Cyberscope

Audit Report

BrushO Network

September 2024

Repository	https://github.com/aitoothbrush/brusho-program-library
Commit	422c414eaed865db88dd0031895db58107674527
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	422c414eaed865db88dd0031895db58107674527
Network	SOL

Audit Updates

Initial Audit	04 Sept 2024 https://github.com/cyberscope-io/audits/blob/main/brush/v1/audit.pdf
Corrected Phase 2	19 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/circuit_breaker.rs	5dd4d0722dedd70689d73d354157631f1bc5b63ff0d562b44e7a92dea4e89485
voter-stake-registry/src/error.rs	74904036751945845e1a16934d66ea6b38bcc46396812d4f520cf890c775966e
voter-stake-registry/src/governance.rs	02ad620279f079456b8961e120f95757a3ac552ed03228c6cea6b0265132e926
voter-stake-registry/src/lib.rs	be6be6054d722fe9d8171fe36c9c3ac3b3f462c2ca9ffc2b5edb0b6ffed80f9f
voter-stake-registry/src/events/mod.rs	b3ac99547effae8a2af5ad237d922c11e50def815064913f658baacec571e788
voter-stake-registry/src/instructions/claim_reward.rs	77c49f7e8dbd970feec91266c3dba6db49aba1a9bdc1976244b7cd70df35d349
voter-stake-registry/src/instructions/close_voter.rs	b3f6f96391495d32b618b11e0fb03cbc0b4067ef857fc068fad0bb6f81234fd9
voter-stake-registry/src/instructions/create_registrar.rs	da9b034c5f65939e95edf231f85cdf3007c2728553bcc80824f55551688aa613
voter-stake-registry/src/instructions/create_voter.rs	cc4e50d725ec3542730b826ad5d1f811d3520ff3bebed8eddbd7cd95fa25bbb3

voter-stake-registry/src/instructions/log_voter_info.rs	0cae317fd6e169320692364d4c22212c4dacf474dd5350936415b2910b1168d0
voter-stake-registry/src/instructions/mod.rs	9df9d46d8edd5751366e8ab10300363950c6bbffc2c134537fe16044618996e0
voter-stake-registry/src/instructions/node_deposit.rs	577877e8aaea6612fabf1b13d6b30c811307ae8b688795a0fe83c1369f2c19fb
voter-stake-registry/src/instructions/node_release_deposit.rs	d3892c347102319d502d7970173595248e9170b163cb36b9261cffb2b7a2c7df
voter-stake-registry/src/instructions/ordinary_deposit.rs	3cc3689792a08dc7406b86e7378ece6c593a08c0d8ac3924e338d17b82a16188
voter-stake-registry/src/instructions/ordinary_release_deposit.rs	1ec41d15835e543f479dcc22ab5c869d4750ad0c3599126552497c2dfd2f3bfd
voter-stake-registry/src/instructions/set_time_offset.rs	339e33ef27d7484d0d92dec5d39e25f812169ef3dc4a7168b00d743a4bd8a7a1
voter-stake-registry/src/instructions/update_deposit_config.rs	9e4d28b7941afa8625f840ad54d885e86539b89eb3a29bd50c78b730b85e5ce9
voter-stake-registry/src/instructions/update_max_vote_weight.rs	ecae8b69f742b5bc959d98991b54e3c600db977639fa1fda672bd63f7b171349
voter-stake-registry/src/instructions/update_voter_weight_record.rs	9c38b6796b843ac64d02658ad3407e94a7f977856ab36c2c54633e297e236255

voter-stake-registry/src/instructions/update_voting_config.rs	15d3551f244c5de706db95339a4b98071a79ac33ad6ca417d947aa16bda66c73
voter-stake-registry/src/instructions/withdraw.rs	5a6a31d553148f5986d1293e419676959c85366eb59c89a4545bbdbef8b2ab9a
voter-stake-registry/src/state/deposit_entry.rs	1ac4325aa8e6ecb389a4b24e296dbefc7d95f81a140fd7064ad9009f8af3d5d2
voter-stake-registry/src/state/lockup.rs	769864b43ae322847d921407cfb0912017677c5e386e8ac2f343c749ed737252
voter-stake-registry/src/state/mod.rs	d20ac91bb4756c0ce1f8cec6028ab03d8ce9e7af260e6e25151f31be120d4163
voter-stake-registry/src/state/registrar.rs	fbf763343f27725ff760e02602379196834732b8dcaaad6531932e077a019f32
voter-stake-registry/src/state/voter.rs	8e870128e84f34fc6eb407842edc6f029f504f3345ca9612d5fddc9348307d03

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	1
Minor / Informative	10

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	1	0	0
Minor / Informative	10	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	PVPM	Potential Voting Power Manipulation	Acknowledged
Minor	DEMA	Descriptive Error Messages Absence	Unresolved
Minor	ICEH	Inadequate Clock Error Handing	Unresolved
Minor	IVUR	Inefficient Variable Usage Repetition	Unresolved
Minor	MEE	Missing Events Emission	Unresolved
Minor	PCR	Program Centralization Risk	Unresolved
Minor	RVB	Redundant Variable Binding	Unresolved
Minor	RCR	Repeated Constraints Reuse	Unresolved
Minor	RTC	Redundant Type Casting	Unresolved
Minor	UTC	Unnecessary Type Conversion	Unresolved
Minor	VAUD	Vulnerabilities and Unmaintained Dependencies	Unresolved

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	Acknowledged

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.load_mut()?;
    let voter = &mut ctx.accounts.voter.load_mut()?;

    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
    );

    // accrue rewards
    let curr_ts = registrar.clock_unix_timestamp();
    registrar.accrue_rewards(curr_ts);

    let node_security_deposit =
d_entry.get_amount_deposited_native();
    let lockup = d_entry.get_lockup();
    let lockup_kind = lockup.kind;
    if let LockupKindKind::Constant = lockup_kind.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(lockup_kind.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.load_mut()?;
    let voter = &mut ctx.accounts.voter.load_mut()?;

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

    let curr_ts = registrar.clock_unix_timestamp();
    // accrue rewards
    registrar.accrue_rewards(curr_ts);

    let lockup = d_entry.get_lockup();
    let lockup_kind = lockup.kind;
    if let LockupKindKind::Constant = lockup_kind.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(lockup_kind.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.

```
use anchor_lang::prelude::*;

#[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,
    #[msg("")]
    InsufficientClaimableRewards,
}
```

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#L63
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.

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/update_deposit_config.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.load_mut()?;
    registrar.deposit_config = deposit_config;

    Ok(())
}

pub fn update_voter_weight_record(ctx:
Context<UpdateVoterWeightRecord>) -> Result<()> {
    let registrar = &ctx.accounts.registrar.load()?;
    let voter = &ctx.accounts.voter.load()?;
    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.load_mut()?;
    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 voter-stake-registry/src/instructions/update_max_vote_weight.rs#L30 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.load_mut()?;
    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.load_mut()?;
    registrar.voting_config = voting_config;

    // Check for overflow in vote weight

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

    Ok(())
}

pub fn update_max_vote_weight(ctx:
Context<UpdateMaxVoteWeight>) -> Result<()> {
    let registrar = &ctx.accounts.registrar.load()?;

    let record = &mut ctx.accounts.max_voter_weight_record;
    record.max_voter_weight =
registrar.max_vote_weight(&ctx.accounts.governing_token_mint)?;
    record.max_voter_weight_expiry = Some(Clock::get()?.slot);

    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.

RVB - Redundant Variable Binding

Criticality	Minor / Informative
Location	voter-stake-registry/src/instructions/claim_reward.rs#L25
Status	Unresolved

Description

In the `claim_reward` function, a variable is declared to hold the result of a function call, which is then returned immediately. This introduces unnecessary complexity and redundancy in the code. Declaring a variable only to return it without further manipulation adds an extra step that reduces code clarity and can potentially lead to confusion for future maintainers. Simplifying the return statement by directly returning the result improves the code's readability and efficiency.

```
let claimed_amount = voter.claim_reward(curr_ts, amount,
registrar)?;
claimed_amount
```

Recommendation

The function should be refactored to return the result of the function call directly, without introducing an intermediate variable. This will help streamline the code, making it more concise and easier to understand, while maintaining the same functionality. Simplifying code in this manner is a good practice that leads to cleaner and more maintainable codebases.

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.load()?.get_registrar() == registrar.key(),
constraint = voter.load()?.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.

RTC - Redundant Type Casting

Criticality	Minor / Informative
Location	voter-stake-registry/src/state/lockup.rs#L142
Status	Unresolved

Description

There is an instance where a value is cast to the same type it already holds. Casting a variable to the same type introduces redundancy and does not provide any functional advantage. This type of unnecessary casting can slightly reduce the code's readability and add confusion for maintainers. Removing such redundant casting will result in cleaner and more efficient code.

```
self.kind.periods() as u64
```

Recommendation

It is recommended to remove unnecessary type casting from the code. By avoiding redundant type conversions, the code becomes more concise and easier to understand, while maintaining the same functionality. This small change helps to streamline the codebase and ensures better maintainability in the long run.

UTC - Unnecessary Type Conversion

Criticality	Minor / Informative
Location	voter-stake-registry/src/state/lockup.rs#L247
Status	Unresolved

Description

There is an instance of an unnecessary conversion of a variable into the same type it already holds. Converting a variable to the same type does not provide any functional benefit and introduces redundant operations in the code. These redundant conversions can impact the clarity of the code and should be avoided for better readability and efficiency.

```
pub fn seconds(&self) -> u64 {
    self.unit
        .seconds()
        .checked_mul(self.periods.into())
        .unwrap()
}
```

Recommendation

It is recommended to remove redundant type conversions. By eliminating unnecessary `.into()` calls, the code will be more streamlined and easier to follow. This will improve both readability and maintainability without affecting the functionality of the program.

VAUD - Vulnerabilities and Unmaintained Dependencies

Criticality	Minor / Informative
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