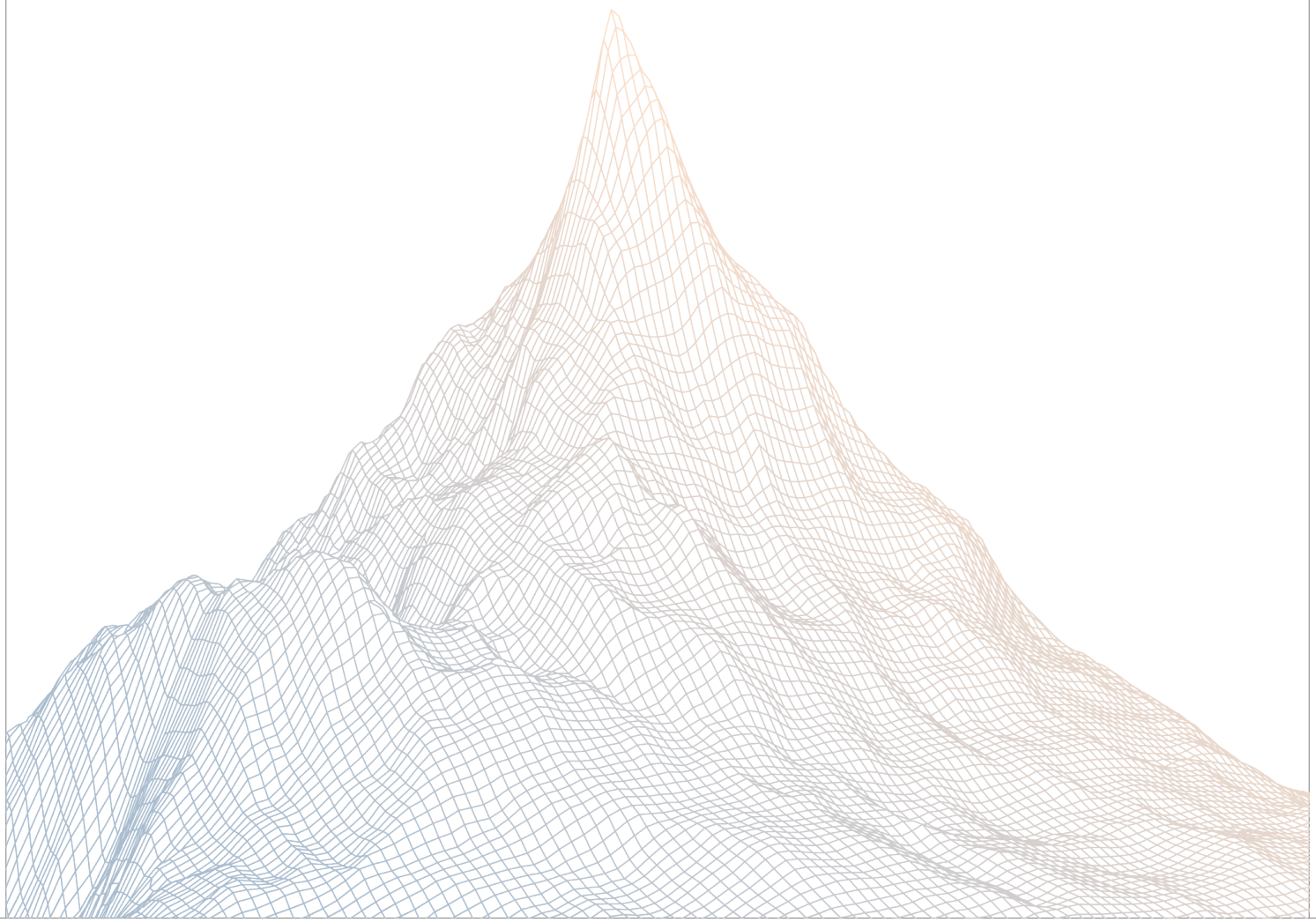
Zenith

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
October 23, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	13
Low Risk	6
Informational	4
Total Issues	**24**

2.5 Security Note

Scope for this engagement is limited to the Lendr protocol. The Liquidity Incentive Program (LIP) operates as an application-layer extension that depends on Lendr; therefore, LIP's design and implementation are out of scope for this report unless explicitly referenced. Conclusions in this report should be interpreted as pertaining to Lendr independently of LIP.

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
H-1	Stale share values in bankruptcy checks result in a wrong bankruptcy handling	Resolved
M-1	The emissions reward design has flaws	Acknowledged
M-2	Unclaimed emission tokens will be locked for an ACCOUNT_DISABLED account	Resolved
M-3	Interest from the previous period is not accumulated before modifying the configuration	Resolved
M-4	Improper management of bank flags	Resolved
M-5	Closing lendr_account does not check whether emissions have been claimed	Resolved
M-6	lending_pool_add_bank_permissionless did not charge a bank initialization fee	Resolved
M-7	Liquidators may pay an unexpected debt amount during liquidation	Acknowledged
M-8	Instant liquidation upon unpausing may prevent user recovery	Acknowledged
M-9	Some Token-2022 extensions may break the protocol	Resolved
M-10	Wrong meta order causes runtime failure and denial of service	Resolved
M-11	Improper flag validation in get_group_bank_config	Resolved
M-12	Unaddressed small positions can accumulate systematic bad debt	Acknowledged
M-13	Exploitable instant withdrawal during bankruptcy events	Acknowledged
L-1	Unused DepositOnly validation branch during deposit	Resolved
L-2	Unused and dangerous set_active function should be removed	Resolved
L-3	sol_pool_adjusted_balance calculation may overflow	Resolved
L-4	The operational_state cannot be adjusted after FREEZE_SETTINGS is enabled	Resolved

ID	Description	Status
L-5	Unused BorrowOnly validation branch	Resolved
L-6	Improper get_side logic	Resolved
I-1	Slashing could significantly impact LST pricing	Acknowledged
I-2	Float numbers are costly in terms of CU	Resolved
I-3	Unused decrease_balance function	Resolved
I-4	Unused account in LendrAccountSetAccountAuthority context	Resolved

4

Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] Stale share values in bankruptcy checks result in a wrong bankruptcy handling

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Medium

Target

- programs/lendr/src/instructions/lendr_group/handle_bankruptcy.rs

Description:

During a bankruptcy, the `lending_pool_handle_bankruptcy` function determines insolvency through an health check, by calculating weighted asset and liability values using `calc_weighted_value`.

Asset values are derived via `get_asset_amount`, which relies on `asset_share_value`. The same happens with liabilities, within its corresponding functions, to update `liability_share_value`.

If the account is considered bankrupt, it then proceeds with accruing the interest, updating both values.

However, the initial share values used in the health check may be stale unless interest has been accrued earlier in the same block through other operations, which may be unlikely in groups that have low market action activity.

Since `accrue_interest` updates both `asset_share_value` and `liability_share_value` to reflect accrued interest, using outdated values during the health check can lead to incorrect bankruptcy determinations.

This may result in either premature bad debt handling or delayed recognition of insolvent positions, depending on how interest accumulation affects the individual position values, ultimately resulting in a loss of funds for all the users in both cases.

Recommendations:

Consider performing the health check only after accruing the interest.

Pyron: Resolved with @b43db7f73a...

Zenith: Verified.

4.2 Medium Risk

A total of 13 medium risk findings were identified.

[M-1] The emissions reward design has flaws

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

Target

- configure_bank.rs
- lendr_account.rs

Description:

The current emissions reward system has some flaws:

1. In the emissions reward system, the group admin sets the emission rewards by specifying a per-second emission rate and transferring all reward tokens into the system during configuration.

   When users interact with the system or manually call the `lending_account_settle_emissions` instruction, the account's emission rewards are settled. Once all rewards have been settled, no additional rewards are generated.

```
let emissions_rate = I80F48::from_num(self.bank.emissions_rate);
let emissions = calc_emissions(
    period,
    balance_amount,
    self.bank.mint_decimals as usize,
    emissions_rate,
)?;

let emissions_real = min(emissions,
    I80F48::from(self.bank.emissions_remaining));
```

Because rewards that are unclaimed within a unit of time are not considered, emission rewards may be over-allocated to users who interact with the system frequently. Although

users who do not interact frequently with the protocol have existed in the system for a long time, their rewards are lost because the rewards have been over-allocated.

2. At any time, the group admin can start a new emission round. But there is no clear start time for each emission round. For accounts that have not been updated for a long time, the time outside the emission period is also counted as rewards, which may allow one person to drain the rewards.

```
let last_update = if self.balance.last_update < MIN_EMISSIONS_START_TIME {
    current_timestamp
} else {
    self.balance.last_update
};
let period = I80F48::from_num(
    current_timestamp
        .checked_sub(last_update)
        .ok_or_else(math_error!())?,
);
```

Recommendations:

It is recommended to modify the reward emissions to be calculated globally, so that whenever anyone interacts with the system, the global emission rewards are settled and stored (in units of shares).

Pyron: The current emissions reward system is designed with the expectation emission parameters are configured based on the maximum deposit cap for each bank. When emissions are properly allocated according to the deposit limit (e.g., setting total_emissions to match the target APR × deposit_limit × time_period), the unfair distribution concerns become negligible as the total deposit amount is bounded and predictable. While minor timing advantages between early and late depositors may still exist within the cap, their impact is minimal compared to scenarios without proper capacity planning. That said, we acknowledge the theoretical benefits of a global share-based emissions system, and the emissions mechanism is currently a work-in-progress that will be redesigned in subsequent protocol updates to further enhance fairness and eliminate any residual order-dependency issues.

Zenith: Acknowledged.

[M-2] Unclaimed emission tokens will be locked for an `ACCOUNT_DISABLED` account

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- emissions.rs

Description:

If borrowing emissions are enabled, bad debt accounts will also accumulate emission rewards, and these rewards are settled during the `repay` process in `handle_bankruptcy`.

```
// Settle bad debt.
// The liabilities of this account and global total liabilities are reduced
    by `bad_debt`
BankAccountWrapper::find(
    &bank_loader.key(),
    &mut bank,
    &mut lendr_account.lending_account,
)?
.repay(bad_debt)?;
```

After a bad debt account is settled, the account is set to `ACCOUNT_DISABLED`. Accounts in this state are prohibited from claiming emission rewards, which can cause the reward tokens to be stuck.

```
pub fn lending_account_withdraw_emissions<'info>(
    ctx: Context<'_, '_, 'info, 'info,
    LendingAccountWithdrawEmissions<'info>>,
) -> LendrResult {
    let mut lendr_account = ctx.accounts.lendr_account.load_mut()?;

    check!(
        !lendr_account.get_flag(ACCOUNT_DISABLED),
        LendrError::AccountDisabled
    );
```

Recommendations:

It is recommended to add an instruction to allow emission rewards of an `ACCOUNT_DISABLED` account to be returned to the emission pool.

Pyron: Resolved with @c5c4452...

Zenith: Verified.

[M-3] Interest from the previous period is not accumulated before modifying the configuration

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- configure_bank.rs

Description:

When modifying a bank's configuration, interest-related parameters of the bank may be updated. However, before these parameters are modified, `accrue_interest` is not called to accumulate interest from the previous period.

```
} else {
    // Settings are not frozen, everything updates
    bank.configure(&bank_config)?;
    msg!("Bank configured!");

    if bank_config.oracle_max_age.is_some() {
        bank.config.validate_oracle_age()?;
    }

    emit!(LendingPoolBankConfigureEvent {
        header: GroupEventHeader {
            lendr_group: ctx.accounts.group.key(),
            signer: Some(*ctx.accounts.admin.key)
        },
        bank: ctx.accounts.bank.key(),
        mint: bank.mint,
        config: bank_config,
    });
}
```

If the bank is inactive and interest rates change significantly, the interest for the previous period will be calculated based on the updated parameters. Changes to the group parameters can also cause the same issue.

Recommendations:

It is recommended to call `accrue_interest` to update interest before changing a bank's configuration, limit the number of banks in a group, and update the interest of all banks when the group parameters are updated.

Pyron: Resolved with @2cde394... and @166e22c...

Zenith: Verified.

[M-4] Improper management of bank flags

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- configure_bank.rs

Description:

There are currently four valid bits for bank flags. The first two bits are related to EMISSIONS. The third bit configures whether PERMISSIONLESS_BAD_DEBT_SETTLEMENT is enabled. The fourth bit is FREEZE_SETTINGS. Once FREEZE_SETTINGS is enabled, only the bank's deposit and borrow limits can be updated thereafter.

```
if bank.get_flag(FREEZE_SETTINGS) {
    bank.configure_unfrozen_fields_only(&bank_config)?;

    msg!("WARN: Only deposit+borrow limits updated. Other settings IGNORED
    for frozen banks!");

    emit!(LendingPoolBankConfigureFrozenEvent {
        header: GroupEventHeader {
            lendr_group: ctx.accounts.group.key(),
            signer: Some(*ctx.accounts.admin.key)
        },
        bank: ctx.accounts.bank.key(),
        mint: bank.mint,
        deposit_limit: bank.config.deposit_limit,
        borrow_limit: bank.config.borrow_limit,
    });
} else {
    // Settings are not frozen, everything updates
    bank.configure(&bank_config)?;
    msg!("Bank configured!");

    if bank_config.oracle_max_age.is_some() {
        bank.config.validate_oracle_age()?;
    }
```

```
    emit!(LendingPoolBankConfigureEvent {
        header: GroupEventHeader {
            lendr_group: ctx.accounts.group.key(),
            signer: Some(*ctx.accounts.admin.key)
        },
        bank: ctx.accounts.bank.key(),
        mint: bank.mint,
        config: bank_config,
    });
}
```

However, in `lending_pool_setup_emissions`, `override_emissions_flag` is not implemented correctly. `verify_emissions_flags` restricts the flags to be updated to only include the first two bits of configuration, which will cause the original configuration in the last two bits of the flags to be cleared.

```
pub(crate) const EMISSION_FLAGS: u64 = EMISSIONS_FLAG_BORROW_ACTIVE |
    EMISSIONS_FLAG_LENDING_ACTIVE;
pub(crate) fn override_emissions_flag(&mut self, flag: u64) {
    assert!(Self::verify_emissions_flags(flag));
    self.flags = flag;
}
const fn verify_emissions_flags(flags: u64) -> bool {
    flags & EMISSION_FLAGS == flags
}
```

In `lending_pool_update_emissions_parameters`, `emissions_flags` is directly set to `bank.flags` without any checks. This means that even if the bank has already enabled `FREEZE_SETTINGS`, it is still possible to unset the freeze and reconfigure the parameters.

```
if let Some(flags) = emissions_flags {
    msg!("Updating emissions flags to {:#010b}", flags);
    bank.flags = flags;
}
```

Recommendations:

It is recommended to re-implement the `override_emissions_flag` function so that it only overrides the first two bits related to `EMISSIONS` without affecting the other flag bits. Additionally, the `override_emissions_flag` function should also be used when configuring flags in `lending_pool_update_emissions_parameters`.

Pyron: Resolved with @c218d4b...

Zenith: Verified.

[M-5] Closing `lendr_account` does not check whether emissions have been claimed

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- close.rs

Description:

The `lendr_account` can be closed. In the `can_be_closed` check, the function iterates through the account's balances, and if each balance's `asset_shares` and `liability_shares` are both below the `EMPTY_BALANCE_THRESHOLD`, the account is considered closable.

```
pub fn can_be_closed(&self) -> bool {
    let is_disabled = self.get_flag(ACCOUNT_DISABLED);
    let only_has_empty_balances = self
        .lending_account
        .balances
        .iter()
        .all(|balance| balance.get_side().is_none());

    !is_disabled && only_has_empty_balances
}
```

However, there is no check for whether the balance still has unclaimed emissions. If the account is closed, any unclaimed emissions will be locked.

Recommendations:

It is recommended to check whether all balances have `active` equal to 0. Because when `active` is 0, it proves that `lending_account_close_balance` has been called to close the balance. This ensures that `emissions_outstanding` is below the tolerance threshold.

Pyron: Resolved with @382bc6b...

Zenith: Verified.

[M-6] `lending_pool_add_bank_permissionless` did not charge a bank initialization fee

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- add_pool_permissionless.rs

Description:

In both `lending_pool_add_bank_with_seed` and `lending_pool_add_bank`, the `fee_payer` pays the `bank_init_flat_sol_fee` to the global fee recipient.

```
// Transfer the flat sol init fee to the global fee wallet
let fee_state = ctx.accounts.fee_state.load()?;
let bank_init_flat_sol_fee = fee_state.bank_init_flat_sol_fee;
if bank_init_flat_sol_fee > 0 {
    anchor_lang::system_program::transfer(
        ctx.accounts.transfer_flat_fee(),
        bank_init_flat_sol_fee as u64,
    )?;
}
```

However, creating an `ASSET_TAG_STAKED` bank via `lending_pool_add_bank_permissionless` does not charge the `bank_init_flat_sol_fee`. This causes the global fee recipient to receive less than expected, and a malicious actor could cheaply increase the number of banks in the `lendr_group`.

Recommendations:

It is recommended to also charge the `bank_init_flat_sol_fee` in the `lending_pool_add_bank_permissionless` instruction.

Pyron: Resolved with @8fc3a56...

Zenith: Verified.

[M-7] Liquidators may pay an unexpected debt amount during liquidation

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

Target

- lendr/src/instructions/lendr_account/liquidate.rs

Description:

Liquidators can purchase discounted collateral from unhealthy accounts by paying off the outstanding debt.

However, the current mechanism lacks slippage protection for the debt payment amount. Liquidators receive the expected collateral amount, but the actual debt they must pay may vary due to asset price fluctuations between transaction submission and execution.

This exposes liquidators to potential overpayment if asset prices change during this period, reducing their expected profits from the liquidation process.

Recommendations:

Consider implementing a slippage parameter that allows liquidators to set a maximum acceptable debt payment amount.

Pyron: Existing Protection: The system already provides substantial slippage protection through the price bias mechanism, using conservative pricing (PriceBias::Low for collateral, PriceBias::High for debt) that offers up to 10% protection against adverse price movements during normal market conditions.Liquidator Choice: Liquidators have the ability to opt out of more volatile banks if they choose to do so, allowing them to manage their own risk exposure through asset selection and position sizing. slippage is better suited as a client side feature.

Protocol Health Priority: The risk of positions not getting liquidated and creating bad debt is significantly greater compared to liquidators experiencing overpayment. Failed liquidations due to overly restrictive slippage protection could threaten protocol solvency, while liquidator losses remain isolated to individual participants.

Zenith: Acknowledged. Slippage protection will be implemented client-side by using an additional instruction after the liquidation instruction.

[M-8] Instant liquidation upon unpausing may prevent user recovery

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

Target

- programs/lendr/src/state/lendr_group.rs

Description:

The protocol uses `assert_operational_mode` to block major operations when the bank is paused.

However, during a pause, market conditions may continue to fluctuate, potentially causing previously healthy positions to become undercollateralized.

When the bank is unpaused, these positions become immediately eligible for liquidation without providing users an opportunity to repay their debt positions.

This results in users losing their collateral without any chance to take corrective action such as adding more collateral or repaying debt.

Recommendations:

Consider introducing a grace period following the unpausing of the bank before liquidations are permitted.

Pyron: The pause system is intended to occur only during catastrophic events like oracle failures or active exploits where users face bigger risks like total fund loss, making liquidation timing secondary concern. Additionally, implementing grace periods would severely degrade protocol safety and robustness by delaying critical liquidations needed for protocol health during recovery periods.

Zenith: Acknowledged.

[M-9] Some Token-2022 extensions may break the protocol

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- deposit.rs
- withdraw.rs
- borrow.rs
- repay.rs
- liquidate.rs

Description:

Some token-2022 extensions may cause problems with the protocol:

- Mint Close Authority - A token with this extension may change its decimals by closing an re-opening with a new decimal amount. An attacker can close and re-open mint account with a different decimal after the token has been added to the bank through group admin review. This allows the attacker to set decimals exceeding `MAX_EXP_10_I80F48`. Anyone holding such a token can deposit it into the vault, making their `lendr_account` impossible to liquidate or have bad debt settled. When they want to restore normal behavior, they only need to fully withdraw that token from the `lendr_account` once.
- Default Account State - A newly created fee account may be frozen.
- Pausable - Token paused may affect various operations related to the token, including liquidation, because the liquidation process requires transferring the `insurance_fee`.
- Transfer Hooks - Accounts with extensions that have a transfer hook may require some additional accounts in `remaining_accounts`. However, for operations like `withdraw`, `remaining_accounts` is expected to include only certain accounts required by oracles and the bank. This mismatch could cause the extension call to fail.

A related issue is that even normal SPL tokens have a freeze who can use that authority to prevent tokens from being transferred to/from any user's address. This may cause liquidations to fail or may prevent borrowers from repaying loans.

Recommendations:

It is recommended to check the mint extensions when adding a bank. If the mint has extensions that are explicitly unsupported, the creation of the bank should be blocked. Additionally, the handling logic for mints with `freeze_authority` privileges should be taken into consideration.

Pyron: Resolved with @76ea2c5...

Zenith: Verified.

[M-10] Wrong meta order causes runtime failure and denial of service

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- programs/lendr/src/state/lendr_account.rs#L1493-L1498
- programs/lendr/src/state/lendr_account.rs#L1513

Description:

The `deposit_spl_transfer_with_session` function contains a mismatch between the account metadata and the actual instruction parameter order:

```
let instruction = fogo_transfer_checked(
    &program.key,
    &from.key,
    &mint.key(),
    &to.key,
    &session_key.key,
    Some(&program_signer.key),
    amount,
    mint.decimals,
)?;
invoke_signed(
    &instruction,
    &[
        from,
        to,
        program_signer,
        mint.to_account_info(),
        program,
        session_key,
    ],
    &[&[PROGRAM_SIGNER_SEED, &[program_signer_bump]],
)?;
```

This inconsistency causes the transaction to revert at runtime, resulting in a denial of

service condition where users cannot successfully execute this operation. This affects deposit and repay operations that use a session.

Recommendations:

Consider using the following order here:

```
invoke_signed(
    &instruction,
    &[
        from,
        to,
        program_signer,
        mint.to_account_info(),
        program,
        session_key,
        from,                  // source
        mint.to_account_info(), // mint
        to,                    // destination
        session_key,           // authority
        program_signer,        // signer
    ],
    &[&[PROGRAM_SIGNER_SEED, &[program_signer_bump]],
)?;
```

and here:

```
invoke_signed(
    &instruction,
    &[from, to, program_signer, program, session_key],
    &[from, to, session_key, program_signer],
    &[&[PROGRAM_SIGNER_SEED, &[program_signer_bump]],
)?;
```

Pyron: Resolved with @e261de5...

Zenith: Verified.

[M-11] Improper flag validation in `get_group_bank_config`

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- lendr_group.rs

Description:

The `get_group_bank_config` function is used to retrieve `GroupBankConfig`, but the function incorrectly determines whether `PROGRAM_FEES_ENABLED` is enabled by checking if `group_flags` is equal to `PROGRAM_FEES_ENABLED`.

```
pub fn get_group_bank_config(&self) -> GroupBankConfig {
    GroupBankConfig {
        program_fees: self.group_flags == PROGRAM_FEES_ENABLED,
    }
}
```

Currently, `group_flags` also includes the `ARENA_GROUP` valid bit. If `ARENA_GROUP` is enabled, then `PROGRAM_FEES_ENABLED` will always be considered disabled regardless of its actual state, which causes program fees not to be settled during interest accrual.

```
pub fn get_fees(&self) -> Fees {
    let (protocol_fee_rate, protocol_fee_fixed) = if self.add_program_fees {
        (self.program_fee_rate, self.program_fee_fixed)
    } else {
        (I80F48::ZERO, I80F48::ZERO)
    };
    // ...
}
```

Recommendations:

It is recommended to only check whether the bit corresponding to `PROGRAM_FEES_ENABLED` is enabled.

```
pub fn get_group_bank_config(&self) -> GroupBankConfig {
    GroupBankConfig {
        program_fees: self.group_flags == PROGRAM_FEES_ENABLED,
        program_fees: self.program_fees_enabled(),
    }
}
```

Pyron: Resolved with @b441a2b...

Zenith: Verified.

[M-12] Unaddressed small positions can accumulate systematic bad debt

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

Target

- programs/lendr/src/instructions/lendr_account/liquidate.rs

Description:

Small liquidatable positions may not be economically viable for liquidators to process due to insufficient incentives (or it might even be impossible with extremely small positions, if the insurance fund fee is zero).

If left unaddressed, these positions can accumulate bad debt, distributing small losses across all depositors. Individual losses may be minor, however, systematic occurrence of these ignored positions could lead to significant cumulative losses over time.

An attacker could reduce their liquidation risk by crafting a transaction with many small borrows that total a large loan. Liquidators are not incentivized to intervene, as the fees for each small liquidation could make the process unprofitable after fees and gas costs.

The result is a large underwater position. Once the insurance fund is depleted, the loss would ultimately fall on the users. The system's goal should be to have as few bankruptcies as possible.

Recommendations:

Consider implementing minimum position size requirements during deposit, withdrawal, and liquidation operations.

Pyron: We're going to acknowledge this issue for now and circle back to fix it later.

Zenith: Acknowledged for now.

[M-13] Exploitable instant withdrawal during bankruptcy events

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

Target

- programs/lendr/src/state/lendr_group.rs
- programs/lendr/src/instructions/lendr_account/withdraw.rs

Description:

The protocol's bankruptcy mechanism distributes bad debt among all depositors when a position with bad debt is closed.

However, users can monitor positions approaching bankruptcy and exploit the system by performing instant withdrawals followed by immediate redeposits after the position with bad debt is closed.

This allows them to avoid the socialized loss allocation, creating an unfair advantage for technically advanced users over ordinary depositors, who will bear a greater loss.

Recommendations:

Consider introducing a time-delayed withdrawal mechanism to prevent opportunistic fund movement around bankruptcy events.

Pyron: While the timing attack vulnerability is technically valid, the risk levels are acceptable for production given the protocol's protective mechanisms. The insurance funds provide a buffer for smaller bad debt events, while automated bankruptcy handling bots in mainnet will minimize exploitation windows by processing insolvencies within blocks rather than allowing time for users to strategically withdraw. This rapid automated response, combined with insurance coverage, reduces the vulnerability to rare edge cases during extreme market stress when both automation fails and insurance is depleted. The residual risk of unfair loss distribution during such catastrophic scenarios is an acceptable trade-off compared to implementing withdrawal delays that would severely compromise the protocol's DeFi competitiveness.

Zenith: Acknowledged.

4.3 Low Risk

A total of 6 low risk findings were identified.

[L-1] Unused `DepositOnly` validation branch during deposit

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- program/src/branch/main/programs/lendr/src/state/lendr_account.rs#1057
- program/src/branch/main/programs/lendr/src/state/lendr_account.rs#L1261

Description:

The deposit intentionally allows users to repay their own debt, which is documented as expected behavior.

However, the code contains an unused validation branch designed for a `DepositOnly` operation type that would prevent debt repayment during deposits:

```
BalanceIncreaseType::DepositOnly => {
    check!(

    liability_amount_decrease.is_zero_with_tolerance(ZERO_AMOUNT_THRESHOLD),
        LendrError::OperationDepositOnly
    );
}
```

Since this branch is never executed in the current implementation, the validation logic remains dead code that doesn't affect actual protocol behavior.

Recommendations:

Depending on the intended implementation, consider one of the following:

1) Remove the `BalanceIncreaseType::DepositOnly` branch/enum entirely or 2) Fix the deposit function:

```
pub fn deposit(&mut self, amount: I80F48) -> LendrResult {
    self.increase_balance_internal(amount, BalanceIncreaseType::Any)

    self.increase_balance_internal(amount, BalanceIncreaseType::DepositOnly)
}
```

Pyron: Resolved with @0e23436...

Zenith: Verified.

[L-2] Unused and dangerous `set_active` function should be removed

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/lendr/src/state/lendr_account.rs#L909

Description:

The `active` flag is used in various parts of the codebase to determine whether a balance can be safely overridden. A function named `set_active` exists to manually modify this flag, but it is never called in the current implementation.

This function presents a potential risk if used incorrectly in the future, as deactivating a balance could lead to the loss of `asset_shares` if the balance is subsequently overwritten while in an inactive state.

Recommendations:

Consider removing the function.

Pyron: Resolved with @99834a4...

Zenith: Verified.

[L-3] `sol_pool_adjusted_balance` calculation may overflow

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- price.rs

Description:

When computing `sol_pool_adjusted_balance`, the protocol assumes `stake.delegation.stake` is greater than `lamports_per_sol`. If this condition is not met in practice, an error will be thrown.

```
let sol_pool_balance = stake.delegation.stake;
// Note: When the pool is fresh, it has 1 SOL in it (an initial and
    non-refundable
// balance that will stay in the pool forever). We don't want to include that
// balance when reading the quantity of SOL that has been staked from actual
// depositors (i.e. the amount that can actually be redeemed again).
let lamports_per_sol: u64 = 1_000_000_000;
let sol_pool_adjusted_balance = sol_pool_balance
    .checked_sub(lamports_per_sol)
    .ok_or_else(math_error!())?;
```

However, although Solana does not have a programmatic slashing mechanism, there is a manual, community-driven social slashing. If a validator engages in malicious behavior, such as compromising network liveness or security, honest participants can coordinate off-chain to initiate a hard fork, restart the network, and slash the violator's stake. In such cases, the slashing rate can reach 100%, which may cause the malicious user's `stake.delegation.stake` to fall below `lamports_per_sol`, resulting in an exception being thrown during health checks and preventing liquidation operations. The probability of such an extreme case occurring is very low. To date, no validator on Solana has ever been slashed.

Recommendations:

It is recommended to use `saturating_sub` instead of `checked_sub` when calculating `sol_pool_adjusted_balance`.

Resolved with Resolved with @e47d447...

Zenith: Verified.

[L-4] The `operational_state` cannot be adjusted after `FREEZE_SETTINGS` is enabled

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Medium

Target

- configure_bank.rs

Description:

After `FREEZE_SETTINGS` is enabled only `deposit_limit` and `borrow_limit` can be adjusted but the `operational_state` field belongs to bank risk control and cannot be changed once frozen which may render this field ineffective.

```
// If settings are frozen, you can only update the deposit and borrow limits,
    everything else is ignored.
if bank.get_flag(FREEZE_SETTINGS) {
  bank.configure_unfrozen_fields_only(&bank_config)?;

  msg!("WARN: Only deposit+borrow limits updated. Other settings IGNORED for
    frozen banks!");

  emit!(LendingPoolBankConfigureFrozenEvent {
      header: GroupEventHeader {
          lendr_group: ctx.accounts.group.key(),
          signer: Some(*ctx.accounts.admin.key)
      },
      bank: ctx.accounts.bank.key(),
      mint: bank.mint,
      deposit_limit: bank.config.deposit_limit,
      borrow_limit: bank.config.borrow_limit,
  });
}
```

For example, after enabling `FREEZE_SETTINGS` it is impossible to switch between Operational and ReduceOnly modes. Worse, if `FREEZE_SETTINGS` is enabled while `operational_state` is Paused the bank will be permanently locked.

Recommendations:

It is recommended to allow modifying the `operational_state` when `FREEZE_SETTINGS` is enabled.

Pyron: Resolved with @2ce304f...

Zenith: Verified.

[L-5] Unused `BorrowOnly` validation branch

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- program/src/branch/main/programs/lendr/src/state/lendr_account.rs

Description:

Similar to the other issue `Unused DepositOnly validation branch during deposit`, the `BorrowOnly` validation branch remains unused in the current implementation.

This dead code serves no functional purpose in the protocol's operations and doesn't impact actual system behavior, while adding unnecessary complexity to the codebase.

Recommendations:

Consider removing the unused `BorrowOnly` validation branch and associated logic to simplify the codebase and improve maintainability.

Alternatively, if this functionality is intended for future use, properly integrate it into the borrow flow and document its purpose clearly.

Pyron: Resolved with @df9eaa1...

Zenith: Verified.

[L-6] Improper `get_side` logic

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- lendr_account.rs

Description:

In the `get_side` logic, any balance with less than 1 share is treated as zero, because any balance below 1 SPL token amount is considered none.

```
pub fn get_side(&self) -> Option<BalanceSide> {
    let asset_shares = I80F48::from(self.asset_shares);
    let liability_shares = I80F48::from(self.liability_shares);

    assert!(
        asset_shares < EMPTY_BALANCE_THRESHOLD || liability_shares <
    EMPTY_BALANCE_THRESHOLD
    );

    if I80F48::from(self.liability_shares) >= EMPTY_BALANCE_THRESHOLD {
        Some(BalanceSide::Liabilities)
    } else if I80F48::from(self.asset_shares) >= EMPTY_BALANCE_THRESHOLD {
        Some(BalanceSide::Assets)
    } else {
        None
    }
}
```

However, since converting shares to SPL token amount requires multiplying by `share_value`, it is possible for less than 1 share to correspond to more than 1 SPL token amount, but it would still be incorrectly treated as zero.

This discrepancy could allow a malicious actor to steal dust amounts of tokens. For example, an attacker could open a flash loan, borrow tokens with less than 1 share, then close the `lendr_account` and reopen it to terminate the flash loan. Each time, the attacker could extract dust amounts of tokens.

Recommendations:

It is recommended to treat a balance as `NONE` only when the SPL token amount is less than 1, rather than when the share amount is less than 1.

Pyron: Resolved with @753f6b7...

Zenith: Verified.

4.4 Informational

A total of 4 informational findings were identified.

[I-1] Slashing could significantly impact LST pricing

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

Target

- price.rs

Description:

Although Solana currently does not have program-level automatic slashing, slashing mechanisms may be introduced in the future. Several proposals are under discussion, such as SIMD-0204 and PR-212. Additionally, there are related program in testnets for validating and recording malicious behavior.

Once a slashing mechanism is introduced, the risk of using LST as collateral would increase significantly, as slashing could drastically affect its pricing and potentially lead to liquidations or bad debt.

Recommendations:

When slashing is introduced, consider use measures such as disabling permissionless addition of LST and using lower LTV ratios to mitigate risk exposure

Pyron: Acknowledged.

[I-2] Float numbers are costly in terms of CU

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/lendr/src/events.rs

Description:

Floating-point operations on Solana are computationally expensive compared to integer arithmetic. For example, a single `f32` multiplication consumes over 22 times more compute units than an equivalent `u64` operation.

The protocol currently uses floating-point numbers primarily for event logging and health cache tracking, resulting in unnecessarily high CU spending.

Recommendations:

Consider refactoring the code to replace floating-point numbers with fixed-point arithmetic.

Pyron: Thanks to your issue, we reconsidered using these variables; They were only used to provide extra -unnecessary- statistical information in events and only were for logging purposes; so we decided to remove these f64 fields, instead of converting them to u64, which also requires extra multiplication/division operations -and cost- to convert them. So this one may not be a patch, but it completely eliminates the problem discussed in the issue.Commit: @a2595cc...

Zenith: Verified. Floating points field were removed entirely.

[I-3] Unused `decrease_balance` function

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/lendr/src/state/lendr_account.rs

Description:

The `decrease_balance` function is implemented in the codebase but never called or utilized in any operational flow, and adds unnecessary complexity to the codebase.

Recommendations:

Consider removing the function.

Pyron: Resolved with @8c04f77...

Zenith: Verified.

[I-4] Unused account in `LendrAccountSetAccountAuthority` context

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- transfer_authority.rs

Description:

In the `LendrAccountSetAccountAuthority` context, an unused account `fee_payer` is passed in and constrained as a Signer.

```
#[derive(Accounts)]
pub struct LendrAccountSetAccountAuthority<'info> {
    // ...
    #[account(mut)]
    pub fee_payer: Signer<'info>,
}
```

This wastes compute units, thereby increasing transaction costs.

Recommendations:

It is recommended to remove the unused `fee_payer` account.

Pyron: Resolved with @15b2423...

Zenith: Verified.