Pump Security Review

Pashov Audit Group

Conducted by: shaflow, 0xeix, JoVi

June 26th 2025 - June 27th 2025

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **pump-fun/pump_transfer_hook** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Pump Transfer Hook

This Pump program implements a token transfer hook with a whitelist system, allowing a super admin to initialize transfer metadata, add specific wallets to a transfer whitelist, and disable the whitelist checks permanently. It enforces that only whitelisted accounts can perform transfers while the whitelist is active, enabling a controlled token distribution mechanism.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. 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.

5.2. 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.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hashes:

- 042e8febfd07ac92f70dd2e2fa0fa378d2532c8c

fixes review commit hashes:

- 620ff3eff8ce97d1bc958a07cb889580b39a34e5

Scope

The following smart contracts were in scope of the audit:

- `lib`

7. Executive Summary

Over the course of the security review, shaflow, 0xeix, JoVi engaged with Pump to review Pump Transfer Hook. In this period of time a total of **7** issues were uncovered.

Protocol Summary

Protocol Name	Pump Transfer Hook
Repository	https://github.com/pump-fun/pump_transfer_hook
Date	June 26th 2025 - June 27th 2025
Protocol Type	Whitelist management

Findings Count

Severity	**Amount**
Medium	1
Low	6
Total Findings	**7**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Any wallet can self-assign as super_admin for arbitrary mint	Medium	Resolved
[L-01]	Token programs required in struct yet not utilized	Low	Resolved
[L-02]	Lacks a mechanism for modifying/removing whitelist entries	Low	Resolved
[L-03]	add_whitelist_ix does not check for duplicate addresses	Low	Resolved
[L-04]	Pre-hook transfer window allows unrestricted token movement	Low	Acknowledged
[L-05]	TransferHookState account is over-allocated by 8 bytes	Low	Resolved
[L-06]	Bypass token-account address whitelisting with SetAuthority	Low	Acknowledged

8. Findings

8.1. Medium Findings

[M-01] Any wallet can self-assign as super_admin for arbitrary mint

Severity

Impact: Medium

Likelihood: Medium

Description

An arbitrary signer is accepted as super_admin at initialize_extra_account_metas and does not verify any relationship between that signer and the target mint:

```
#[account(init, seeds = [TRANSFER_HOOK_STATE_SEED.as_bytes(), mint.key().as_ref
  ()], ...)]
pub transfer_hook_state: Account<'info, TransferHookState>,
```

There is no constraint on mint.mint_authority or mint.freeze_authority. No additional account (e.g., extension authority) is required to sign. Once the PDA at [TRANSFER_HOOK_STATE_SEED, mint] is initialized, we cannot initialize it again.

Due to the mentioned points, anyone can initialize the PDA before the real token team, disable the whitelist, and walk away with no risk or major cost.

Because toggle_off_transfer_whitelist is one-way, there is no recovery path.

Recommendations

Gate initialization behind mint authority control:

```
pub struct InitializeExtraAccountMeta<'info> {
    #[account(constraint = mint.mint_authority == COption::Some(super_admin.key
      ()).into())]
    pub mint: InterfaceAccount<'info, Mint>,
    #[account(mut, signer)]
    pub super_admin: Signer<'info>,
    // ...
}
```

8.2. Low Findings

[L-01] Token programs required in struct yet not utilized

Currently, the `InitializeExtraAccountMeta` struct contains the following accounts:

```
pub struct InitializeExtraAccountMeta<'info> {
    #[account(mut)]
    pub super_admin: Signer<'info>,
    /// CHECK: ExtraAccountMetaList Account, must use these exact seeds
    #[account(mut, seeds=[b"extra-account-metas", mint.key().as_ref()], bump)]
    pub extra_account_meta_list: AccountInfo<'info>,
    pub mint: InterfaceAccount<'info, Mint>,
    #[account(init, seeds=[TRANSFER_HOOK_STATE_SEED.as_bytes(), mint.key
      ().as_ref()], bump, payer=super_admin, space=8+ TransferHookState::SIZE)]
    pub transfer_hook_state: Account<'info, TransferHookState>,
    pub token_program: Interface<'info, TokenInterface>,
    pub associated_token_program: Program<'info, AssociatedToken>,
    pub system_program: Program<'info, System>,
}
```

The problem is that the only program used in the `initialize_extra_account_metas_ix()` instruction is the `system_program` and therefore `associated_token_program` and `token_program` are not needed. There is no serious impact under that as the runtime does nothing with them but it affects the overall readability.

[L-02] Lacks a mechanism for modifying/removing whitelist entries

Currently, only the admin can add addresses to the `transfer_hook_state` whitelist, but there is no way to remove or modify addresses once they have been added. If a whitelisted address becomes malicious, there is no mechanism to prevent it from performing token transfers.

```
pub fn add_whitelist_ix(
    ctx: Context<UpdateTransferWhitelist>,
    whitelist_address: Pubkey,
) -> Result<()> {
    let transfer_hook_state = &mut ctx.accounts.transfer_hook_state;

    if let Some(pos) = transfer_hook_state
        .whitelisted_wallets
        .iter()
        .position(|key| *key == Pubkey::default())
    {
        transfer_hook_state.whitelisted_wallets[pos] = whitelist_address;
    } else {
        return err!(TransferError::WhitelistFull);
    }

    Ok(())
}
```

It is recommended to add a mechanism for removing or modifying whitelist entries.

[L-03] `add_whitelist_ix` does not check for duplicate addresses

The `add_whitelist_ix` instruction is used to add a new address to the token whitelist.

```
pub fn add_whitelist_ix(
    ctx: Context<UpdateTransferWhitelist>,
    whitelist_address: Pubkey,
) -> Result<()> {
    let transfer_hook_state = &mut ctx.accounts.transfer_hook_state;

    if let Some(pos) = transfer_hook_state
        .whitelisted_wallets
        .iter()
        .position(|key| *key == Pubkey::default())
    {
        transfer_hook_state.whitelisted_wallets[pos] = whitelist_address;
    } else {
        return err!(TransferError::WhitelistFull);
    }

    Ok(())
}
```

However, it does not check whether an address has already been added. As a result, duplicate addresses can be inserted into the whitelist. Additionally, there is no mechanism to remove or replace an existing address, and the whitelist has a limited size. If the admin mistakenly adds duplicate addresses, it may lead to wasted whitelist slots.

It is recommended to check whether the newly added whitelist address is a duplicate before inserting it.

[L-04] Pre-hook transfer window allows unrestricted token movement

When the mint is created before the extra_account_meta_list account is initialized, the Transfer-Hook extension is not yet enforced. Any tokens minted or transferred during that gap bypass the whitelist and can land in non-whitelisted wallets.

Recommendation Create the mint, initialise extra_account_meta_list, and set the TransferHook program in the same transaction (or freeze the mint until all three steps are done).

[L-05] `TransferHookState` account is over-allocated by 8 bytes

The PDA is initialised with:

```
#[account(init, space = 8 + TransferHookState::SIZE, …)]
```

but also adds 8 bytes at TransferHookState:

```
impl TransferHookState {
    pub const SIZE: usize = 8 + Self::INIT_SPACE;
}
```

Adding another "+ 8" in the #[account(init, …)] line adds a second header, so the account is created with 8 (extra) + 8 (in SIZE) + 673 (fields) = 689 bytes while only 8 (header) + 673 (fields) = 681 bytes are required. For this reason, 8 bytes of rent are wasted per deployment.

Recommendation 1. Initialize the account with the SIZE constant :

```
#[account(init, space = TransferHookState::SIZE, …)]
```

[L-06] Bypass token-account address whitelisting with `SetAuthority`

The check_transfer_account() function validates transfers by comparing the source token-account's address (source_token.key()) against the whitelisted_wallets array. That means the hook trusts the account itself, not the wallet that controls it.

Because SPL Token allows the owner of any account to call SetAuthority { authority_type: AccountOwner, new_authority }, a whitelisted account can hand its ownership to any arbitrary Pubkey in a single instruction. After that call: 1. the account address remains unchanged (still whitelisted); 2. the new authority, who was never approved, can now sign transfers that the hook will accept.

Effectively, one legitimate user can delegate or sell whitelisted status to an attacker, bypassing the intended access-control. No additional funds or privileges are required— only a standard SPL instruction.

Recommendations

Check the access by the token account owner, not by its own address:

```
let owner = ctx.accounts.source_token.owner;
require!(
 transfer_hook_state.whitelisted_wallets.contains(&owner),
    TransferError::UnauthorizedTransfer
);
```