CODESPECT

LGT Adrena

SECURITY ASSESSMENT REPORT

November 7, 2025

Prepared for:

VOTEX

Contents

1 About CODESPECT

CODESPECT is a specialized smart contract security firm dedicated to ensure the safety, reliability, and success of blockchain projects. Our services include comprehensive smart contract audits, secure design and architecture consultancy, and smart contract development across leading blockchain platforms such as Ethereum (Solidity), Starknet (Cairo), and Solana (Rust).

At CODESPECT, we are committed to build secure, resilient blockchain infrastructures. We provide strategic guidance and technical expertise, working closely with our partners from concept development through deployment. Our team consists of blockchain security experts and seasoned engineers who apply the latest auditing and security methodologies to help prevent exploits and vulnerabilities in your smart contracts.

Smart Contract Auditing: Security is at the core of everything we do at CODESPECT. Our auditors conduct thorough security assessments of smart contracts written in Solidity, Cairo, and Rust, ensuring that they function as intended without vulnerabilities. We specialize in providing tailored security solutions for projects on EVM-compatible chains and Starknet. Our audit process is highly collaborative, keeping clients involved every step of the way to ensure transparency and security. Our team is also dedicated to cutting-edge research, ensuring that we stay ahead of emerging threats.

Secure Design & Architecture Consultancy: At CODESPECT, we believe that secure development begins at the design phase. Our consultancy services offer deep insights into secure smart contract architecture and blockchain system design, helping you build robust, secure, and scalable decentralized applications. Whether you're working with Ethereum, Starknet, or other blockchain platforms, our team helps you navigate the complexity of blockchain development with confidence.

Tailored Cybersecurity Solutions: CODESPECT offers specialized cybersecurity solutions designed to minimize risks associated with traditional attack vectors, such as phishing, social engineering, and Web2 vulnerabilities. Our solutions are crafted to address the unique security needs of blockchain-based applications, reducing exposure to attacks and ensuring that all aspects of the system are fortified.

With a focus on the intersection of security and innovation, CODESPECT strives to be a trusted partner for blockchain projects at every stage of development and for each aspect of security.

2 Disclaimer

Limitations of this Audit: This report is based solely on the materials and documentation provided to CODESPECT for the specific purpose of conducting the security review outlined in the Summary of Audit and Files. The findings presented in this report may not be comprehensive and may not identify all possible vulnerabilities. CODESPECT provides this review and report on an "as-is" and "as-available" basis. You acknowledge that your use of this report, including any associated services, products, protocols, platforms, content, and materials, is entirely at your own risk.

Inherent Risks of Blockchain Technology: Blockchain technology is still evolving and is inherently subject to unknown risks and vulnerabilities. This review focuses exclusively on the smart contract code provided and does not cover the compiler layer, underlying programming language elements beyond the reviewed code, or any other potential security risks that may exist outside of the code itself.

Purpose and Reliance of this Report: This report should not be viewed as an endorsement of any specific project or team, nor does it guarantee the absolute security of the audited smart contracts. Third parties should not rely on this report for any purpose, including making decisions related to investments or purchases.

Liability Disclaimer: To the maximum extent permitted by law, CODESPECT disclaims all liability for the contents of this report and any related services or products that arise from your use of it. This includes but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Third-Party Products and Services: CODESPECT does not warrant, endorse, or assume responsibility for any third-party products or services mentioned in this report, including any open-source or third-party software, code, libraries, materials, or information that may be linked to, referenced by, or accessible through this report. CODESPECT is not responsible for monitoring any transactions between you and third-party providers. We strongly recommend conducting thorough due diligence and exercising caution when engaging with third-party products or services, just as you would for any other product or service transaction.

Further Recommendations: We advise clients to schedule a re-audit after any significant changes to the codebase to ensure ongoing security and reduce the risk of newly introduced vulnerabilities. Additionally, we recommend implementing a bug bounty program to incentivize external developers and security researchers to identify and disclose potential vulnerabilities safely and responsibly.

Disclaimer of Advice: FOR AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, AND ANY ASSOCIATED SERVICES OR MATERIALS SHOULD NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.

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

Table 1: Risk Classification Matrix based on Likelihood and Impact

3.1 Impact

- **High** - Results in a substantial loss of assets (more than 10%) within the protocol or causes significant disruption to the majority of users.
- **Medium** - Losses affect less than 10% globally or impact only a portion of users, but are still considered unacceptable.
- **Low** - Losses may be inconvenient but are manageable, typically involving issues like griefing attacks that can be easily resolved or minor inefficiencies such as gas costs.

3.2 Likelihood

- **High** - Very likely to occur, either easy to exploit or difficult but highly incentivized.
- **Medium** - Likely only under certain conditions or moderately incentivized.
- **Low** - Unlikely unless specific conditions are met, or there is little-to-no incentive for exploitation.

3.3 Action Required for Severity Levels

- **Critical** - Must be addressed immediately if already deployed.
- **High** - Must be resolved before deployment (or urgently if already deployed).
- **Medium** - It is recommended to fix.
- **Low** - Can be fixed if desired but is not crucial.

In addition to High, Medium, and Low severity levels, CODESPECT utilizes two other categories for findings: **Informational** and **Best Practices**.

a) **Informational** findings do not pose a direct security risk but provide useful information the audit team wants to communicate formally.

b) **Best Practices** findings indicate that certain portions of the code deviate from established smart contract development standards.

4 Executive Summary

This document presents the security assessment conducted by CODESPECT for the LGT Solana program of Votex. Votex protocol suite is Solana's premier vote market. It transforms governance influence into a tradable commodity, creating a dynamic and efficient system for monetizing voting power and directing protocol incentives.

This audit focuses on the Adrena-Lock program, which is a liquid staking wrapper for the Adrena's ADX governance token.

The audit was performed using:

a) Manual analysis of the codebase.

b) Dynamic analysis of smart contracts, execution testing.

CODESPECT found seven points of attention, one classified as `High`, two classified as `Medium`, one classified as `Low`, one classified as `Info`, and two classified as `Best Practices`. All of the issues are summarised in Table 2.

Organisation of the document is as follows:

- **Section 5** summarizes the audit.
- **Section 6** describes the functionality of the code in scope.
- **Section 7** presents the issues.
- **Section 8** discusses the documentation provided by the client for this audit.
- **Section 9** presents the compilation and tests.

Issues found:

Severity	Unresolved	Fixed	Acknowledged
High	0	1	0
Medium	0	2	0
Low	0	1	0
Informational	0	1	0
Best Practices	0	2	0
Total	**0**	**7**	**0**

Table 2: Summary of Unresolved, Fixed, and Acknowledged Issues

5 Audit Summary

Audit Type	Security Review
Project Name	LGT Adrena
Type of Project	Liquid governance token
Duration of Engagement	8 Days
Duration of Fix Review Phase	1 Day
Draft Report	November 4, 2025
Final Report	November 7, 2025
Repository	lgt-adrena
Commit (Audit)	3eca6d0d04c5055da163ccfc83413ab8af10aef5
Commit (Final)	099d6029a9a8820f34eb604e327b63ccca62d4b3
Documentation Assessment	Medium
Test Suite Assessment	Medium
Auditors	jecikPo, ctrusonchain

Table 3: Summary of the Audit

5.1 Scope - Audited Files

	Contract	**LoC**
1	adrena-lock/src/instructions/adrena/upgrade_deposit.rs	261
2	adrena-lock/src/instructions/adrena/adrena_claim.rs	257
3	adrena-lock/src/instructions/adrena/initial_deposit.rs	206
4	adrena-lock/src/instructions/admin/init_alock.rs	140
5	adrena-lock/src/instructions/adrena/create_user.rs	139
6	adrena-lock/src/instructions/mint_alock.rs	113
7	adrena-lock/src/instructions/quarry/sync_reward_rate.rs	93
8	adrena-lock/src/instructions/set_governance_delegate.rs	76
9	adrena-lock/src/lib.rs	68
10	adrena-lock/src/programs.rs	36
11	adrena-lock/src/state.rs	36
12	adrena-lock/src/calcs.rs	33
13	adrena-lock/src/errors.rs	28
14	adrena-lock/src/seeds.rs	23
15	adrena-lock/src/instructions/admin/accept_operator.rs	21
16	adrena-lock/src/instructions/admin/accept_owner.rs	21
17	adrena-lock/src/instructions/admin/set_fee_destination.rs	20
18	adrena-lock/src/instructions/admin/set_platform_fee.rs	20
19	adrena-lock/src/instructions/admin/transfer_operator.rs	20
20	adrena-lock/src/instructions/admin/transfer_owner.rs	20
21	adrena-lock/src/instructions/mod.rs	19
22	adrena-lock/src/instructions/admin/mod.rs	7
23	adrena-lock/src/instructions/adrena/mod.rs	4
24	adrena-lock/src/instructions/quarry/mod.rs	1
	Total	**1662**

5.2 Findings Overview

	Finding	Severity	Update
1	Claim can locked by calling Adrena's `claim_stakes` directly	High	Fixed
2	Claiming Adrena rewards before the initial deposit will DoS the `alock` instance	Medium	Fixed
3	Quarry program's reward rate update is incorrect	Medium	Fixed
4	Only one `alock` instance can be created for a single Redeemer	Low	Fixed
5	The `operator` account is not used	Info	Fixed
6	Lack of Redeemer validation during initialization	Best Practices	Fixed
7	`alock` instance does not have pausing mechanism	Best Practices	Fixed

6 System Overview

The **Adrena-Lock** program is a Solana-based liquid staking wrapper protocol that enables users to deposit ADX tokens into locked staking positions within the Adrena protocol while receiving tradeable share tokens (xADX) in return. The protocol acts as an intermediary layer that manages collective staking operations, reward distribution, and governance participation on behalf of users.

Core State Account

The central state account that represents a collective locked staking position in the Adrena protocol is the `alock`. Each `alock` acts as a vault that aggregates user deposits and manages them as a single unified stake. The account is uniquely identified by its share mint address (xADX token mint).

Key responsibilities:

- Tracks the total amount of pending ADX deposits waiting to be staked (`pending_lm_deposits`)
- Maintains references to all associated token accounts (LM tokens, reward tokens)
- Stores configuration parameters including platform fees, fee destinations, and governance settings
- Manages ownership and operator roles for administrative operations
- Coordinates interactions with the Adrena staking protocol through its associated authorities

The `alock` operates through two PDA authorities:

- **Staker Authority**: Acts on behalf of the `alock` to execute staking operations in the Adrena protocol. This authority owns the staking token account and reward token accounts, enabling it to stake deposits and claim rewards.
- **Share Mint Authority**: A PDA with minting privileges for the xADX share tokens, allowing the protocol to mint shares in a 1:1 ratio with deposited ADX tokens during the mint operation.

Instruction Categories

Instructions can be grouped into two main categories: permissioned and permissionless.

Permissionless operations:

a. `init_alock`: Initializes the `alock` with initial configuration along with its associated authorities (`staker_authority`, `share_mint_authority`, `quarry_rate_setter`), initializes all necessary token accounts, and sets the initial platform fee.

b. `mint_alock`: Mints share tokens (xADX) to a depositor in exchange for LM tokens (ADX). Transfers ADX from the user to the `alock`'s token account, mints xADX 1:1 to the user, and increments the `pending_lm_deposits` counter.

c. `adrena_claim`: Claims rewards from Adrena staking and distributes them. Claims both ADX and USDC rewards, calculates and transfers platform fees to the fee destination, and sends remaining rewards to the respective Quarry redeemers for further distribution.

d. `sync_reward_rate`: Sets the reward distribution rate for a specific reward token. Checks the redeemer token account balance (ADX or USDC), calculates the annualized rate, updates the rate via Quarry operator, and sets the famine timestamp.

e. `initial_deposit`: Makes the first deposit into the Adrena staking system. Takes all tokens from `pending_lm_deposits`, stakes them into Adrena with a 540-day lock period, marks the vault as upgradeable, and resets `pending_lm_deposits` to 0.

f. `upgrade_deposit`: Upgrades the existing stake with new pending deposits. Takes all tokens from `pending_lm_deposits`, adds them to the existing Adrena stake, restarts the 540-day lock period, and resets `pending_lm_deposits` to 0.

Permissioned operations:

a. `create_user`: Creates a new user account for staking. Calls Adrena's `init_user_staking` instruction on behalf of the `staker_authority` to initialize the user staking account in the Adrena protocol.

b. `set_governance_delegate`: Sets a governance delegate for voting rights. Calls the governance program to set a delegate for the token owner record, allowing the delegate to vote on governance proposals on behalf of the `alock` instance.

c. `set_fee_destination`: Updates the destination account for platform fees. Sets the `platform_fee_destination` field in the Alock state to a new address where platform fees will be sent during reward claims.

d. set_platform_fee: Updates the platform fee percentage. Sets the platform_fee field.

e. transfer_operator: Owner initiates operator role transfer by setting a pending operator. Updates the pending_operator field in the alock state, requiring the new operator to accept the transfer.

f. transfer_owner: Owner initiates ownership transfer by setting a pending owner. Updates the pending_owner field in the alock state, requiring the new owner to accept the transfer.

g. accept_operator: Pending operator accepts operator role transfer.

h. accept_owner: Pending owner accepts ownership transfer from the current owner.

Key Protocol Flows

Protocol Initialization Sequence

This is the one-time setup sequence that must be executed when deploying a new alock instance:

Step 1: init_alock

- Creates the alock PDA account seeded by the share mint address
- Derives and stores the three authority PDAs: staker_authority, share_mint_authority, and quarry_rate_setter
- Initializes token accounts owned by staker_authority: lm_token_account (for ADX) and reward_token_account (for USDC)
- Creates redeemer token accounts for both LM (ADX) and reward (USDC) tokens
- Sets initial configuration: owner, operator, platform fee (15%), fee destination
- Sets is_stake_upgradable = 0 (not yet ready for upgrades)

Step 2: mint_alock (Initial Mint)

- First user(s) deposit ADX tokens into the protocol
- ADX tokens transferred from depositor to alock's lm_token_account
- xADX share tokens minted 1:1 to depositor
- pending_lm_deposits counter incremented by deposit amount
- Tokens remain in the alock's token account, not yet staked in Adrena

Step 3: create_user

- Owner calls this instruction to initialize the Adrena user staking account
- Creates user_staking PDA in the Adrena protocol for the staker_authority
- This account is required before the alock can stake tokens in Adrena
- Must be called before initial_deposit

Step 4: initial_deposit

- Takes all tokens from pending_lm_deposits and stakes them into Adrena
- Calls Adrena's add_locked_stake with a 540-day lock period
- Creates the first locked stake position in Adrena on behalf of staker_authority
- Mints governance tokens to the staker_authority for voting rights
- Sets is_stake_upgradable = 1 (vault is now operational)
- Resets pending_lm_deposits = 0

After initialization, the protocol is ready for normal operations (mint_alock, upgrade_deposit, adrena_claim, etc.).

Regular User Deposit Flow: mint_alock

Purpose: Users deposit ADX tokens and receive xADX share tokens

Steps:

- User approves transfer of ADX tokens from their wallet
- Protocol transfers ADX from user to alock's lm_token_account
- Protocol mints xADX share tokens to user's account (1:1 ratio)

– pending_lm_deposits counter incremented by deposit amount

Staking Pending Deposits: upgrade_deposit

Purpose: Move pending deposits from alock into the Adrena staking protocol

Steps:

- Takes all tokens from pending_lm_deposits
- Calls Adrena's upgrade_locked_stake to add tokens to the existing stake
- Restarts the 540-day lock period
- Resets pending_lm_deposits = 0

Reward Claiming and Distribution: adrena_claim

Purpose: Harvest staking rewards from Adrena and distribute them to the Quarry Redeemer.

Steps:

a. Calls Adrena's claim_stakes to claim both ADX and USDC rewards into lm_token_account and reward_token_account
b. Calculates platform fees based on platform_fee
c. Transfers platform fees (ADX and USDC) to platform_fee_destination
d. Transfers remaining rewards to Quarry Redeemer accounts (lm_redeemer_tokens and reward_redeemer_tokens)

After this, rewards are claimable by xADX holders through the Quarry protocol.

Reward Rate Synchronization: sync_reward_rate

Purpose: Updates the annual reward distribution rate in Quarry for either ADX or USDC (permissionless).

Steps:

a. Validates the correct Redeemer token account based on reward mint type (ADX or USDC)
b. Calculates annualized rate: current_redeemer_balance * 365 / 7
c. Calls Quarry Operator's delegate_set_annual_rewards to set the new rate
d. Calls Quarry Operator's delegate_set_famine with timestamp = now + 7 days

Should be called after adrena_claim to update reward distribution rates based on newly claimed rewards.

7 Issues

7.1 [High] Claim can locked by calling Adrena's `claim_stakes` directly

File(s): `adrena_claim.rs`

Description: The `adrena_claim` instruction is used to collect claims from Adrena's stake. It CPIs to `claim_stakes` to withdraw the rewards to the two accounts owned by the `staker_authority`: `lm_token_account` (ADX) and `reward_token_account` (USDC). Those two account keys are stored in `alock` for verification and they are ATAs as defined in the `init_alock` account context:

```
#[account(
    init,
    associated_token::mint = fee_redistribution_mint,
    associated_token::authority = staker_authority,
    payer = payer,
)]
pub reward_token_account: Box<InterfaceAccount<'info, TokenAccount>>,

#[account(
    init,
    associated_token::mint = lm_token_mint,
    associated_token::authority = staker_authority,
    payer = payer,
)]
pub lm_token_account: Box<InterfaceAccount<'info, TokenAccount>>,
```

The problem is that the `claim_stakes` instruction at Adrena is permissionless and the only condition there is that the two accounts for rewards have the authority set to the owner of the of the `user_stake`:

```
#[account(mut)]
pub owner: AccountInfo<'info>,

#[account(
    mut,
    token::mint = fee_redistribution_mint,
    has_one = owner
)]
pub reward_token_account: Box<Account<'info, TokenAccount>>,

#[account(
    mut,
    token::mint = cortex.load()?.lm_token_mint,
    has_one = owner
)]
pub lm_token_account: Box<Account<'info, TokenAccount>>,
```

The malicious user could call Adrena's `claim_stakes` directly on behalf of the `staker_authority` and provide two reward accounts that are non-ATAs, but still have the owner set as `staker_authority`, yet because those are different accounts than those stored at `alock`, the rewards will become stuck on them without the possibility of withdrawal by `adrena-lock` program.

Impact: A malicious user can lock all rewards accrued for the given `alock` repeatedly, especially by front running the legitimate `adrena_claim` calls.

Recommendation(s): Provide functionality (through a separate instruction) to transfer reward tokens from any `staker_authority` owned account to the valid ATA account. Another option is to remove the following validation:

```
require_keys_eq!(alock.lm_redeemer_tokens, self.lm_redeemer_tokens.key());
require_keys_eq!(
    alock.reward_redeemer_tokens,
    self.reward_redeemer_tokens.key()
);
```

And instead add Anchor constraint that the `staker_authority` is the owner of the two provided accounts. In case the provided `lm_redeemer_tokens` is equal to `alock.lm_redeemer_tokens` deduct the `alock.pending_lm_amount` to calculate the amount of rewards to distribute, otherwise take the account's `amount` directly.

Status: Fixed

7.2 [Medium] Claiming Adrena rewards before the initial deposit will DoS the `alock` instance

File(s): `adrena_claim.rs`

Description: The `adrena_claim` instruction is supposed to be called once the Adrena stake is initialized. The instruction firstly CPIs to Adrena's `claim_stakes` then transfers the collected rewards to respective accounts of the `quarry_redeemer` and `platform_fee_destination`. There is no validation guarding against calling `adrena_claim` before `initial_deposit` and inside the handler we can see that the stake state is marked as upgradable:

```
alock.is_stake_upgradable = 1;
```

Which will prevent calling the `initial_deposit` instruction. The requirement for this attack to succeed is that the Adrena's `user_stake` must be created, however this can be done by a malicious user, because it is a permissionless function and some small amount of deposit needs to be made to the `staker_authority` accounts to pass the following check:

```
// Amount of LM tokens that are pending rewards
let pending_lm_amount = lm_token_account_balance
    .checked_sub(self.alock.load()?.pending_lm_deposits)
    .ok_or(AdrenaLockError::Overflow)?;

// If there are no pending unclaimed tokens, return
if pending_lm_amount == 0 && pending_reward_amount == 0 {
    return Ok(());
}
```

After `alock.is_stake_upgradable` is set to one, it is not possible to call the `inital_deposit` and without the initialization of the stake it will not be possible to upgrade it with `upgrade_deposit` and the `alock` instance will be DoSed. The initial ADX deposit done through `mint_alock` will be lost.

Impact: DoS of the `alock` instance.

Recommendation(s): Remove the `alock.is_stake_upgradable = 1` statement. As a best practice it is recommended to ensure proper order initialization of the `alock` with flags: `init_alock` -> `mint_alock` -> `create_user` -> `initial_deposit`. Be ware however of the permissionless `init_user_staking` of Adrena, so the `create_user` should be idempotent.

Status: Fixed

7.3 [Medium] Quarry program's reward rate update is incorrect

File(s): `sync_reward_rate.rs`

Description: The `adrena_claim` call claims tokens to the `reward_redeemer_tokens` account, which is used to determine the reward rate in quarry program here. The Annual Reward rate at Quarry is used to control the amount of minted IOUs to the staking miners. The IOUs could then be used to redeem the rewards claimed using `adrena_claim` which are transferred to the Redeemer program (part of Quarry suite). The problem is that the Rate of IOU accrual is based on the amounts currently held at the Redeemer and does not take into account the already minted IOUs. It is possible that the Miners will be claiming IOUs but not withdrawing their claims from Redeemer, in such a scenario the amount of minted IOUs will be disproportionally larger than the available ADX and USDC rewards at the Redeemer

Impact: Too many IOUs will could be minted to Miners, hence those who claim at the Redeemer first will be at an advantage over the later claimants.

Recommendation(s): When calculating the Annual Rate and Famine values during `sync_reward_rate` take into account the pending rewards from the Redeemer.

Status: Fixed

7.4 [Low] Only one `alock` instance can be created for a single Redeemer

File(s): `init_alock.rs`

Description: During the `init_alock` two accounts are created for sending the rewards to the `quarry_redeemer` program. The accounts are defined as follows:

```
pub lm_redeemer: Box<Account<'info, quarry_redeemer::accounts::Redeemer>>,

#[account(
    init,
    associated_token::mint = lm_token_mint,
    associated_token::authority = lm_redeemer,
    payer = payer,
)]
pub lm_redeemer_tokens: Box<InterfaceAccount<'info, TokenAccount>>,

pub reward_redeemer: Box<Account<'info, quarry_redeemer::accounts::Redeemer>>,

#[account(
    init,
    associated_token::mint = fee_redistribution_mint,
    associated_token::authority = reward_redeemer,
    payer = payer,
)]
pub reward_redeemer_tokens: Box<InterfaceAccount<'info, TokenAccount>>,
```

Because they are defined as ATAs and have the `init` attribute they must not exist before the `init_alock` is called. This however cannot be guaranteed. Furthermore it is not possible to create second `alock` instance using the same Redeemer, which might be a desired requirement.

Impact: The `init_alock` can be DoSed and it is not possible to create more than one `alock` tied to the same Redeemer.

Recommendation(s): Use `init_if_needed` attribute instead of `init`

Status: Fixed

7.5 [Info] The `operator` account is not used

File(s): `init_alock.rs`

Description: The `init_alock` instruction populates the `alock.operator` with the provided `initial_operator` account:

```
alock.operator = self.initial_operator.key();
```

The protocol provides also the `transfer_operator` and `accept_operator` instructions to handle the operator handover. The operator account is however not required for any operation within the protocol hence it is unnecessary.

Impact: Unnecessary functionality

Recommendation(s): Remove the operator member from `alock`.

Status: Fixed

7.6 [Best Practice] Lack of Redeemer validation during initialization

File(s): init_alock.rs

Description: Two Redeemer owned accounts are used to transfer claimed rewards to the quarry_redeemer program:

```
pub lm_redeemer: Box<Account<'info, quarry_redeemer::accounts::Redeemer>>,

#[account(
    init,
    associated_token::mint = lm_token_mint,
    associated_token::authority = lm_redeemer,
    payer = payer,
)]
pub lm_redeemer_tokens: Box<InterfaceAccount<'info, TokenAccount>>,

/// The redeemer for the reward token.
pub reward_redeemer: Box<Account<'info, quarry_redeemer::accounts::Redeemer>>,

#[account(
    init,
    associated_token::mint = fee_redistribution_mint,
    associated_token::authority = reward_redeemer,
    payer = payer,
)]
pub reward_redeemer_tokens: Box<InterfaceAccount<'info, TokenAccount>>,
```

Those accounts are assigned to alock during init_alock and are stored there throughout the lifecycle of the alock instance. There is however no guarantee that the Redeemers are correct. Ideally they should be validated against their respective mint accounts. The Redeemer account holds the mint inside (from the quarry-redeemer program):

```
redeemer.redemption_mint = ctx.accounts.redemption_mint.key();
```

Impact: While this is just a best practice indication and the caller is responsible for the correct inputs, the consequences of mistake here will appear only later once someone tries to use the quarry-redeemer program, hence very late and the alock instance might be running already with substantial stake.

Recommendation(s): Add validation of the Redeemer data accounts so that they reflect correct reward mints.

Status: Fixed

7.7 [Best Practice] alock instance does not have pausing mechanism

File(s): adrena_claim.rs

Description: The alock instance lacks any emergency pause functionality, which means that any operations cannot be halted in case of security incidents, bugs, or other emergencies. The protocol allows continuous execution of all user-facing instructions including whenever transfer of assets is involved: minting xadx, claiming rewards, upgrading stake etc.

Impact: We can't pause protocol in case of any emergency.

Recommendation(s): Add a pausing mechanism. We recommend to add it for at least the adrena_claim instruction.

Status: Fixed

8 Evaluation of Provided Documentation

The **Adrena-Lock** documentation was provided in the form of a README file and NatSpec comments:

- **NatSpec:** The in-code NatSpec comments were generally sufficient and very helpful in explaining specific flows and code branches. In several cases, they also clarified assumptions underlying the implementation, providing context for why certain approaches were taken and the intended behavior of the system.
- **README:** The provided README offered a rather basic overview of the protocol's functionality.

Overall, the documentation was adequate for the scope of this audit, however it lacked of thorough explanation of the interaction with the Quarry program suite, which posed certain challenges in understanding the full token flow but the Votex team remained consistently available and responsive, promptly addressing all questions and concerns raised by **CODESPECT** throughout the audit process.

9 Test Suite Evaluation

9.1 Compilation Output

```
> anchor build
[...]
   Finished `test` profile [unoptimized + debuginfo] target(s) in 18.86s
     Running unittests src/lib.rs (039-LGT-Adrena/target/debug/deps/adrena_lock-2adee27f1c2997e3)
```

9.2 Tests Output

```
% anchor test
[...]
Found a 'test' script in the Anchor.toml. Running it as a test suite!

Running test suite: "039-LGT-Adrena/Anchor.toml"

yarn run v1.22.22
$ 039-LGT-Adrena/node_modules/.bin/ts-mocha -p ./tsconfig.json -t 1000000 'tests/**/*.ts'

[...]

  25 passing (31s)

  Done in 33.08s.

% cargo test

running 17 tests
test calcs::tests::test_calc_annualized_quarry_rewards_rate_weekly_to_annual_conversion ... ok
test calcs::tests::test_calc_annualized_quarry_rewards_rate_overflow ... ok
test calcs::tests::test_calc_fee_and_distribution_edge_case_rounding ... ok
test calcs::tests::test_calc_fee_and_distribution_near_max_values_with_small_fee ... ok
test calcs::tests::test_calc_fee_and_distribution_max_fee ... ok
test calcs::tests::test_calc_fee_and_distribution_normal_case ... ok
test calcs::tests::test_calc_fee_and_distribution_extremely_tiny_fees ... ok
test calcs::tests::test_calc_fee_and_distribution_one_basis_point ... ok
test calcs::tests::test_calc_fee_and_distribution_overflow_on_fee_calculation ... ok
test calcs::tests::test_calc_fee_and_distribution_small_amounts_with_fee_one ... ok
test calcs::tests::test_calc_fee_and_distribution_tiny_fee_boundary ... ok
test calcs::tests::test_calc_fee_and_distribution_various_tiny_fees ... ok
test calcs::tests::test_calc_fee_and_distribution_zero_fee ... ok
test calcs::tests::test_calc_functions_all_zero ... ok
test calcs::tests::test_calc_functions_asymmetric_amounts ... ok
test calcs::tests::test_calc_functions_max_safe_values ... ok
test test_id ... ok

test result: ok. 17 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

   Doc-tests adrena_lock

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

9.3 Notes on the Test Suite

The Adrena-Lock test suite provides comprehensive coverage for the protocol's primary flows and addresses most common usage scenarios. It is worth noting that that the test suite included tests involving time warping, which allows to correctly validate reward accruals.

While the existing tests are valuable, the primary area for improvement is the end to end reward flow involving the Quarry Rewarder and Redeemer programs, as the issue discovered was not covered.