Chain-Fox Stake Smart Contract Audit

Overview

- Github Link: https://github.com/Chain-Fox/chain-fox-stake
- Commit: 00f6621982e5311f238849e3ac567a5fa482297c
- Auditied by Chain-Fox
- Audit Date: 2025-06-22

System Accounts (Always Present)

- `system_program`: Program Account (System Program - `11111111111111111111111111111111`)
- `token_program`: Program Account (Token Program - `TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA`)
- `rent`: Sysvar Account (Rent Sysvar - `SysvarRent111111111111111111111111111111111`)

Core Program Accounts

1. **Stake Pool PDA**
    - Type: PDA Account (created in contract)
    - Seeds: `["stake_pool", token_mint.key()]`
    - Used in: All instructions
    - Contains: Pool configuration and state
2. **Token Vault**
    - Type: Token Account (created in contract)
    - Authority: Stake Pool PDA
    - Used in: Initialize, Stake, Withdraw, AdminWithdraw
3. **Multisig Config PDA**
    - Type: PDA Account (created in contract)
    - Seeds: `["multisig_config", stake_pool.key()]`
    - Used in: Multisig-related instructions
4. **Proposal PDA**

- Type: PDA Account (created in contract)
- Seeds: `["proposal", multisig_config.key(), proposal_count]`
- Used in: Proposal-related instructions

User Accounts

1. **User Stake PDA**
   - Type: PDA Account (created in contract)
   - Seeds: `["user_stake", stake_pool.key(), owner.key()]`
   - Used in: CreateUserStake, Stake, RequestWithdrawal, Withdraw
   - Contains: Individual user's stake information
2. **User Token Account**
   - Type: Token Account (external)
   - Authority: User Wallet
   - Used in: Stake, Withdraw
   - Must match stake pool's token mint
3. **Owner/User**
   - Type: Wallet Account (Signer)
   - Used in: User-related operations
   - Must sign for user-initiated transactions

Authority Accounts

1. **Authority**
   - Type: Wallet Account (Signer)
   - Used in: Initialize, InitializeMultisig, TogglePause
   - Original pool administrator
2. **Stake Pool Authority**
   - Type: PDA Account (derived from stake pool)
   - Seeds: Same as stake pool PDA
   - Used as token vault authority
3. **Multisig Signers**
   - Type: Wallet Accounts (Signers when needed)
   - Used in: CreateProposal, SignProposal
   - Defined in MultisigConfig

Special Case Accounts

1. **Token Mint**
   - Type: Token Mint Account (external)
   - Used in: Initialize

- Defines the staked token type

2. **Recipient Token Account**
    - Type: Token Account (external)
    - Used in: ExecuteAdminWithdraw
    - Receives admin-withdrawn tokens

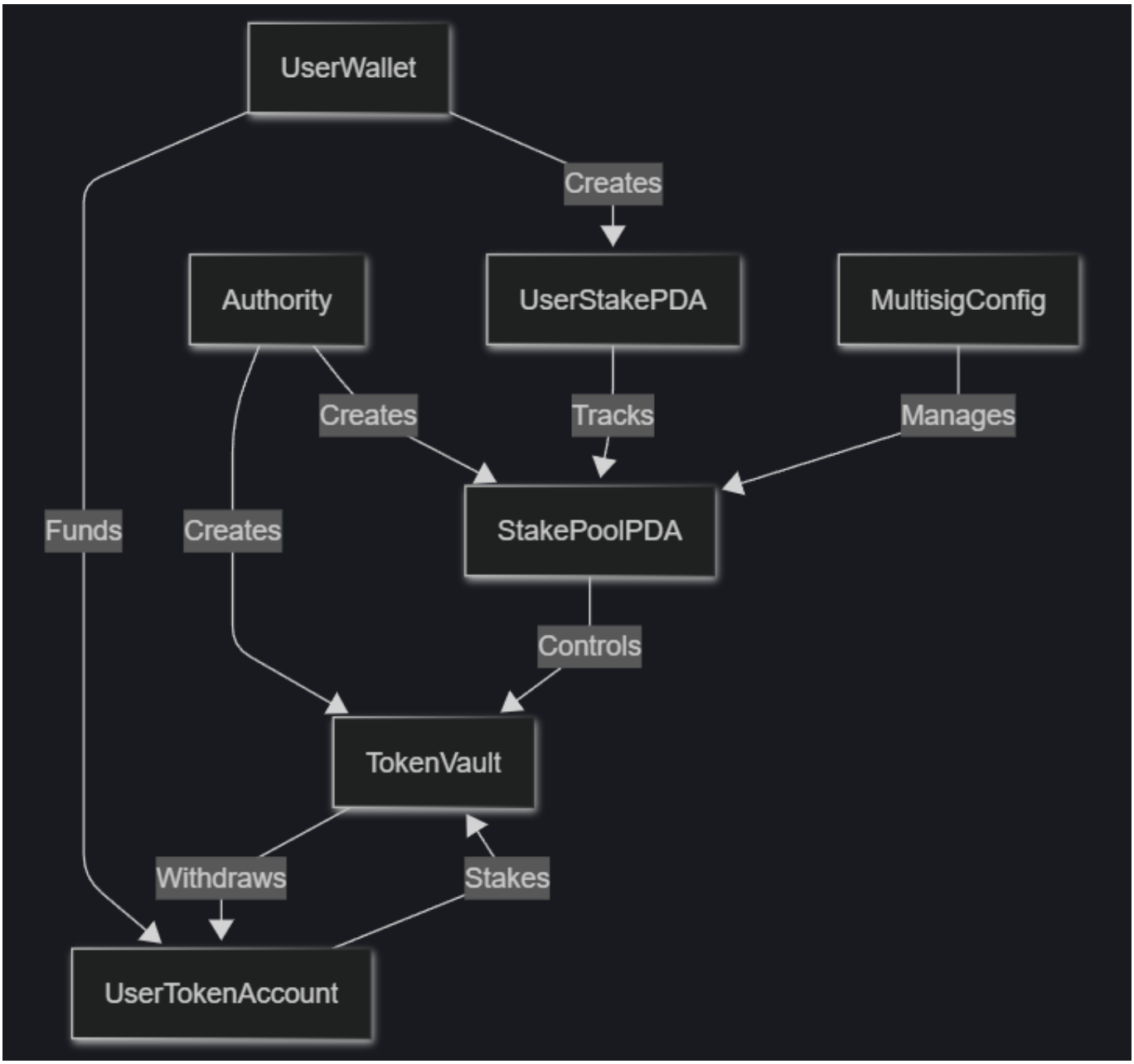
Account Creation Summary

Account Type	Created in Contract	Initialized By
Stake Pool PDA	Yes	Initialize
Token Vault	Yes	Initialize
Multisig Config PDA	Yes	InitializeMultisig
Proposal PDA	Yes	CreateProposal
User Stake PDA	Yes	CreateUserStake

Signer Requirements

- **Must be Signer**: `authority`, `owner`, `proposer`, `signer`, `executor`
- **Never Signer**: PDAs, token accounts, mint accounts
- **Conditional Signer**: Multisig signers (only when signing proposals)

Account Relationships



This structure shows how privileges flow through the system, with PDAs controlling key functions while requiring proper authorization from either user wallets or multisig signers for sensitive operations.

Priviledge checking

1. The same **AuthorityWallet** initializes **StakePoolPDA** and **MultiSigConfigPDA**
2. **MultiSigPDA** contains 3 pubkeys as multisig signers
3. Any signer in the multisig signers can create a **ProposalPDA** and immediately sign it
4. Any signer in the multisig signers can sign a **ProposalPDA** (no duplicate signing)
5. **ProposalPDA** is approved only after 2 in 3 signers having signed the **ProposalPDA**
6. Any **ExecutorWallet** can execute **ProposalPDA** that has been approved to
    - toggle **StakePoolPDA**'s emergency mode
    - update **StakePoolPDA**'s authority
    - withdraw amount of tokens from **TokenVault** to the **ProposalPDA**'s recipient
7. A **UserWallet** initializes **UserStakePDA** and can authorize it.
8. The authorized **UserWallet** can stake: transfer tokens from **UserTokenAccount** owned by **UserWallet** to **TokenVault**, record the increment in **UserStakePDA** and **StakePoolPDA**, record the staking time slot

in **UserStake**, and reset withdrawal request
9. The authorized **UserWallet** can request withdrawal and set withdrawel request
    - on emergency mode of **StakePoolPDA**, the stake can be unlocked now
    - on non-emerency mode of **StakePoolPDA**, the stake can only be unlocked after ~30 days in slots
10. The authorized **UserWallet** can withdraw when the stake has been unlocked: transfer all the staked tokens from **TokenVault** to the **UserTokenAccount** owned by **UserWallet**, record the decrement in **UserStakePDA** and reset the staked_amount in **StakePoolPDA**, staking time slot in **UserStake**, and withdrawal request
11. The **AuthorityWallet** can toggle the emergency mode of **StakePoolPDA**
12. Close instructions are not implemented for the PDAs to prevent accidental close during staking or approving.

Conclusion: The privilege checking of ownership, authority, PDA bumps, instruction types, close, multisig, time slot is safely implemented and no missing check is found.

Financial checking

1. The total amount of tokens transfered from **UserTokenAccount** to **TokenVault** always equals to the staked amount recorded in **UserStakePDA**, where **UserTokenAccount** and **UserStakePDA** are authorized or owned by the same **UserWallet**.
2. The staked amount recorded in **StakePoolPDA** always equals to the total staked amount recorded in all of the **UserStakePDA**s.
3. The amount of tokens in **TokenVault** is always larger than or equals to the staked amount recorded in **StakePoolPDA** plus the total amount of multisig withdrawal.
4. The user cannot withdraw the staked tokens before the stake is unlocked.
5. The user can always withdraw the staked tokens when the stake is unlocked assuming enough tokens reside in **TokenVault** before withdrawal.
6. Repeated staking: Every stake will reset the unlock time and withdrawal request. e.g. If users stake and request withdrawal on time slot T1, and they stake again on time slot T2 before withdraw, then they have to request withdrawal again and can only withdraw after at least T2+30 days.

Conclusion: It is impossible to perform a financial attack on a user's stake account without the user's permission. It is impossible to perform a financial attack on the stake pool and token vaults without multisig's permission.

Compliance checking

Compliance rules are extracted from the specification defined in README.md

1. Security Features

- ☑ Slot-based Timing: Uses Solana slots rather than timestamps for enhanced security: *All the time measurement throughout the contract is based on SLOT.*
- ☑ Emergency Pause: Multi-signature can pause new staking operations in emergencies: `L255` `require!(!stake_pool.emergency_mode, StakeError::ContractPaused);`
- ☑ Unified Contract Vault: All user funds stored in single contract-controlled token vault: *`L792-L710` authority creates token_vault *

- ☑ Individual User Records: Each user has their own UserStake PDA recording staking information: `L798-L806 user creates and authorizes user_stake PDA`
- ☑ User Fund Protection: Only users themselves can deposit and withdraw, administrators cannot access user funds even in emergencies: *see stake and withdraw*
- ☑ Multi-signature Management: 3-wallet multi-signature mechanism for all critical administrator operations: *see the usage of multisig_config*
- ☑ Reentrancy Attack Protection: Guards against reentrancy attacks in critical functions: *see reentrancy_guard*
- ☑ Staking Limits: Maximum individual stake (10 million CFX) and maximum total pool size (900 million CFX) `L19 const MAX_POOL_SIZE: u64 = 900_000_000 * 1_000_000; // 900,000,000 CFX`
- ☑ Time Range Checks: Lock periods cannot exceed 1 year: `L48 const MAX_LOCK_DURATION_SLOTS: u64 = 365 * SLOTS_PER_DAY; // 1 year in slots`
- ☑ Arithmetic Safety: All calculations include overflow protection: *Verified by Arithmetic Checking*
- ☑ Administrator Withdrawal Control: Administrators can only withdraw CFX from contract's token vault through AdminWithdraw multi-sig proposals: *Verfied by Priviledge-Checking*
- ☑ User Fund Protection: Contract tracks total_staked to ensure administrator withdrawals cannot access user staked funds: *Verified by Priviledge-Checking*

2. Permission Control Mechanisms

2.1. User-exclusive Permissions

Verified by Priviledge-Checking

- ☑ Staking Operations: Only users themselves can stake CFX tokens
- ☑ Withdrawal Requests: Only users themselves can request withdrawal of their own stakes
- ☑ Fund Withdrawal: Only users themselves can withdraw their own staked funds
- ☑ Administrator Permission Limitations
- ☑ Administrators cannot perform the following operations:
- ☑ Cannot withdraw user staked funds from contract vault without user consent
- ☑ Cannot operate user staking accounts or modify user staking records
- ☑ Cannot bypass user signatures for any user fund operations
- ☑ Cannot access user funds even in emergency situations

2.2. Administrators can only perform the following operations

Verified by Priviledge-Checking

- ☑ Toggle emergency mode (through multi-sig proposals)
- ☑ Update contract permissions (through multi-sig proposals)
- ☑ Withdraw CFX from contract's token vault (through AdminWithdraw multi-sig proposals)
- ☑ Update multi-signature configuration (through multi-sig proposals)
- ☑ Important Note: Administrator withdrawals from the token vault are separate from user staked funds. The contract tracks total_staked to ensure user funds are protected, and administrator withdrawals can only access excess funds in the vault: Note the authority cannot withdraw from the token_vault. Multisig is allowed to withdraw from the token_vault but only to the approved account. Enough tokens will always be kept in token_account by the multisig verifier.

2.3. Technical Implementation

Verified by Priviledge-Checking

- ☑ Account Binding: Each user staking account is bound to specific users through PDA seeds
- ☑ Signature Verification: All user operations require the user's own digital signature Ownership Checks: Contract verifies that operators are the true owners of accounts
- ☑ Multi-signature Management: The contract uses a 3-wallet multi-signature mechanism to enhance the security of administrator operations.

2.4. Multi-signature Security Advantages

Verified by Priviledge-Checking

- ☑ No Single Point of Failure: Critical operations require multiple signatures
- ☑ Transparent Governance: All proposals are recorded on-chain
- ☑ Flexible Threshold: Configurable (e.g., 2/3, 3/3)
- ☑ Audit Trail: Complete history of all administrator operations
- ☑ Emergency Response: Multiple parties can respond to security incidents

Arithmetic checking

Overflow, underflow, rounding issues

All the arithmetic operations throughout the contract are checked.

Note that the following two places contain unchecked arithmetic operations but no security issues are found.

In `create_proposal`:

```
//programs/cfx-stake-core/src/lib.rs#L137
multisig_config.proposal_count += 1;
```

proposal_count is of type u64. To overflow signers need to propose 2**64 proposals, which cannot occur in reality.

In `impl StakePool`:

```
//programs/cfx-stake-core/src/lib.rs#508-529
// Utility functions for slot-time conversion
impl StakePool {
    /// Convert slots to approximate seconds
    pub fn slots_to_seconds(slots: u64) -> u64 {
        slots * AVERAGE_SLOT_TIME_MS / 1000
    }

    /// Convert seconds to approximate slots
    pub fn seconds_to_slots(seconds: u64) -> u64 {
        seconds * 1000 / AVERAGE_SLOT_TIME_MS
```

```
    }

    /// Convert slots to approximate days
    pub fn slots_to_days(slots: u64) -> f64 {
        slots as f64 / SLOTS_PER_DAY as f64
    }

    /// Convert days to slots
    pub fn days_to_slots(days: u64) -> u64 {
        days * SLOTS_PER_DAY
    }
}
```

1. They are util funcs that are not called in the contract and arithmetic issues in them cannot cause any effect in the contract logic.
2. u64, f64 is large enough counter overflow, precision loss, etc.
3. A valid time slot is within bounds that cannot overflow.

Improvement

Minor Issues

1. One stake pool for each token_mint

```
//programs/cfx-stake-core/src/lib.rs#L697C20-L697C66
seeds = [b"stake_pool".as_ref(), token_mint.key().as_ref()],
```

Note that only one stake_pool can be created for each token_mint. Cannot have stake_pools for the same token_mint with different lock periods. It is okay for now and will not cause any security issues.

2. Document improvement

In README.md#staking-limits-boundary-protection

`Maximum Total Pool Size 400,000,000 CFX`: Should change to 900,000,000 CFX by implementation.

3. Previous audit results

See https://github.com/Eason748/chain-fox-stake/pull/1