Cyberscope

Audit Report

BrushO Network

September 2024

Repository	https://github.com/aitoothbrush/brusho-program-library
Commit	d25489ea7dd8077ecfd6e0971e079317efaea231
Audited by	© cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Repository	https://github.com/aitoothbrush/brusho-program-library
Commit	d25489ea7dd8077ecfd6e0971e079317efaea231
Network	SOL

Audit Updates

Initial Audit	04 Sept 2024

Source Files

Filename	SHA256
circuit-breaker/src/errors.rs	8280c1f88face5f4f37363aa2cc0e09ca8b3df9b46da9cc90cddc0d368edcf27
circuit-breaker/src/lib.rs	e2eea618ed4431b40b40b1a93f4c2d26ac96ef119790a4ba152c7a8e6ecd268f
circuit-breaker/src/state.rs	e3ad7494caf0075b3b0804ab39bf7bd84f44465fc3aee4d6d294340a829355e5
circuit-breaker/src/window.rs	8b9ef0f104b465ba7a7c7f8ede235e9a031186a45ca33e8e31d968bc62258fc0
circuit-breaker/src/instructions/initialize_account_windowed_breaker_v0.rs	429990a392c6b6bb180b814a47c8d4f68ff9b4040ed5b37bcb136480b66b6975
circuit-breaker/src/instructions/mod.rs	4fac2af9cccad8d5423ad18e4aaa30833d0feef0f7fc9f5422202021025b3f2a
circuit-breaker/src/instructions/transfer_v0.rs	2d386e7a33ca8554707a943163e010f58227a9a746b9a316a4f0723e12bd7c07

circuit-breaker/src/instructions/update_account_windowed_breaker_v0.rs	39173bb68d19edf8e0a1f6b27afcfa80cce322c4a9ead11c0228bf0614d2d14b
voter-stake-registry/src/error.rs	06866c2e249af598967f7f4c05d5746091111097ff971ff5484f6aa389c9a0e5
voter-stake-registry/src/governance.rs	9da7da2eb636de9619e430970d91d399a8fa6d65710ee7a10e7056de66375b91
voter-stake-registry/src/lib.rs	e2765e541cca4087b58c44bb79a4b20c91502513eac5759ad56910d8c34364ba
voter-stake-registry/src/events/mod.rs	8af40f61b16a7edcd9fa535b8e22b78442422bea6a5336973ee44c975b4588d8
voter-stake-registry/src/instructions/claim_reward.rs	dcead9c9d8a14ed1d430e93c2413350c5023b556c6b66e7fc0d358395716064e
voter-stake-registry/src/instructions/close_voter.rs	c6fa26978874ed313e64e15752a7ee227aa7168b5ac81e45d453e767593834d3
voter-stake-registry/src/instructions/create_registrar.rs	ebf6bbc6bd8da23c08df6e11f25a2a00c42e0c0764338e0a702ca14718b9a783
voter-stake-registry/src/instructions/create_voter.rs	db7b99a0e27eea02ba7b9cd7ae8f386f88387baa35dc480dd7e922af57689912
voter-stake-registry/src/instructions/log_voter_info.rs	89150c509b62e9900c8ddf8495e878e165e5cc7e0b3b8c100520da9bcc442b09
voter-stake-registry/src/instructions/mod.rs	9df9d46d8edd5751366e8ab10300363950c6bbffc2c134537fe16044618996e0

voter-stake-registry/src/instructions/node_deposit.rs	409457c6a40e69059f47acad0306cadc7f7933e00b30b185bbf01df472576b04
voter-stake-registry/src/instructions/node_release_deposit.rs	fc3ba8b00b6eae2c5d65e2d6d7dc5514b78bf1ad444248a0f5ab49408b9f6df9
voter-stake-registry/src/instructions/ordinary_deposit.rs	c9fde79aeb435ec84bf083179283e0393460ac777efdab1ad290147e7aeea793
voter-stake-registry/src/instructions/ordinary_release_deposit.rs	8cdea3d20d95e058d12111bcea0d0b11b593ac73c69b43853e7ffcdad550795e
voter-stake-registry/src/instructions/set_time_offset.rs	8a8a5b8a788219a755d5228001bf8e24b2f8fd4774718391a21b0d336b736802
voter-stake-registry/src/instructions/update_deposit_config.rs	bd1f922492594b019446ddafa8e2d0921879705f15dc03a93a805e8519915408
voter-stake-registry/src/instructions/update_max_vote_weight.rs	024809ce30bb6f3e86c0c58a7c2c633a1f63ca0b88db4fb7a8358c0943ddbfa5
voter-stake-registry/src/instructions/update_voter_weight_record.rs	347ff438ceadbdf5879e9d16d90a86d256c61efd806126dd6633b6c6768bbb8c
voter-stake-registry/src/instructions/update_voting_config.rs	f2d7026530bfdf08bfa6ecda59bb87c4da50bec18d7f40d743d2a8f833d90b53
voter-stake-registry/src/instructions/withdraw.rs	df7c8a03278f7377f9891d6fd1e62574083b543c3c20f6cdca55f94bf6095ee4

voter-stake-registry/src/state/deposit_entry.rs	2df1bb3c1d91debe912b8c69e0c7269561a23a1b8a9f9d111fb547e810a28ba0
voter-stake-registry/src/state/lockup.rs	8402f8c4f23888f29d4789992b4e0dbd1f7ce62dbb75dc7ecfa714886380f613
voter-stake-registry/src/state/mod.rs	4d0269d8c496a4fd787a223d499a104b7f5202de7b6adc650c1b424b3b55bf75
voter-stake-registry/src/state/registrar.rs	0d93cb9b394533dae2ec696e47750928f71cb1d207e9fb6df2cbec674a7dc501
voter-stake-registry/src/state/voter.rs	2a9c5cc4183915b31e5416b1bf21aa420836cf4fc9e687330d1b62807887d1fb

Overview

Circuit Breaker Overview

The Circuit Breaker enforces transfer limits on token accounts within defined time windows on the Solana blockchain. Its primary function is to prevent excessive or rapid token movements that could destabilize the system.

Upon initialization, the contract assumes control of a token account's authority and sets parameters such as the time window size and transfer thresholds. During each transfer, the contract checks if the cumulative value of transactions within the current window exceeds the set limit. If it does, the transfer is blocked.

The contract also allows for updates to its configuration or authority, providing flexibility to adapt to changing operational needs. This system is particularly useful in scenarios where managing token transfer rates is essential to maintaining stability.

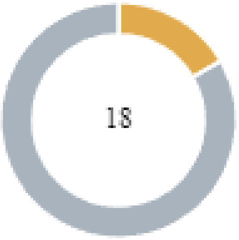
Voter Stake Registry Overview

The Voter Stake Registry is a decentralized governance framework designed to manage voting rights, reward distribution, and token deposits for participants within a governance system. The system is built around a registrar that oversees the core functionalities, including the creation and management of voter accounts, reward claims, and security of token deposits. It allows users to participate in governance decisions by staking tokens, which in turn contributes to their voting power.

Key components of the system include voter accounts, deposit entries, and voter weight records. Voter accounts hold tokens deposited by users, and these deposits are subject to configurable lockup periods, determining when tokens can be withdrawn or reassigned. Voter weight records are continuously updated to accurately reflect a user's influence in governance decisions based on the current state of their deposits and rewards.

The system also integrates various safeguards, such as circuit breakers, to ensure the secure management of token transfers and prevent unauthorized actions. Additionally, administrative controls are provided to realm authorities, allowing them to update deposit and voting configurations, ensuring flexibility and adaptability within the governance structure. Event logging mechanisms are in place to maintain transparency throughout the system, with critical operations like deposits, withdrawals, and account closures being recorded for auditing purposes.

Findings Breakdown



● Critical	0
● Medium	3
● Minor / Informative	15

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	3	0	0	0
● Minor / Informative	15	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ⚪ Minor / Informative

Severity	Code	Description	Status
🟠	IRC	Inconsistent Reward Calculations	Unresolved
🟠	MMZ	Missing Memory Zeroing	Unresolved
🟠	PVPM	Potential Voting Power Manipulation	Unresolved
⚪	DEMA	Descriptive Error Messages Absence	Unresolved
⚪	ICEH	Inadequate Clock Error Handing	Unresolved
⚪	IC	Incomplete Contract	Unresolved
⚪	IBU	Inefficient Box<Account> Usage	Unresolved
⚪	IVUR	Inefficient Variable Usage Repetition	Unresolved
⚪	MEE	Missing Events Emission	Unresolved
⚪	PCR	Program Centralization Risk	Unresolved
⚪	RB	Redundant Borrowing	Unresolved
⚪	RFN	Redundant Field Names	Unresolved
⚪	RMB	Redundant Mutable Borrowing	Unresolved
⚪	RCR	Repeated Constraints Reuse	Unresolved

●	TE	Typographical Errors	Unresolved
●	UC	Unformatted Code	Unresolved
●	UTC	Unnecessary Type Casting	Unresolved
●	VAUD	Vulnerabilities and Unmaintained Dependencies	Unresolved

IRC - Inconsistent Reward Calculations

Criticality	Medium
Location	voter-stake-registry/src/state/registrar.rs#L10,12
Status	Unresolved

Description

The contract defines two constants, `TOTAL_REWARD_AMOUNT` and `FULL_REWARD_PERMANENTLY_LOCKED_FLOOR`, which are intended to represent token amounts for reward distribution. However, there is an inconsistency between the numerical values and the comments describing them. `TOTAL_REWARD_AMOUNT` is stated to represent 7.7 billion tokens, or 770,000 tokens when accounting for 6 decimal places. On the other hand, `FULL_REWARD_PERMANENTLY_LOCKED_FLOOR` is described as 195 million tokens, and the value provided aligns with 195 million tokens when calculated using 6 decimal places. The issue arises because `TOTAL_REWARD_AMOUNT` only corresponds to 770 million tokens, not the intended 7.7 billion. This suggests there is a miscalculation in either the values or the comments.

```
pub const TOTAL_REWARD_AMOUNT: u64 = 770_000_000_000_000; //
7.7b
pub const FULL_REWARD_PERMANENTLY_LOCKED_FLOOR: u64 =
195_000_000_000_000; // 195M
```

Recommendation

To resolve this inconsistency, the calculations for both `TOTAL_REWARD_AMOUNT` and `FULL_REWARD_PERMANENTLY_LOCKED_FLOOR` should be carefully reviewed. Ensure that the values and their corresponding comments accurately reflect the intended token amounts. Verifying these constants is essential to ensure accurate reward distribution and maintain the integrity of the reward mechanism.

MMZ - Missing Memory Zeroing

Criticality	Medium
Location	voter-stake-registry/src/instructions/close_voter.rs#L36
Status	Unresolved

Description

The `close_voter` instruction fails to properly zero out the voter account before closing it. This issue leaves the protocol vulnerable to potential reinit attacks, where an attacker could reuse the same account with the same seeds, potentially leading to unintended behavior or exploitation. Without zeroing out the account, the data persists in memory, which increases the risk of unauthorized reinitialization.

```
pub fn close_voter<'info>(ctx: Context<'_, '_, 'info, 'info,
CloseVoter<'info>>) -> Result<()> {
    {
        let voter = &ctx.accounts.voter;
        let amount = voter.amount_deposited_native();
        require_eq!(amount, 0,
VsrError::GoverningTokenNonZero);
        require_eq!(voter.get_reward_claimable_amount(), 0,
VsrError::GoverningTokenNonZero);

        // let voter_seeds = voter_seeds!(voter);
        // let voter_seeds =
        for account in ctx.remaining_accounts.iter() {
            let token =
Account::<TokenAccount>::try_from(&account).unwrap();
            require_keys_eq!(
                token.owner,
                ctx.accounts.voter.key(),
                VsrError::InvalidAuthority
            );
            require_eq!(token.amount, 0,
VsrError::VaultTokenNonZero);

            let cpi_accounts = CloseAccount {
                account: account.to_account_info(),
                destination:
ctx.accounts.sol_destination.to_account_info(),
                authority:
ctx.accounts.voter.to_account_info(),
            };
            let cpi_program =
ctx.accounts.token_program.to_account_info();
            token::close_account(CpiContext::new_with_signer(
                cpi_program,
                cpi_accounts,
                &[voter_seeds!(voter)],
            ))?;

            account.exit(ctx.program_id)?;
        }
    }

    Ok(())
}
```

Recommendation

The function should ensure that the voter account data is thoroughly cleared upon closure. Implement a mechanism to zero out all memory associated with the voter account, effectively preventing it from being reused with the same seeds in future operations. This would significantly reduce the risk of reinit attacks and strengthen the security of the protocol.

PVPM - Potential Voting Power Manipulation

Criticality	Medium
Location	voter-stake-registry/src/instructions/ordinary_release_deposit.rs#L25 voter-stake-registry/src/instructions/node_release_deposit.rs#L25
Status	Unresolved

Description

The system allows users to transfer deposits between different deposit entries, which may create an opportunity for manipulating voting power. Although the system recalculates the voter's weight before each vote, there is no mechanism to prevent a user from transferring their tokens between deposit entries. This could allow users to influence governance decisions disproportionately by reallocating deposits across multiple indexes without sufficient restrictions. The lack of control over when and how tokens can be moved between deposits creates the potential for users to artificially boost their voting power across multiple votes, undermining the fairness and integrity of the governance process.

```
pub fn node_release_deposit(
    ctx: Context<NodeReleaseDeposit>,
    target_deposit_entry_index: u8,
) -> Result<()> {
    let registrar = &mut ctx.accounts.registrar;
    let voter = &mut ctx.accounts.voter;

    let d_entry = voter.deposit_entry_at(NODE_DEPOSIT_ENTRY_INDEX)?;
    require!(d_entry.is_active(), VsrError::InactiveDepositEntry);
    require!(
        !voter.is_active(target_deposit_entry_index)?,
        VsrError::ActiveDepositEntryIndex
    );

    // accure rewards
    let curr_ts = registrar.clock_unix_timestamp();
    registrar.accure_rewards(curr_ts);

    let node_security_deposit = d_entry.get_amount_deposited_native();
    let lockup = d_entry.get_lockup();
    if let LockupKind::Constant(duration) = lockup.kind() {
        if curr_ts < lockup.end_ts() {
            return
Err(error!(VsrError::NodeDepositUnreleasableAtPresent));
        }

        voter.deactivate(NODE_DEPOSIT_ENTRY_INDEX, curr_ts,
registrar)?;

        let target_lockup = Lockup::new_from_duration(duration,
curr_ts, curr_ts)?;

        voter.activate(
            target_deposit_entry_index,
            curr_ts,
            target_lockup,
            registrar,
        )?;
        voter.deposit(
            target_deposit_entry_index,
            curr_ts,
            node_security_deposit,
            registrar,
        )?;

        emit!(NodeReleaseDepositEvent {
            voter: voter.get_voter_authority(),
            target_deposit_entry_index,
        });
        Ok(())
    } else {
        Err(error!(VsrError::InternalProgramError))
    }
}

pub fn ordinary_release_deposit(
    ctx: Context<OrdinaryReleaseDeposit>,
    deposit_entry_index: u8,
    target_deposit_entry_index: u8,
    amount: u64,
) -> Result<()> {
    require!(amount > 0, VsrError::ZeroAmount);
    require!(
        deposit_entry_index != NODE_DEPOSIT_ENTRY_INDEX
            && target_deposit_entry_index != NODE_DEPOSIT_ENTRY_INDEX,
        VsrError::NodeDepositReservedEntryIndex
    );

    let registrar = &mut ctx.accounts.registrar;
    let voter = &mut ctx.accounts.voter;

    let d_entry = voter.deposit_entry_at(deposit_entry_index)?;
    require!(d_entry.is_active(), VsrError::InactiveDepositEntry);

    let curr_ts = registrar.clock_unix_timestamp();
    // accure rewards
    registrar.accure_rewards(curr_ts);

    let lockup = d_entry.get_lockup();
    if let LockupKind::Constant(duration) = lockup.kind() {
        let amount_deposited_native =
d_entry.get_amount_deposited_native();
        require_gte!(
            amount_deposited_native,
            amount,
            VsrError::InsufficientLockedTokens
        );

        voter.deactivate(deposit_entry_index, curr_ts, registrar)?;
        if amount_deposited_native > amount {
            voter.activate(deposit_entry_index, curr_ts, lockup,
registrar)?;
            voter.deposit(
                deposit_entry_index,
                curr_ts,
                amount_deposited_native - amount,
                registrar,
            )?;
        }
        require!(
            !voter.is_active(target_deposit_entry_index)?,
            VsrError::ActiveDepositEntryIndex
        );

        let target_lockup = Lockup::new_from_duration(duration,
curr_ts, curr_ts)?;

        voter.activate(target_deposit_entry_index, curr_ts,
target_lockup, registrar)?;
        voter.deposit(target_deposit_entry_index, curr_ts, amount,
registrar)?;

        emit!(OrdinaryReleaseDepositEvent {
            voter: voter.get_voter_authority(),
            deposit_entry_index,
            target_deposit_entry_index,
            amount,
        });

        Ok(())
    } else {
        Err(error!(VsrError::NotOrdinaryDepositEntry))
    }
}
```

Recommendation

It is recommended to implement restrictions on deposit transfers during active votes or impose a lock period after voting, where deposits cannot be moved. This would prevent users from reallocating their tokens in a way that could influence voting outcomes unfairly. Additionally, enforcing stricter conditions on when tokens can be moved between deposit entries will help ensure that the governance process remains secure and resistant to manipulation. Proper checks should be consistently applied across all functions handling deposits to prevent users from exploiting this vulnerability.

DEMA - Descriptive Error Messages Absence

Criticality	Minor / Informative
Location	voter-stake-registry/src/error.rs#L4
Status	Unresolved

Description

The error definitions in the program are currently defined without any accompanying error messages. This practice can lead to confusion and make it more challenging to diagnose and understand the reasons behind transaction failures. Providing descriptive error messages helps quickly identify the specific issues, improving the overall experience and facilitating easier debugging.

```
#[error_code]
pub enum VsrError {

    #[msg("")]
    InvalidGoverningMint,
    #[msg("")]
    GoverningTokenNonZero,
    #[msg("")]
    OutOfBoundsDepositEntryIndex,
    #[msg("")]
    InsufficientUnlockedTokens,
    #[msg("")]
    InvalidLockupPeriod,
    #[msg("")]
    DebugInstruction,
    #[msg("")]
    InvalidAuthority,
    #[msg("")]
    InvalidTokenOwnerRecord,
    #[msg("")]
    InvalidRealmAuthority,
    #[msg("")]
    VoterWeightOverflow,
    #[msg("")]
    LockupSaturationMustBePositive,
    #[msg("")]
    InternalProgramError,
    #[msg("")]
    InsufficientLockedTokens,
    #[msg("")]
    InternalErrorBadLockupVoteWeight,
    #[msg("")]
    DepositStartTooFarInFuture,
    #[msg("")]
    VaultTokenNonZero,
    #[msg("")]
    NodeDepositReservedEntryIndex,
    #[msg("")]
    InactiveDepositEntry,
    #[msg("")]
    NotOrdinaryDepositEntry,
    #[msg("")]
    CanNotShortenLockupDuration,
    #[msg("")]
    NodeDepositUnreleasableAtPresent,
    #[msg("")]
    ZeroAmount,
    #[msg("")]
    NodeSecurityDepositMustBePositive,
    #[msg("")]
    DuplicateNodeDeposit,
    #[msg("")]
    ActiveDepositEntryIndex,
    #[msg("")]
    InvalidLockupDuration,
}
```

Recommendation

It is recommended to enhance the error definitions by including descriptive messages for each error type. These messages should clearly explain the nature of the error and provide context to help understand the cause of the issue. By incorporating descriptive error messages, the program will offer more informative feedback. Consistent and clear error messaging is a best practice that significantly contributes to the reliability and usability of the code.

ICEH - Inadequate Clock Error Handing

Criticality	Minor / Informative
Location	voter-stake-registry/src/state/registrar.rs#L62
Status	Unresolved

Description

the code uses `Clock::get().unwrap()` to retrieve the current time in the `clock_unix_timestamp`. This usage is risky because `unwrap()` will panic if the `Clock::get()` call fails, which may cause unintended behavior or disruption in the contract’s normal operations. This behavior compromises the safety of the program since any failure in fetching the clock will cause the program to terminate unexpectedly.

```
pub fn clock_unix_timestamp(&self) -> i64 {
        Clock::get()
            .unwrap()
            .unix_timestamp
            .checked_add(self.time_offset)
            .unwrap()
    }
```

Recommendation

The `unwrap()` method should be replaced with proper error handling. A better approach would be to use Rust's `?` operator, which would allow the function to return an error if the `Clock::get()` call fails. This method would ensure that the function gracefully handles the error instead of panicking.

IC - Incomplete Contract

Criticality	Minor / Informative
Location	voter-stake-registry/src/instructions/update_max_vote_weight.rs#L24
Status	Unresolved

Description

The `update_max_vote_weight` function contains a TODO comment indicating that the SPL governance program has not yet implemented the necessary functionality for handling the maximum vote weight calculation. This lack of implementation makes the contract incomplete since its full functionality is dependent on a future feature from SPL governance.

```
pub fn update_max_vote_weight(ctx:
Context<UpdateMaxVoteWeight>) -> Result<()> {
    let registrar = &ctx.accounts.registrar;
    let _max_vote_weight =
registrar.max_vote_weight(&ctx.accounts.governing_token_mint)?;
    // TODO: SPL governance has not yet implemented this
feature.
    //         When it has, probably need to write the result
into an account,
    //         similar to VoterWeightRecord.
    Ok(())
}
```

Recommendation

Document this as a work in progress and ensure that the required functionality is implemented once the SPL governance program provides the necessary support. In the meantime, consider marking the function as disabled in production or clearly documenting its limitations to prevent confusion or unintended use.

IBU - Inefficient Box<Account> Usage

Criticality	Minor / Informative
Location	voter-stake-registry/src/instructions/close_voter.rs#L11,22 voter-stake-registry/src/state/registrar.rs#L18
Status	Unresolved

Description

The current implementation uses `Box<Account>` for handling the Voter and Registrar accounts, which leads to increased serialization and deserialization overhead. This approach can be suboptimal, particularly when dealing with larger accounts or frequent access. By using `AccountLoader` along with `zero_copy` , the program can load data directly into memory without the need for costly serialization, improving both performance and efficiency.

```
#[account]
pub struct Registrar {
    pub governance_program_id: Pubkey,
    pub realm: Pubkey,
    pub realm_authority: Pubkey,
    pub governing_token_mint: Pubkey,

    /// Storage for voting configuration: voting_config +
reserved2.
    pub voting_config: VotingConfig,
    pub reserved1: [u8; 40],
    /// Storage for deposit configuration: deposit_config +
reserved3.
    pub deposit_config: DepositConfig,
    pub reserved2: [u8; 40],

    // The current value of reward amount per second.
    pub current_reward_amount_per_second: Exponential,

    /// The last time 'current_reward_amount_per_second' was
rotated.
    pub last_reward_amount_per_second_rotated_ts: i64,

    /// The timestamp that rewards was last accrued at
    pub reward_accrual_ts: i64,

    /// Accumulator of the total earned rewards rate since the
opening
    pub reward_index: Exponential,

    /// Amount of rewards that were issued.
    pub issued_reward_amount: u64,

    /// Total permanently locked amount.
    /// Depositions with lockup kind 'Constant' are considered
permanently locked
    pub permanently_locked_amount: u64,

    /// Debug only: time offset, to allow tests to move forward
in time.
    pub time_offset: i64,

    pub bump: u8,
    pub reserved3: [u8; 55],
}

pub registrar: Box<Account<'info, Registrar>>

pub voter: Box<Account<'info, Voter>>
```

IVUR - Inefficient Variable Usage Repetition

Criticality	Minor / Informative
Location	voter-stake-registry/src/instructions/node_deposit.rs#L59,76
Status	Unresolved

Description

The `node_security_deposit` variable from `registrar.deposit_config` is used multiple times throughout the function without being stored in a local variable. This leads to unnecessary repetition and reduces code readability. Storing the value in a local variable at the start of the function would simplify the code and make it easier to maintain and modify in the future.

```
let node_security_deposit =
registrar.deposit_config.node_security_deposit;
```

Recommendation

It is advised to store the `node_security_deposit` value in a local variable at the beginning of the function. This approach improves readability, reduces code repetition, and makes future updates easier to manage. By referencing the local variable instead of repeatedly accessing the same value, the code will become more concise and maintainable.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	voter-stake-registry/src/instructions/voter-stake-registry/src/instructions/node_release_deposit.rs#L46.rs#L17 voter-stake-registry/src/instructions/update_voter_weight_record.rs#L36 voter-stake-registry/src/instructions/update_voting_config.rs#L20
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
pub fn update_deposit_config(
    ctx: Context<UpdateDepositConfig>,
    deposit_config: DepositConfig,
) -> Result<()> {
    require!(
        deposit_config.node_security_deposit > 0,
        VsrError::NodeSecurityDepositMustBePositive
    );

    let registrar = &mut ctx.accounts.registrar;
    registrar.deposit_config = deposit_config;

    Ok(())
}

pub fn update_voter_weight_record(ctx:
Context<UpdateVoterWeightRecord>) -> Result<()> {
    let registrar = &ctx.accounts.registrar;
    let voter = &ctx.accounts.voter;
    let record = &mut ctx.accounts.voter_weight_record;
    let curr_ts = registrar.clock_unix_timestamp();
    record.voter_weight = voter.weight(curr_ts, registrar)?;
    record.voter_weight_expiry = Some(Clock::get()?.slot);

    Ok(())
}

pub fn update_voting_config(
    ctx: Context<UpdateVotingConfig>,
    voting_config: VotingConfig,
) -> Result<()> {
    require!(
        voting_config.lockup_saturation_secs > 0,
        VsrError::LockupSaturationMustBePositive
    );

    let registrar = &mut ctx.accounts.registrar;
    registrar.voting_config = voting_config;

    // Check for overflow in vote weight

registrar.max_vote_weight(&ctx.accounts.governing_token_mint)?;

    Ok(())
}
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

PCR - Program Centralization Risk

Criticality	Minor / Informative
Location	voter-stake-registry/src/instructions/update_deposit_config.rs#L17 voter-stake-registry/src/instructions/update_voting_config.rs#L20 circuit-breaker/src/instructions/initialize_account_windowed_breaker.rs#L38 circuit-breaker/src/instructions/update_account_windowed_breaker.rs#L21
Status	Unresolved

Description

The program's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion. Specifically, the program's functionality and behavior are heavily dependent on external parameters or configurations. These functions must be executed by a specific authorized account to set and update critical parameters within the protocol.

```
pub fn update_deposit_config(
    ctx: Context<UpdateDepositConfig>,
    deposit_config: DepositConfig,
) -> Result<()> {
    require!(
        deposit_config.node_security_deposit > 0,
        VsrError::NodeSecurityDepositMustBePositive
    );

    let registrar = &mut ctx.accounts.registrar;
    registrar.deposit_config = deposit_config;

    Ok(())
}

pub fn update_voting_config(
    ctx: Context<UpdateVotingConfig>,
    voting_config: VotingConfig,
) -> Result<()> {
    require!(
        voting_config.lockup_saturation_secs > 0,
        VsrError::LockupSaturationMustBePositive
    );

    let registrar = &mut ctx.accounts.registrar;
    registrar.voting_config = voting_config;

    // Check for overflow in vote weight

registrar.max_vote_weight(&ctx.accounts.governing_token_mint)?;

    Ok(())
}

pub fn initialize_account_windowed_breaker(
  ctx: Context<InitializeAccountWindowedBreakerV0>,
  args: InitializeAccountWindowedBreakerArgsV0,
) -> Result<()> {
  require!(args.config.is_valid(), ErrorCode::InvalidConfig);

  ctx
    .accounts
    .circuit_breaker
    .set_inner(AccountWindowedCircuitBreakerV0 {
      token_account: ctx.accounts.token_account.key(),
      authority: args.authority,
      owner: args.owner,
      config: args.config,
      last_window: WindowV0 {
        last_aggregated_value: 0,
        last_unix_timestamp: 0,
      },
      bump_seed: ctx.bumps.circuit_breaker,
    });

  set_authority(
    CpiContext::new(
      ctx.accounts.token_program.to_account_info(),
      SetAuthority {
        account_or_mint:
ctx.accounts.token_account.to_account_info(),
        current_authority:
ctx.accounts.owner.to_account_info(),
      },
    ),
    AuthorityType::CloseAccount,
    Some(ctx.accounts.circuit_breaker.key()),
  )?;

  set_authority(
    CpiContext::new(
      ctx.accounts.token_program.to_account_info(),
      SetAuthority {
        account_or_mint:
ctx.accounts.token_account.to_account_info(),
        current_authority:
ctx.accounts.owner.to_account_info(),
      },
    ),
    AuthorityType::AccountOwner,
    Some(ctx.accounts.circuit_breaker.key()),
  )?;

  Ok(())
}

pub fn update_account_windowed_breaker(
  ctx: Context<UpdateAccountWindowedBreakerV0>,
  args: UpdateAccountWindowedBreakerArgsV0,
) -> Result<()> {
  let circuit_breaker = &mut ctx.accounts.circuit_breaker;
  if args.new_authority.is_some() {
    circuit_breaker.authority = args.new_authority.unwrap();
  }
  if args.config.is_some() {
    circuit_breaker.config = args.config.unwrap();
  }

  Ok(())
}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the program's codebase itself. This approach would reduce external dependencies and enhance the program's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

RB - Redundant Borrowing

Criticality	Minor / Informative
Location	voter-stake-registry/src/instructions/close_voter.rs#L46 voter-stake-registry/src/instructions/withdraw.rs#L107
Status	Unresolved

Description

The code contains an instance where a reference is created and immediately dereferenced by the compiler. This pattern is redundant and results in unnecessary complexity without adding any functional benefit. The use of needless borrowing makes the code less efficient and harder to read, as it introduces unnecessary operations. While the compiler handles dereferencing, this pattern could also lead to confusion for developers maintaining the code, as it deviates from Rust's idiomatic approach.

```
let token =
Account::<TokenAccount>::try_from(&account).unwrap();

if *ctx.accounts.token_owner_record.owner ==
registrar.governance_program_id {
        // Governance may forbid withdraws, for example when
engaged in a vote.
        let token_owner_record = load_token_owner_record(
            &ctx.accounts.token_owner_record.to_account_info(),
            &voter,
            registrar,
        )?;

token_owner_record.assert_can_withdraw_governing_tokens()?;
    }
```

Recommendation

To improve code readability and efficiency, unnecessary references should be removed in cases where they are immediately dereferenced. By doing so, the code will become more concise and follow Rust's best practices for reference handling, making it easier to maintain and understand.

RFN - Redundant Field Names

Criticality	Minor / Informative
Location	voter-stake-registry/src/instructions/ordinary_deposit.rs#L125
Status	Unresolved

Description

The code contains instances where field names are redundantly repeated in struct initializations. In these cases, the field name and the variable being assigned to it are the same. While this does not impact functionality, it introduces unnecessary verbosity, making the code less concise and harder to read. This redundancy can also lead to potential confusion when maintaining or extending the codebase, as it adds extra noise without providing additional clarity or benefit.

```
amount: amount,
```

Recommendation

The code should be updated to remove redundant field names in struct initialization where the field and variable names are identical. This will make the code cleaner and more readable, reducing unnecessary complexity. Following Rust's idiomatic approach to struct initialization will also help maintain consistency across the codebase.

RMB - Redundant Mutable Borrowing

Criticality	Minor / Informative
Location	voter-stake-registry/src/instructions/node_deposit.rs#L58,65
Status	Unresolved

Description

The contract contains instances of mutable borrowing for the `registrar` account. While mutable references are necessary for modifying the state of these accounts, borrowing them repeatedly throughout the code introduces redundancy and makes the code harder to follow. This repetition can increase the complexity of managing the borrow checker and potentially lead to borrowing conflicts, as Rust allows only one mutable reference at a time. Additionally, it detracts from code readability and maintainability by cluttering the logic with unnecessary mutable references.

```
let registrar = &mut ctx.accounts.registrar;
```

Recommendation

It is recommended to refactor the code by consolidating the mutable references to the `registrar` account. By limiting mutable borrowing to only the necessary sections of the code and combining these borrow operations, the code will be cleaner and more efficient. This approach will reduce redundancy and simplify the logic, enhancing both readability and maintainability without affecting the contract's functionality.

RCR - Repeated Constraints Reuse

Criticality	Minor / Informative
Location	voter-stake-registry/src/instructions/claim_reward.rs#L22,23 voter-stake-registry/src/instructions/log_voter_info.rs#L10 voter-stake-registry/src/instructions/node_deposit.rs#L19 voter-stake-registry/src/instructions/node_release_deposit.rs#L18,19
Status	Unresolved

Description

There are multiple instances where constraints are repeatedly applied across different account validation structs. Specifically, constraints such as checking the association between a voter and a registrar, or validating token accounts, are repeatedly implemented in various parts of the smart contract. This leads to code duplication, increased complexity, and a larger potential for errors or inconsistencies in the event that any of these constraints need to be updated or modified. The repetition of similar logic across multiple locations can also make the codebase harder to maintain and less readable, as future changes may require updates in multiple places, increasing the risk of missed or inconsistent updates.

```
constraint = voter.get_registrar() == registrar.key(),
constraint = voter.get_voter_authority() ==
voter_authority.key(),
```

Recommendation

To improve maintainability and reduce code duplication, it is recommended to abstract commonly used constraints into reusable functions or modules. By centralizing these validations, the codebase will become more modular, easier to manage, and less prone to errors. This also allows for updates to the constraints to be applied consistently across the system by modifying the logic in a single location rather than multiple scattered instances. Adopting this approach ensures cleaner code, promotes reusability, and reduces the overall complexity of the contract.

TE - Typographical Errors

Criticality	Minor / Informative
Location	voter-stake-registry/src/state/registrar.rs#L85 voter-stake-registry/src/instructions/ordinary_deposit.rs#L36
Status	Unresolved

Description

The code contains several typographical errors that could impact clarity and readability. Examples include the misspelling of "accrue" as "accure" and "duration" as "duraiton." While these typos may not directly affect the functionality of the program, they introduce unnecessary complexity and could lead to confusion for developers and auditors trying to understand or maintain the code. Proper spelling is essential in source code to maintain consistency and prevent potential misunderstandings.

```
pub fn accure_rewards(&mut self, curr_ts: i64) {

duraiton: LockupTimeDuration,
```

Recommendation

It is recommended to review the entire codebase for typographical errors and correct any found instances. Clear and accurate variable names and function definitions contribute to code readability, maintainability, and prevent miscommunication among team members or future developers.

UC - Unformatted Code

Criticality	Minor / Informative
Location	/voter-stake-registry/src/instructions/update_voting_config.rs#L8,9
Status	Unresolved

Description

The codebase contains instances of trailing whitespace that result in unformatted code. Trailing whitespace does not affect the program's functionality, but it can lead to inconsistencies in the code's appearance and structure. Unformatted code can reduce readability and complicate version control.

```
#[derive(Accounts)]
pub struct UpdateVotingConfig<'info> {
    #[account(
        mut,
        has_one = governing_token_mint,
        has_one = realm_authority,
    )]
    pub registrar: Box<Account<'info, Registrar>>,
    pub governing_token_mint: Box<Account<'info, Mint>>,

    pub realm_authority: Signer<'info>,
}
```

Recommendation

Ensure that all trailing whitespace is removed from the codebase, and consistently format the code. Adopting a uniform formatting style improves code readability and maintainability.

UTC - Unnecessary Type Casting

Criticality	Minor / Informative
Location	voter-stake-registry/src/state/deposit_entry.rs#L233
Status	Unresolved

Description

The code contains an instance where a value is cast to the same type (u64 to u64), which is redundant and does not serve any functional purpose. Unnecessary type casting introduces extra complexity and can make the code harder to read and maintain without providing any additional benefit. In this case, the casting operation can be safely removed to streamline the code and improve clarity.

```
fn voting_power_linear_vesting(
        &self,
        curr_ts: i64,
        max_locked_vote_weight: u64,
        lockup_saturation_secs: u64,
    ) -> Result<u64> {
        let periods_left = self.lockup.periods_left(curr_ts)?;
        let periods_total = self.lockup.periods_total();
        let period_secs = self.lockup.kind().period_secs() as
u64;

        if periods_left == 0 {
            return Ok(0);
        }
```

Recommendation

It is recommended to remove unnecessary type casting where the source and target types are the same. This will make the code cleaner, reduce redundancy, and improve readability, following best practices for type handling in Rust.

VAUD - Vulnerabilities and Unmaintained Dependencies

Criticality	Minor / Informative
Location	voter-stake-registry/
Status	Unresolved

Description

The project relies on several third-party crates that contain known security vulnerabilities or have been marked as unmaintained. These vulnerabilities could expose the project to security risks, including timing attacks, denial of service, infinite loops, and configuration corruption. Additionally, using unmaintained dependencies can introduce further risks, as they may no longer receive security patches or updates. These issues affect critical crates within the dependency tree, potentially impacting the overall security and stability of the project. Such issues can be identified by running a tool like `cargo audit`, which helps detect vulnerabilities in dependencies.

Recommendation

Review all identified vulnerabilities and unmaintained dependencies in the project's third-party crates. It is recommended to update the affected crates to the latest secure versions where possible, or consider alternative libraries that are actively maintained. Regularly monitoring and auditing third-party dependencies with tools such as `cargo audit` is crucial to ensure the project's security and integrity, reducing exposure to potential risks.

Summary

The BrushO Network implements the Circuit Breaker, which enforces token transfer limits within time windows, and the Voter Stake Registry, which is a voter weight add-in for Solana's spl-governance program. This audit investigates security issues, business logic concerns, and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

cyberscope.io