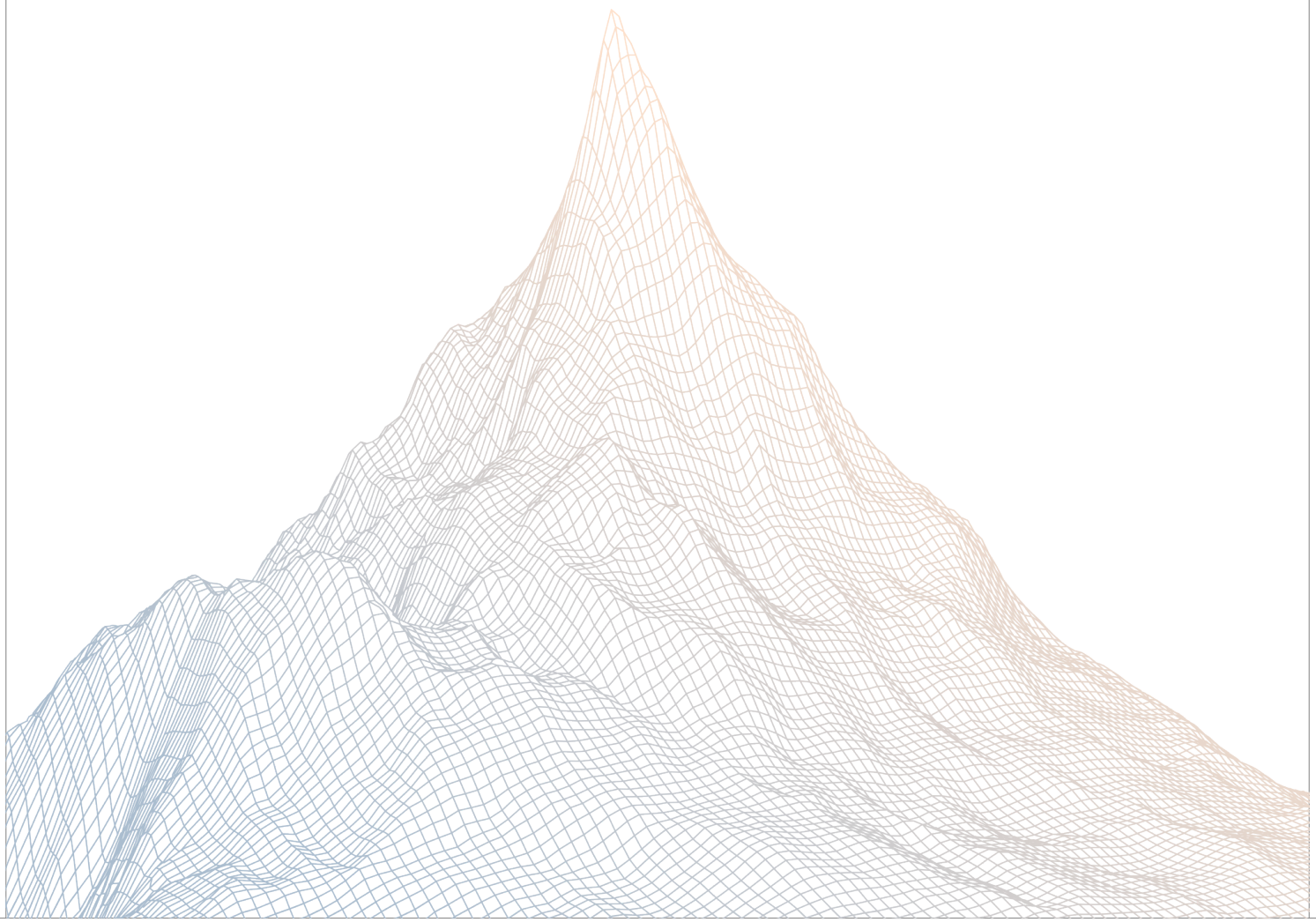
Zenith

Pyron

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: September 17th to October 7th, 2025

AUDITED BY: DadeKuma
shaflow2

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Pyron

Pyron is the asset productivity layer of Fogo. At its core, it is a composable, efficient, and scalable lending and borrowing protocol built natively for Fogo's high-speed architecture. It is designed to enable smooth capital movement between positions. Users can supply supported assets to earn interest or use them as collateral to borrow other assets. By integrating directly with Fogo's infrastructure, Pyron benefits from low latency and fast execution, enabling real-time lending operations without off-chain intermediaries.

2.2 Scope

The engagement involved a review of the following targets:

Target	program
Repository	https://github.com/pyron-finance/program
Commit Hash	`6a6f677bb09dd618a1b18edc58e899ab01396184`
Files	`programs/lendr/src/**.rs`

2.3 Audit Timeline

September 17, 2025	Audit start
October 7, 2025	Audit end
November 12, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	2
Low Risk	2
Informational	0
Total Issues	**4**

2.5 Security Note

Scope for this engagement is limited to the Liquidity Incentive Program (LIP). It operates as an application-layer extension that depends on Lendr.

Some issues identified during this audit have not been resolved at the time of publication, and have been acknowledged by the client for remediation in a future release.

3

Findings Summary

ID	Description	Status
M-1	Emissions rewards may be locked in the lendr_account controlled by lndr_pda_signer	Acknowledged
M-2	Unallocated rewards tokens cannot be withdrawn	Acknowledged
L-1	Enabling transfer fees may affect the reward compensation in the end_deposit instruction	Acknowledged
L-2	The end_deposit instruction does not close the lendr_account resulting in rent wastage	Acknowledged

4

Findings

4.1 Medium Risk

A total of 2 medium risk findings were identified.

[M-1] Emissions rewards may be locked in the `lendr_account` controlled by `lndr_pda_signer`

SEVERITY: Medium	IMPACT: Medium
STATUS: Acknowledged	LIKELIHOOD: High

Target

- end_deposit.rs

Description:

During the liquidity locker locking process, emissions rewards may be generated in the `lendr_account`. These emissions rewards are settled during the `withdraw` cpi.

```
pub fn withdraw_all(&mut self) -> LendrResult<u64> {
    self.claim_emissions(Clock::get()?.unix_timestamp as u64)?;
    // ...
}
```

The `lendr_account` actually accrues emissions rewards and consumes the bank's `emissions_remaining`, but there is no instruction available to claim these rewards. Moreover, since `emissions_destination_account` is not configured, permissionless claim cannot be executed either.

This may cause some emission tokens to become unclaimable.

Recommendations:

It is recommended to add relevant instructions to withdraw the accumulated emission tokens from the `lendr_account` controlled by `lndr_pda_signer`.

Pyron: Acknowledged.

[M-2] Unallocated rewards tokens cannot be withdrawn

SEVERITY: Medium	IMPACT: Medium
STATUS: Acknowledged	LIKELIHOOD: High

Target

- end_deposit.rs

Description:

When a Campaign is created, the admin deposits the `max_rewards` amount of reward tokens to ensure that lockers' yields are guaranteed even if the profits are insufficient. At `end_deposit`, this portion of tokens will be used to cover the profit shortfall, supplementing the part calculated as `deposit_amount * max_rewards / max_deposits - base_yield`.

```
let additional_reward_amount = {
    let initial_deposit = ctx.accounts.deposit.amount;
    let end_deposit = ctx.accounts.temp_token_account.amount;

    let base_yield = end_deposit.saturating_sub(initial_deposit);

    let max_rewards_pre_campaign
    = I80F48::from_num(ctx.accounts.campaign.max_rewards);
    let max_deposits_pre_campaign
    = I80F48::from_num(ctx.accounts.campaign.max_deposits);
    let deposit_amount = I80F48::from_num(ctx.accounts.deposit.amount);

    let max_reward_for_deposit = deposit_amount
        .checked_div(max_deposits_pre_campaign)
        .unwrap()
        .checked_mul(max_rewards_pre_campaign)
        .unwrap()
        .checked_to_num::<u64>()
        .unwrap();

    msg!(
        "Base yield: {}, max reward for deposit: {}",
        base_yield,
        max_reward_for_deposit
    );
```

```
    max_reward_for_deposit.saturating_sub(base_yield)
};
```

When the total deposits reach `max_deposits`, the `max_rewards` may not be fully allocated. This portion of tokens cannot be claimed by anyone.

Recommendations:

It is recommended to allow the Campaign `admin` to withdraw the unutilized reward tokens, as well as the portion calculated as `min(base_yield, max_reward_for_deposit)` at each `end_deposit`.

Pyron: Acknowledged.

4.2 Low Risk

A total of 2 low risk findings were identified.

[L-1] Enabling transfer fees may affect the reward compensation in the `end_deposit` instruction

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- end_deposit.rs

Description:

In the `create_deposit` instruction, tokens with the Token2022 `TransferFeeConfig` extension whose current cycle fee is set to 0 are permitted to participate in locking.

```
pub fn nonzero_fee(mint_ai: AccountInfo, epoch: u64) -> LendrResult<bool> {
    if mint_ai.owner.eq(&Token::id()) {
        return Ok(false);
    }

    let mint_data = mint_ai.try_borrow_data()?;
    let mint =
    StateWithExtensions::<spl_token_2022::state::Mint>::unpack(&mint_data)?;

    if let Ok(transfer_fee_config)
    = mint.get_extension::<TransferFeeConfig>() {
        return Ok(u16::from(
            transfer_fee_config
                .get_epoch_fee(epoch)
                .transfer_fee_basis_points,
        ) != 0);
    }

    Ok(false)
}
```

However, in the `end_deposit` instruction, the possibility that transfer fees may be enabled during the current cycle is not considered. This causes the balance in `temp_token_account` to reflect the amount after deducting transfer fees.

```
let additional_reward_amount = {
    let initial_deposit = ctx.accounts.deposit.amount;
    let end_deposit = ctx.accounts.temp_token_account.amount;

    let base_yield = end_deposit.saturating_sub(initial_deposit);

    let max_rewards_pre_campaign
    = I80F48::from_num(ctx.accounts.campaign.max_rewards);
    let max_deposits_pre_campaign
    = I80F48::from_num(ctx.accounts.campaign.max_deposits);
    let deposit_amount = I80F48::from_num(ctx.accounts.deposit.amount);

    let max_reward_for_deposit = deposit_amount
        .checked_div(max_deposits_pre_campaign)
        .unwrap()
        .checked_mul(max_rewards_pre_campaign)
        .unwrap()
        .checked_to_num::<u64>()
        .unwrap();

    msg!(
        "Base yield: {}, max reward for deposit: {}",
        base_yield,
        max_reward_for_deposit
    );

    max_reward_for_deposit.saturating_sub(base_yield)
};
```

Using this balance to calculate `base_yield` may result in a smaller value or even a negative value, leading to additional reward compensation.

Recommendations:

It is recommended to calculate the token balance before fees and use it to compute `base_yield` if a transfer fee is enabled.

Pyron: Acknowledged.

[L-2] The `end_deposit` instruction does not close the `lendr_account` resulting in rent wastage

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: High

Target

- end_deposit.rs

Description:

After the deposit ends in the locker, the Deposit account will be closed and the corresponding `lendr_account` will become unused.

```
#[derive(Accounts)]
pub struct EndDeposit<'info> {
    //...
    #[account(
        mut,
        close = signer,
    )]
    pub deposit: Box<Account<'info, Deposit>>,
    //...
    #[account(
        mut,
        seeds = [
            LENDR_ACCOUNT_SEED.as_bytes(),
            deposit.key().as_ref(),
        ],
        bump,
    )]
    /// CHECK: Asserted by PDA derivation
    pub lendr_account: AccountInfo<'info>,
    ...
}
```

Due to the `liquidity-incentive-program` lacking a CPI to close this account, the account cannot be closed, resulting in rent wastage.

Recommendations:

It is recommended to provide instructions that allow calling the Lendr program's `close_balance` and `close_account` functions to close the `lendr_account` and release rent.

Pyron: Acknowledged.