Pashov Audit Group

Pump Security Review

October 8th 2025 - October 10th 2025

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of 40+ freelance security researchers, who are well proven in the space - most have earned over $100k in public contest rewards, are multi-time champions or have truly excelled in audits with us. We only work with proven and motivated talent.

With over 300 security audits completed — uncovering and helping patch thousands of vulnerabilities — the group strives to create the absolute very best audit journey possible. While 100% security is never possible to guarantee, we do guarantee you our team's best efforts for your project.

Check out 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. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: 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** - leads to a moderate material loss of assets in the protocol or moderately harms a group of users
- **Low** - leads to a minor material loss of assets in the protocol or harms a small group of users

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

4. About Pump

Pump on Solana is a platform for launching SPL coins that can be traded on a bonding curve without needing to provide initial liquidity. Once the coin reaches a particular market cap, liquidity is deposited from the bonding curve to Raydium, and the received LP tokens are burnt. Pump AMM is an AMM on the Solana blockchain.

5. Executive Summary

A time-boxed security review of the **pump-fun/pump-contracts-solana** and **pump-fun/pump-amm-2** repositories was done by Pashov Audit Group, during which **ctrus**, **FrankCastle**, **newspace** engaged to review **Pump**. A total of **3** issues were uncovered.

Protocol Summary	
Project Name	Pump
Protocol Type	AMM and Bonding Curve tokensale
Timeline	October 8th 2025 - October 10th 2025

Review commit hashes:

- 3c16c4c7b1a67b4c70818baf220ff0e8fc30c470 (pump-fun/pump-contracts-solana)
- 8325db469809c58f489f4ba94f69872483a90754 (pump-fun/pump-amm-2)

Fixes review commit hashes:

- 2e7fc66d31e63a28fd7262ec7de3b023b93305d8 (pump-fun/pump-contracts-solana)
- ad66e35541d04b6fd9561d6fe83a7809c1c081c3 (pump-fun/pump-amm-2)

Scope

`buy.rs` `lib.rs` `fee.rs` `mod.rs` `common.rs` `constant_product.rs` `volume_accumulator.rs`

6. Findings

Findings count

Severity	Amount
Low	3
Total findings	**3**

Summary of findings

ID	Title	Severity	Status
[L-01]	`get_token_account_rent` miscalculates Token-2022 rent by ignoring extensions	Low	Resolved
[L-02]	Fees are overcharged because they are calculated before adjusting `net_sol`	Low	Resolved
[L-03]	Rent calculation uses total exemption over missing balance	Low	Resolved

Low findings

[L-01] `get_token_account_rent` miscalculates Token-2022 rent by ignoring extensions

In `utils/rent.rs`, the function `get_token_account_rent` assumes a **fixed size of bytes** for both standard SPL Token and Token-2022 accounts when computing the rent exemption amount.

However, **Token-2022 accounts can include multiple extensions** (such as transfer fees, confidential transfers, interest-bearing accounts, etc.), which increase the account's data size beyond the default 165 bytes.

Current implementation:

```
pub fn get_token_account_rent(rent: &Rent, token_program: &Pubkey) -> Result<u64> {
    Ok(if token_program == &spl_token::ID {
        rent.minimum_balance(spl_token::state::Account::LEN)
    } else {
        rent.minimum_balance(spl_token_2022::state::Account::LEN)
    })
}
```

This logic treats both `spl_token::Account` and `spl_token_2022::Account` as having the same static length (`165`), leading to **undercalculated rent exemption** for Token-2022 accounts with extensions.

As a result, accounts with extensions may **fail to be rent-exempt**, triggering **unexpected lamport drains** or runtime errors during token initialization and transfers.

Recommendations

Update the function to dynamically calculate the **total account size including extensions** for Token-2022 accounts.

A more accurate implementation would involve querying or simulating the extensions applied to the account, for example:

```
use spl_token_2022::extension::ExtensionType;

pub fn get_token_account_rent(rent: &Rent, token_program: &Pubkey, extensions:
&[ExtensionType]) -> Result<u64> {
    if token_program == &spl_token::ID {
        Ok(rent.minimum_balance(spl_token::state::Account::LEN))
    } else {
        let account_size =
spl_token_2022::extension::get_account_len::<spl_token_2022::state::Account>(extensions);
        Ok(rent.minimum_balance(account_size))
    }
}
```

This ensures rent is correctly calculated for Token-2022 accounts regardless of the extensions they use, maintaining proper rent exemption and avoiding unexpected lamport deficiencies.

[L-02] Fees are overcharged because they are calculated before adjusting `net_sol`

The protocol adjusts the `net_sol` value by subtracting excess amounts after initial computation. However, the **fees are calculated using the old (larger)** `net_sol` **value** before the adjustment takes place.

This results in **fee overcharging**, since the fee calculation does not reflect the actual final amount of SOL being transferred or used.

In simplified terms:

1. `net_sol` is first computed.
2. Fees are calculated based on this value.
3. Later, the code adjusts `net_sol` (e.g., by subtracting excess SOL).
4. But the fees remain based on the outdated, higher `net_sol` — leading to an overpayment.

This inconsistency can cause users to **lose extra SOL in fees**, misalign accounting between buyer/seller amounts, and complicate refund or reconciliation logic.

Recommendations

Recalculate the fees **after adjusting** `net_sol`, ensuring that the final fee reflects the **updated net value**.

Example fix:

```
// Adjust net_sol first
net_sol = net_sol.saturating_sub(excess_amount);

// Then recalculate fees based on the adjusted net_sol
let fee_amount = calculate_fees(net_sol);
```

This guarantees that users are charged accurate fees proportional to the actual net SOL transferred, preventing overcharging and improving accounting integrity.

[L-03] Rent calculation uses total exemption over missing balance

In `Buy::calculate_rent`, the function currently adds the **entire rent exemption amount** for accounts that are not yet rent-exempt, rather than calculating only the **difference** between the current balance and the required rent exemption.

This causes **overestimation of required lamports**, since accounts may already hold some balance. As a result, excess SOL could be transferred or reserved unnecessarily during initialization or buy operations, leading to **inefficient use of funds** Current implementation:

```
total = total
    .checked_add(rent.minimum_balance(CREATOR_VAULT_DATA_LEN))
    .ok_or(PumpError::Overflow)?;
```

However, this ignores the account's current balance. The rent should only account for the **remaining** amount needed to reach the rent-exempt threshold.

Correct calculation should be:

```
let current_balance = self.creator_vault.lamports();
let rent_exemption_lamports = rent.minimum_balance(CREATOR_VAULT_DATA_LEN);

if current_balance < rent_exemption_lamports {
    let rent_amount = rent_exemption_lamports - current_balance;
    total = total.checked_add(rent_amount).ok_or(PumpError::Overflow)?;
}
```

Recommendations

Update the rent calculation logic to consider only the difference between the rent exemption amount and the current account balance:

```
rent_amount = rent_exemption_lamports.saturating_sub(current_balance);
```

This ensures that only the **necessary lamports** are added, prevents overcharging, and maintains more accurate and efficient rent estimation across all involved accounts.