Security Assessment Report

Relay Escrow

June-2025

Prepared for **Relay Protocol**

Table of Contents

Project Summary

Project Scope

Project Name	Repository (link)	Initial Commit Hash	Platform
Relay Escrow	Github Repository	91b182a	Solana

Project Overview

This document describes the verification of **Relay Escrow** code using manual code review. The work was undertaken from **June 6** to **June 13, 2025**.

The following contracts are considered in scope for this review:

- `escrow-contracts/packages/solana-vm/programs/relay-escrow/src/lib.rs`
- `escrow-contracts/packages/solana-vm/programs/relay-forwarder/src/lib.rs`

During the audit, Certora discovered bugs in the Solana programs, as listed on the following page.

Protocol Overview

The Relay Escrow is a system that enables secure fund transfers and swaps using off-chain authorization. Users interact with the forwarder program responsible for transferring assets from the user account to the escrow program which, in turn, stores them in a dedicated vault account. The funds remain safe in the vault until an authorized party signs a message that allows the trade to be finalized and the promised assets to be sent to the legitimate recipient. This design adds a layer of security where outgoing financial flows are governed by the program logic and a trusted signer.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	2	2	2
High	1	1	1
Medium	2	2	2
Low	2	2	2
Informational	2	2	2
Total	9	9	9

Severity Matrix

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

Detailed Findings

ID	Title	Severity	Status
C-01	Anyone can initialize the relay escrow and become the owner/allocator	Critical	Fixed
C-02	Transfer recipient can be changed to arbitrary address, allowing complete escrow drainage	Critical	Fixed
H-01	Signer account is mistakenly treated as the forwarder account, leading to user drainage and broken logic	High	Fixed
M-01	Contract fails to support Token2022, limiting compatibility with modern SPL tokens	Medium	Fixed
M-02	Depositors can specify arbitrary vault token account, enabling fake deposits that bypass actual escrow	Medium	Fixed
L-01	Token account rent is susceptible to be stolen	Low	Fixed
L-02	Vault account automatic closure prevents token transfer operations	Low	Fixed

Critical Severity Issues

C-01 Anyone can initialize the relay escrow and become owner/allocator

Severity: **Critical**	Impact: **High**	Likelihood: **High**
Files: relay-escrow/src/lib.rs #L32-L38	Status: Fixed	

Description: The `initialize` function lacks access control and uses a static PDA seed, allowing any user to call the initialization function and gain complete control over the relay escrow system. Since there is only one global escrow PDA derived from `seeds = [b"relay_escrow"]`, the first caller becomes the permanent owner and allocator with full control over all future transfers.
An attacker can set himself as the `allocator` which allows him to later sign `TransferRequests` and specify a recipient for the tokens so he can steal them.

Recommendations: Restrict initialization to a specific authority and make sure it matches the signer/owner of the `initialize` function

Customer's response: Fixed in commit 3a0579b

Fix Review: Fixed.

C-02 Transfer recipient can be changed to arbitrary address, allowing complete escrow drainage		
Severity: **Critical**	Impact: **High**	Likelihood: **High**
Files: relay-escrow/src/lib.rs #L163	Status: Fixed	

Description: The `execute_transfer` function validates that a `TransferRequest` was properly signed by the authorized allocator, but fails to verify that the actual recipient account passed in the instruction matches the recipient specified in the signed request. This creates a critical vulnerability where an attacker can intercept valid signatures and redirect funds to their own address.

Exploit Scenario:

1. Allocator signs a legitimate `TransferRequest` with `recipient: Alice`
2. Attacker intercepts the signature and transaction details
3. Attacker calls `execute_transfer` with the valid `request` and signature, but passes `recipient: Attacker` in the instruction accounts
4. Signature validation passes (correct allocator, correct request hash)
5. Funds are transferred to attacker instead of intended recipient

Recommendations: Add explicit validation to ensure the recipient account matches the signed request

Customer's response: Fixed in commit cba84c6

Fix Review: Fixed.

High Severity Issues

H-01 Signer account is mistakenly treated as forwarder account, leading to user drainage and broken logic		
Severity: **High**	Impact: **High**	Likelihood: **High**
Files: relay-forwarder/src/lib.rs#L16-L26	Status: Fixed	

Description: The relay-forwarder program acts as an intermediary between users and the relay-escrow program, handling both native SOL and SPL token transfers. It uses a two-program architecture where the forwarder makes CPIs to the escrow program to perform the funds transfer.

However, the design incorrectly treats the **user's** wallet (Signer) as the forwarder account (the program itself) which breaks the entire program logic.

```rust
#[derive(Accounts)]
#[instruction(
   id: [u8; 32],
)]
pub struct ForwardNative<'info> {
   // The forwarder account that holds and will send the tokens
   #[account(mut)]
   pub forwarder: Signer<'info>, // <--------- HERE

   /// CHECK: Used as public key only
   pub depositor: UncheckedAccount<'info>,

   /// CHECK: Relay escrow program account
   pub relay_escrow: UncheckedAccount<'info>,

   /// CHECK: Relay escrow vault
```

```
    #[account(mut)]
    pub relay_vault: UncheckedAccount<'info>,

    pub relay_escrow_program: Program<'info, relay_escrow::program::RelayEscrow>,
    pub system_program: Program<'info, System>,
}
```

For example, when `forward_native()` is called, it mistakenly takes the **user's** entire SOL balance (instead of the forwarder program itself) via `ctx.accounts.forwarder.lamports()` and sends them to the escrow via CPI.

```rust
pub fn forward_native(ctx: Context<ForwardNative>, id: [u8; 32]) -> Result<()> {
    let amount = ctx.accounts.forwarder.lamports(); // <------------ 1
    require!(amount > 0, ForwarderError::InsufficientBalance);

    relay_escrow::cpi::deposit_native(
        CpiContext::new(
            ctx.accounts.relay_escrow_program.to_account_info(),
            ctx.accounts.into_deposit_accounts(),
        ),
        amount,// <------------ 2
        id,
    )?;

    Ok(())
}
```

As a result, the user's wallet will be completely drained by the program after the `relay_escrow::deposit_native()` CPI.

On Solana, when a program makes a CPI, the signer privileges from the initial transaction [extend to the callee program](). This means that if an account is passed as a signer to the calling program, it remains a signer for the callee program as well. This mechanism allows programs to forward signer accounts, effectively extending the authority of the original transaction to the called program.

Exploit Scenario:

1. Bob has a token account holding 100 USDC
2. Bob wants to bridge 50 USDT so he first swaps the USDT to USDC and then forwards that amount to the escrow
3. The 50 USDT were swapped and now in Bob's token account there are 150 USDC (100 from before and 50 after the swap happened)
4. When `forward_token` happens it will forward all of Bob's USDC - 150 tokens instead of the 50 USDC Bob wanted to bridge, so now Bob's left with 0 USDC

Since the program is entirely built upon this oversight, other parts of the code are also affected by the bug with more or less critical impacts.

Recommendations: Introduce a new `vault` account that should be a PDA account that has authority over a `vault_token_account.`
The following must be taken into account:

- The `vault_token_account` must exist for each new mint being forwarded and the newly introduced `vault` PDA must be its authority.
- The newly introduced `vault` PDA must sign the CPI responsible for forwarding the tokens to the escrow.

Customer's response: Fixed in commit 333c266

Fix Review: Fixed, but the refund from closing the account goes to the forwarder instead of the sender.
Fixed in commit f92c6d7

Medium Severity Issues

M-01 Contract fails to support Token2022 tokens, limiting compatibility with modern SPL tokens		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: [relay-escrow/src/lib.rs #L325]()	Status: Fixed	

Description: The `deposit_token` function and related token operations are hardcoded to use the legacy SPL Token program, preventing the contract from supporting Token2022 tokens. Token2022 is the new standard for SPL tokens on Solana and includes important features like transfer fees, confidential transfers, and other extensions that many new tokens require.

Recommendations: Replace the hardcoded Token program with `TokenInterface` to support both legacy and Token2022 tokens

Customer's response: Fixed in commit 4d95204

Fix Review: Fixed. Latest commit - 6a4e19c adds support for fee-on-transfer tokens

M-02 Depositors can specify arbitrary vault token account, enabling fake deposits that bypass actual escrow

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: relay-escrow/src/lib.rs #L316	Status: Fixed	

Description: The `deposit_token` function accepts `vault_token_account` as an `UncheckedAccount`, allowing depositors to specify any token account address, including their own. This enables attackers to perform "fake deposits" where they transfer tokens to themselves while the program emits a legitimate `DepositEvent`, potentially deceiving off-chain systems and creating accounting discrepancies.

Exploit Scenario:

1. Attacker calls `deposit_token` with legitimate parameters
2. Attacker specifies `vault_token_account` as their own token account (not the escrow's)
3. Tokens are transferred from attacker's sender account to attacker's "vault" account (net zero effect)
4. Program emits `DepositEvent` indicating a legitimate deposit occurred
5. Off-chain systems detect the event and credit the attacker's balance

Recommendations: Ensure the vault token account is the correct Associated Token Account for the vault.

Customer's response: Fixed in commit 4f5d142

Fix Review: Fixed.

Low Severity Issues

L-01 Token account rent is susceptible to be stolen		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: relay-forwarder/src/lib.rs#L52-L62	Status: Fixed	

Description: When users interact with `forward_token()`, they have the ability to close the token account and receive its rent once the account balance has been forwarded to the escrow by setting the `should_close` parameter to `true`. This design is relevant in case the token account does not exist yet and needs to be created. In this case, the user will pay a temporary rent just to set up the token account, perform the forwarding and then close it to claw back the rent.
In case the user that created the account decides **to not close** it after the forwarding, another user has the ability to do it for himself and obtain the rent.

Recommendations: The easiest way to mitigate this edge case is to close the account unconditionally after the forwarding has been completed.

Customer's response: Fixed in commit 341552e

Fix Review: Fixed.

L-02 Vault account automatic closure prevents token transfer operations		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: relay-escrow/lib.rs	Status: Fixed	

Description: The `execute_transfer` function contains a vulnerability related to the vault account's lifecycle management. The vault account, which serves as a Program Derived Account (PDA) for signing transactions, is never explicitly initialized and therefore retains the System program and is initialized whenever someone sends funds to that account.

When the `execute_transfer` function processes native SOL transfers, it uses `invoke_signed` with `system_instruction::transfer` to transfer the requested amount from the vault account to the recipient. If this transfer consumes all available lamports in the vault account (reducing the balance to 0), Solana's runtime automatically closes the account since it's no longer rent exempt.

Once the vault account is closed, it can no longer be used as a signing authority for subsequent operations. This creates a failure scenario where:

1. The vault account gets depleted of lamports during a native SOL transfer
2. Solana automatically closes the zero-balance System-owned account
3. Any remaining tokens in the `vault_token_account` are effectively locked until the vault gets a balance != 0

Recommendations: Ensure the vault account has some lamports left to cover the rent

Customer's response: Fixed in commit c2da3f3 and 66f3302

Fix Review: Fixed.

Informational Issues

I-01. Hard-coded account space calculation in relayEscrow initialization

Description: The accounts initialization uses a manually calculated space allocation instead of Anchor's built-in INIT_SPACE constant. The current implementation hard-codes the space calculation as:

```rust
space = 8 + 32 + 32 + 1, // discriminator(8) + owner(32) +
allocator(32) + vault_bump(1)
```

This approach is not considered best practice and can lead to confusion and mistakes when determining the exact space we want for a specific account.

Recommendation: Replace the manual space calculation with Anchor's INIT_SPACE constant to ensure automatic synchronization between struct definition and space allocation:
rust

```rust
#[account(
    init,
    payer = owner,
    space = 8 + RelayEscrow::INIT_SPACE, // Automatically calculated
based on struct
    seeds = [b"relay_escrow"],
    bump
)]
pub relay_escrow: Account<'info, RelayEscrow>,
```

Customer’s response: Fixed in commit 7c2b546

Fix Review: Fixed.

I-02. Missing PDA constraints for RelayEscrow account validation

Description: The `SetAllocator` and `ExecuteTransfer` account structs do not enforce proper Program Derived Address (PDA) constraints on the `relay_escrow` account, unlike in the other instructions. While this may not be directly exploitable due to Anchor's type system ensuring the account deserializes to `RelayEscrow` and is owned by the program, it is best practice to include the seeds and the bump to that account.

Recommendation: Set the corresponding PDA constraints for the `relay_escrow` account in the `SetAllocator` and `ExecuteTransfer` account structs

Customer's response: Fixed in commit 7daefc6

Fix Review: Fixed.

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.