.claimed_amount`, where `claimed_amount` only tracks the amount that has been actually withdrawn through the claim instruction. This means that vested tokens which have accrued over time but have not been actively claimed by the recipient are still treated as remaining and will be transferred back to the revoke authority upon revocation. The revocation process does not compute the current vested entitlement using `calculate_claimable_amount`, nor does it automatically pay out any unclaimed vested portion to the recipient before reclaiming the rest.

IMPACT

Recipients may lose tokens that have already vested if they do not claim them immediately. A revoke authority can cancel the stream at any time and reclaim all tokens except those already withdrawn, effectively confiscating vested but unclaimed amounts. This creates an economic fairness and trust risk where users must claim frequently to avoid losing earned vesting, and revocation can be used to deprive recipients of funds they would reasonably expect to have already unlocked.

MITIGATION

During revocation, compute the total vested amount at the current timestamp using `calculate_claimable_amount` and ensure any vested-but-unclaimed portion is first transferred to the recipient. Only the unvested remainder should be returned to the revoke authority. Alternatively, clearly document the current behavior as an explicit design choice so recipients understand that only claimed tokens are guaranteed and that vested tokens can be forfeited upon revocation.

L-02 Dynamic Recipient Toggle Bricks Pending Claims

Severity Low Status ● Not recorded

DESCRIPTION

Flipping `distributor.dynamic_recipient` from true to false after publishing a dynamic-root breaks all pending dynamic claims. The handler derivation during claim is keyed off the mutable `dynamic_recipient` flag rather than data committed to the Merkle root, so toggling the flag forces recomputation as `keccak(claimer_pubkey)` and invalidates proofs that were valid at publication.

SETTER

```
pub fn set_is_dynamic_recipient(
    ctx: Context<SetIsDynamicRecipient>,
    is_dynamic_recipient: bool,
) -> Result<()> {
    ctx.accounts.distributor.dynamic_recipient = is_dynamic_recipient;
    emit!(IsDynamicRecipientUpdatedEvent { /* ... */ });
    Ok(())
}
```

IMPACT

A single toggle bricks all dynamic claims, halting distributions and trapping user funds until a fresh root (or distributor) is deployed.

RECOMMENDED MITIGATION

Bind the handler mode to the published root instead of global mutable state. Store `dynamic_recipient` alongside each root on creation and require claim to use the flag committed in the root metadata. Reject toggling while a dynamic-configured root remains active, or enforce a phased rollout (e.g., clearing outstanding dynamic roots before switching to static). Add integration tests covering dynamic-to-static transitions to prevent regressions.

```
pub struct RootConfig {
    pub merkle_root: [u8; 32],
    pub dynamic_recipient: bool,
}
```

L-03 Vault Parameter Should Use SystemAccount Type

Severity Low Status ● Not recorded

DESCRIPTION

In `set_vault`, the vault parameter is provided as a raw Pubkey without on-chain type/owner validation. This allows setting an uninitialized or non-system account as the vault, which is a configuration hazard and can result in zero-initialized or incorrect destinations being recorded.

IMPACT

Primarily an operability/configuration risk rather than direct fund loss. An invalid or zero-initialized vault can cause later transfers, accounting, or withdrawals to fail or behave unexpectedly (DoS-like until corrected).

RECOMMENDED MITIGATION

Require a `SystemAccount<'info>` for the new vault and assign the distributors vault from that accounts key. This ensures the account exists and is owned by the system program. Optionally add rent-exemption checks if the vault must maintain minimum balance guarantees.

```
#[derive(Accounts)]
pub struct SetVault<'info> {
    #[account(mut, seeds = [MerkleDistributor::SEED, base.key().to_bytes().as_ref()], bump)]
    pub distributor: Account<'info, MerkleDistributor>,
    pub base: Signer<'info>,
    pub new_vault: SystemAccount<'info>, // Ensure it's a valid system account
    pub system_program: Program<'info, System>,
}

pub fn set_vault(ctx: Context<SetVault>) -> Result<()> {
    ctx.accounts.distributor.vault = ctx.accounts.new_vault.key();
    Ok(())
}
```

L-04 Base Account Constraint Prevents Protocol Integration

Severity **Low** Status ● Not recorded

DESCRIPTION

The base account in the Claim instruction is constrained to SystemAccount, which restricts it to EOAs owned by the System Program. This prevents using PDAs, multisigs, or governance-controlled accounts as the distributor controller and blocks integrations where another protocol or DAO should own and operate the distributor. The account is also marked mut and is the recipient of system transfers, further limiting composability.

IMPACT

Reduces composability and decentralization by preventing PDA- or governance-controlled distributors, limiting integrations with protocols/DAOs and forcing EOA-controlled deployments.

RECOMMENDED MITIGATION

Allow any account type for base by switching to `AccountInfo<'info>` (or a more flexible account type). If specific authority constraints are needed (e.g., PDA seeds or multisig checks), enforce them explicitly rather than via SystemAccount.

```
/// CHECK: Base account that controls the distributor (PDA, multisig, or EOA)
#[account(mut)]
pub base: AccountInfo<'info>,
```

L-05 Insufficient Timestamp Validation Allows Vesting Bypass via Instant or Pre-Vested Streams

Severity **Low** Status ● Not recorded

DESCRIPTION

The stream creation logic enforces only weak relative ordering (`start_time <= cliff_time <= end_time`) for vesting timestamps. This validation is insufficient and allows multiple misconfigurations that effectively bypass the vesting mechanism:

1. Instant Unlock Streams: All vesting timestamps (start_time, cliff_time, and end_time) can be set to the same value. When this occurs, the vesting calculation in `calculate_claimable_amount` immediately treats the stream as fully vested, resulting in instant token unlock instead of gradual vesting.
2. Pre-Vested Streams: There is no validation ensuring that vesting timestamps are in the future or at least greater than the current block timestamp. As a result, creators can set all timestamps to a time in the past, causing the stream to be considered fully vested at creation.

In both cases, the vesting schedule becomes ineffective due to missing temporal constraints.

IMPACT

- Vesting schedules can be unintentionally or deliberately bypassed
- Tokens may become instantly claimable despite expectations of time-based vesting
- Protocol behavior deviates from intended vesting guarantees

MITIGATION

- Enforce strict timestamp ordering (`start_time < cliff_time < end_time`)
- Validate that end_time (and ideally start_time) is greater than the current block timestamp

L-06 Missing Chain Identifier Enables Cross-Chain Signature Replay Attacks

Severity **Low** Status ● Not recorded

DESCRIPTION

The claim signature lacks a mandatory domain separator (chain identifier). The signed message includes the distributor PDA but omits any chain-unique identifier (e.g., program ID or cluster/chain ID). Since the distributor PDA is deterministically derived from `[MerkleDistributor::SEED, base.key()]`, the same PDA can exist across deployments on different SVM chains when using the same base key. A signature valid on one chain (e.g., devnet) can be replayed on another (e.g., mainnet) if roots and signer are reused. The extra parameter is optional and unconstrained, making it unreliable as a domain separator without explicit on-chain enforcement.

```
// claim.rs excerpt
let msg = [
    handler.as_ref(),
    claimer.key().to_bytes().as_ref(),
    amount.to_le_bytes().as_ref(),
    released.to_le_bytes().as_ref(),
    root.as_ref(),
    ctx.accounts.distributor.key().to_bytes().as_ref(),
    extra.as_ref(),
].concat();
```

IMPACT

Enables cross-chain signature replay, allowing double-claims across deployments before operators notice, potentially draining vaults and undermining the distributions economic model.

RECOMMENDED MITIGATION

Add explicit domain separation by binding signatures to a specific deployment. Include the program ID and a chain/cluster identifier in the signed message, or enforce a structured extra that contains and validates a chain ID committed on-chain (e.g., stored in distributor state). Prefer a non-optional, validated field for reliability.

```
// Example: include program ID and chain identifier in the signed message
let chain_id: u64 = /* retrieve from distributor state or config constant */;
let msg = [
    handler.as_ref(),
    claimer.key().to_bytes().as_ref(),
    amount.to_le_bytes().as_ref(),
    released.to_le_bytes().as_ref(),
    root.as_ref(),
    ctx.accounts.distributor.key().to_bytes().as_ref(),
    &crate::ID.to_bytes(), // bind to this program deployment
    &chain_id.to_le_bytes(), // bind to this chain/cluster
    extra.as_ref(),
].concat();

// Alternative: enforce structured `extra` carrying chain_id (first 8 bytes)
require!(extra.len() >= 8, ErrorCode::InvalidExtra);
let provided_chain_id = u64::from_le_bytes(extra[0..8].try_into().unwrap());
let expected_chain_id: u64 = /* retrieve from state or constant */;
require!(provided_chain_id == expected_chain_id, ErrorCode::InvalidChainId);
```

L-07 Caller Can Redirect NFT Mint To Incorrect Token Account

Severity **Low** Status ● Not recorded

DESCRIPTION

The `LockHookAccounts::try_from_remaining` parser accepts `nft_token_account` from `remaining_accounts` and only validates that it is writable. The intended behavior is for this account to be the recipients associated token account (ATA) for the (claimer, `nft_mint`) pair, which is later used in CPIs for ATA creation and NFT minting.

However, the code does not verify that the provided `nft_token_account` matches the deterministic associated token address derived from the claimer and the NFT mint.

This allows a caller to supply an arbitrary writable token account, deviating from the expected ATA invariant and undermining assumptions made by the minting and account-creation logic.

IMPACT

If a mis-specified or malicious `nft_token_account` is provided, CPIs may mint the NFT into an unintended token account or fail during ATA creation, causing confusion, griefing, or inconsistent state. While direct theft is unlikely under current constraints, this weakens state integrity guarantees and can break the claim flow or require manual investigation and cleanup.

RECOMMENDED MITIGATION

Enforce that `nft_token_account` equals the canonical associated token address for the claimer and the NFT mint by deriving the expected ATA and comparing it against the provided account.

```
+ let expected_ata = get_associated_token_address(  
+   &ctx.recipient,  
+   &accounts.nft_mint.key(),  
+ );  
+ require_keys_eq!(  
+   accounts.nft_token_account.key(),  
+   expected_ata,  
+   ErrorCode::InvalidTokenAccount  
+ );
```

I-01 Mutable Mint Allows Token Swap After Root Publication

Severity Informational

Status ● Not recorded

DESCRIPTION

Setter functions (e.g., `set_mint`, `set_vault`, `set_signer`) do not verify that the provided value differs from the current on-chain state, allowing no-op updates that consume compute and fees while producing redundant events.

IMPACT

No security impact, but wastes compute/fees, increases write contention, and can clutter event logs with redundant updates.

RECOMMENDED MITIGATION

Add guards to reject or early-return on no-op updates. For example:

```
pub fn set_mint(ctx: Context<SetMint>, mint: Pubkey) -> Result<()> {
    require!(ctx.accounts.distributor.mint != mint, ErrorCode::NoStateChange);
    ctx.accounts.distributor.mint = mint;
    // ... emit event
    Ok(())
}
```

Apply analogous checks in `set_vault` and `set_signer` against their current values.

I-02 No Fee Caps Allow Unbounded Fee Extraction

Severity Informational

Status ● Not recorded

DESCRIPTION

The `set_fixed_fee` and `set_fee_per_tier` functions do not enforce maximum fee limits, allowing the base controller to set arbitrarily high fees that could exceed claim amounts.

```
// set_fixed_fee.rs - no cap on fee parameter
pub fn set_fixed_fee(ctx: Context<SetFixedFee>, configurator: [u8; 32], fee: u64)

// set_fee_per_tier.rs - no cap on rate, max_fee, or min_fee
pub fn set_fee_per_tier(
    ctx: Context<SetFeePerTier>,
    configurator: [u8; 32],
    rate: u64,
    max_fee: u64,
    min_fee: u64,
)
```

IMPACT

This allows fees that exceed claim amounts, preventing users from claiming their allocations.

RECOMMENDED MITIGATION

Add fee validation constraints:

```
const MAX_FEE_LAMPORTS: u64 = 1_000_000_000; // 1 SOL max
const MAX_FEE_RATE: u64 = 100_000_000; // 10% max (out of 1B basis)

require!(fee <= MAX_FEE_LAMPORTS, ErrorCode::FeeTooHigh);
require!(rate <= MAX_FEE_RATE, ErrorCode::RateTooHigh);
require!(max_fee <= MAX_FEE_LAMPORTS, ErrorCode::FeeTooHigh);
require!(min_fee <= max_fee, ErrorCode::InvalidFeeRange);
```

I-03 set_mint Lacks Mint Account Validation and Legacy Token Enforcement

Severity Informational Status ● Not recorded

DESCRIPTION

The program uses legacy SPL Token (v1) throughout (e.g., Token, TokenAccount, CPIs via `anchor_spl::token`) and validates a legacy Mint at initialization, but `set_mint` function accepts a raw Pubkey without owner/layout checks. This allows setting a Token-2022 mint or even a non-mint, which will later mismatch legacy Token CPIs and ATAs during claim, causing failures or undefined behavior.

IMPACT

Infocore protocol logic but operationally disruptive: misconfiguration can DoS claimants (claims fail), strand distributions, and force redeployments/rollbacks. This is primarily a configuration risk rather than direct fund loss.

RECOMMENDED MITIGATION

Change `set_mint` to accept and validate a full Mint account alongside the legacy Token program, enforcing that the mints owner equals the provided legacy Token program and thereby ensuring a valid legacy mint. Keep initialization strict by continuing to accept `Account<'info, Mint>` and applying the same owner constraint there. This binds all ATAs and CPIs to Token v1, prevents Token-2022/non-mint keys from being set, and aligns on-chain checks with Anchors owner/layout guarantees.

I-04 Frontrunning can pre-create stream ATA and DoS create_stream and create_nft_bound_stream

Severity Informational Status ● Not recorded

DESCRIPTION

`create_stream` and `create_nft_bound_stream` initializes the stream ATA with `#[account(init, associated_token:...)]`. ATAs can be created by any payer for any owner (including PDAs) as long as the address is known. A frontrunner who learns the `stream_id` (or computes the PDA) can precreate the ATA, causing the victims transaction to fail because `init` requires the account to be uninitialized.

```
#[account(
  >> init,
    payer = payer,
    associated_token::mint = mint,
    associated_token::authority = stream,
    associated_token::token_program = token_program,
  )]
pub stream_token_account: InterfaceAccount<'info, TokenAccount>,
```

IMPACT

Stream creation can be denied (transaction failure). No loss of funds, but the user must retry with a new stream_id. This is a griefing/DoS vector.

RECOMMENDED MITIGATION

Use init_if_needed for the ATA and verify it matches the expected mint and owner (stream PDA). This allows safe reuse of a preexisting ATA while preserving correctness.

I-05 Misspelled Helper Function receover_signer Hurts Clarity

Severity **Informational** Status ● Not recorded

DESCRIPTION

The helper function that recovers the signer from a secp256k1 signature is misspelled as receover_signer instead of recover_signer. While functionally harmless, this reduces readability and risks confusion or propagation of the typo in future refactors and integrations.

IMPACT

No runtime/security impact, but degrades code clarity and increases the chance of human error during maintenance or external referencing.

RECOMMENDED MITIGATION

Rename the function and all call sites from receover_signer to recover_signer. Keep the implementation unchanged.

```
// before
pub fn receover_signer(sig: [u8; 65], hash: [u8; 32]) -> Result<[u8; 64]> {
  let recovery_id = {
    let v = sig[64];
    if v > 27 { v - 27 } else { v }
  };
  let signature = &sig[..64];
  let recovered = secp256k1_recover(&hash, recovery_id, signature)
    .map_err(|_| ErrorCode::InvalidSignature)?;
  Ok(recovered.to_bytes())
}

// after
pub fn recover_signer(sig: [u8; 65], hash: [u8; 32]) -> Result<[u8; 64]> {
```

```
    let recovery_id = {
        let v = sig[64];
        if v > 27 { v - 27 } else { v }
    };
    let signature = &sig[..64];
    let recovered = secp256k1_recover(&hash, recovery_id, signature)
        .map_err(|_| ErrorCode::InvalidSignature)?;
    Ok(recovered.to_bytes())
}

// call site change in claim
// before
let signer = receover_signer(sig, hash(&msg).to_bytes());
// after
let signer = recover_signer(sig, hash(&msg).to_bytes());
```

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