CERTIK

Security Assessment

Lucia - Audit

CertiK Assessed on Jun 27th, 2024

CertiK Assessed on Jun 27th, 2024

Lucia - Audit

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
Vesting	Solana (SOL)	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Rust	Delivered on 06/27/2024	N/A

CODEBASE

Lucia Github Repo

View All in Codebase Page

COMMITS

- 67ce48e87112c47b3f11fd900051bbf0184fc83f
- 99d471ccf823819ab91f822aa1d85a265599259c
- 6f3465487f1adc22f7a06ef63973024799c2e803

View All in Codebase Page

Highlighted Centralization Risks

Contract upgradeability

Vulnerability Summary

15	11	0	1	3	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

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.
3	Major	2 Resolved, 1 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
6	Medium	6 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
4	Minor	2 Resolved, 1 Partially Resolved, 1 Acknowledged	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.
1	Informational	1 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 | LUCIA - AUDIT

CODEBASE | LUCIA - AUDIT

Repository

Lucia Github Repo

Commit

- 67ce48e87112c47b3f11fd900051bbf0184fc83f
- 99d471ccf823819ab91f822aa1d85a265599259c
- 6f3465487f1adc22f7a06ef63973024799c2e803
- 5317945d14acd3ef988b8ca0b819f0a8c6a8e7f7
- 69edcbde64ea6b1d831d3ef362f3648562e55391
- c1996e20ac0e6d8900f30c5af83702df011397a6

AUDIT SCOPE | LUCIA - AUDIT

1 file audited ● 1 file with Acknowledged findings

ID	Repo	File	SHA256 Checksum
● LCD	DavidLee9291/Lucia_Contract	programs/lucia_vesting/src/lib.rs	3e0cbcccbce363c1a9ba392c04d803b3e96393f90840277f1105e48590e9ee55

APPROACH & METHODS | LUCIA - AUDIT

This report has been prepared for Lucia to discover issues and vulnerabilities in the source code of the Lucia - Audit project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing 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 | LUCIA - AUDIT

System Overview

This project provides token assets vesting service which locks token assets first and releases the assets to the specific beneficiaries every month if the lockup period is expired.

The contract uses an unknown `token_mint` account, and it is uncertain whether the tokens can be paused, frozen, or destroyed.

External Dependencies

The project mainly contains the following dependencies:

Dependency	Version
anchor-lang	0.30.0
anchor-spl	0.30.0

We assume these dependencies are valid and non-vulnerable factors and implement proper logic to collaborate with the current project.

Privileged Functions

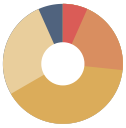
Any compromise to the **initializer** account may allow a hacker to take advantage of this authority and update the state.

In `lib.rs`, the `initializer` who initialize the contract has the highest level of control over the variable, as specified in finding **LCD-04**.

Any compromise to the privileged roles may allow the hacker to take advantage of this authority.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should also be considered to move to the execution queue of `Timelock` contract.

FINDINGS | LUCIA - AUDIT



15	1	3	6	4	1
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Lucia - Audit. Through this audit, we have uncovered 15 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
LDL-02	Repeated Reward Over-Claiming In `claim_lux()` Function	Logical Issue	Critical	Resolved
LCD-01	Missing `initialized_at` Variable Initialization	Logical Issue	Major	Resolved
LCD-02	Incorrect Reward Calculation	Logical Issue	Major	Resolved
LCD-04	**Centralization Related Risks And Upgradability**	**Centralization**	**Major**	**Acknowledged**
CDL-01	Incomplete Production Code	Volatile Code	Medium	Resolved
LCD-03	Missing State Verification In `claim_lux()` Function	Logical Issue	Medium	Resolved
LCD-05	Unrestricted Access To `initialize()` Function	Logical Issue	Medium	Resolved
LCD-06	Unrestricted Reward Claims In `claim_lux()` Function	Logical Issue	Medium	Resolved
LCD-11	Incorrect Space Size Constraint For `data_account`	Volatile Code	Medium	Resolved
LIB-01	Invalid Amount Check Condition	Coding Issue	Medium	Resolved
CAL-01	Lack Of Check For `confirm_round`	Logical Issue	Minor	Resolved

ID	Title	Category	Severity	Status
LCD-07	Inadequate Initialization Checks	Logical Issue	Minor	Partially Resolved
LCD-08	Out-Of-Scope Dependencies	Volatile Code	Minor	Acknowledged
LCD-09	Unsafe Integer Cast	Incorrect Calculation	Minor	Resolved
CDL-02	Potential Front-Running Risk With Initialize Instructions	Logical Issue, Volatile Code	Informational	Acknowledged

LDL-02 REPEATED REWARD OVER-CLAIMING IN `claim_lux()` FUNCTION

Category	Severity	Location	Status
Logical Issue	Critical	programs/lucia_vesting/src/lib.rs (v2): 128	Resolved

Description

The `claim_lux()` function, intended to allow users to claim due rewards, contains a logic error. The variable `confirm_round`, which represents the number of reward rounds claimed, is not updated after each claim. This oversight allows the `calculate_schedule` function to compute rewards from the `zero` to `vesting_end_month` for each claim, enabling users to claim more than their due share.

lucia_vesting/src/lib.rs

```
128         let confirm_round = beneficiary.confirm_round;
129
130         // LCD - 02
131         let schedule = calculate_schedule(
132             lockup_end_time,
133             vesting_end_month as i64,
134             beneficiary.unlock_duration as i64,
135             allocated_tokens as i64,
136             confirm_round
137         );
138
139         let mut total_claimable_tokens: u64 = 0;
140
141         for item in schedule {
142             let round_num = item.0.split(": ").nth(1).unwrap().parse::<u64>().
unwrap();
143             if current_time >= item.1 && (confirm_round as u64) <= round_num {
144                 msg!(
145                     "Tokens claimable: {}, timestamp: {}, claimable_token: {}",
146                     item.0,
147                     item.1,
148                     item.2
149                 );
150                 total_claimable_tokens += item.2 as u64;
151             } else {
152                 msg!(
153
"Tokens not claimable: {}, timestamp: {}, claimable_token: {}",
154                     item.0,
155                     item.1,
156                     item.2
157                 );
158             }
```

The variable `confirm_round` represents the number of rounds for which the award has been collected. However, the formula used in line 131 to calculate the schedule does not take into account the number of award periods that have been claimed. This causes that `total_claimable_tokens` variable always accumulates all rewards that have expired when calculating unlockable `LUX` assets.

As a result, the beneficiary can receive the first month's unlocked assets multiple times.

Proof of Concept

Below is an example test code in Js for the described finding:

```
  it("Test Double Claim", async () => {
    dataAccount = _dataAccountAfterClaim;
    const doubleClaimTx = await program.methods
      .claimLux(dataBump, escrowBump)
      .accounts({
        dataAccount: dataAccount,
        escrowWallet: escrowWallet,
        sender: beneficiary.publicKey,
        tokenMint: mintAddress,
        walletToDepositTo: beneficiaryATA,
        associatedTokenProgram: spl.ASSOCIATED_TOKEN_PROGRAM_ID,
        tokenProgram: spl.TOKEN_PROGRAM_ID,
        systemProgram: anchor.web3.SystemProgram.programId,
      })
      .signers([beneficiary])
      .rpc();
    console.log(
      `double claim TX: https://explorer.solana.com/tx/${doubleClaimTx}?
cluster=custom`
    );
    await provider.connection.confirmTransaction(doubleClaimTx);

    const beneficiaryBalance = await getTokenBalanceWeb3(
      beneficiaryATA,
      provider
    );

    console.log("Double Beneficiary Balance:", beneficiaryBalance.toString());
  });
```

```
double claim TX:
https://explorer.solana.com/tx/2w98tukGa5gMg5hJj796r1biKLagrWs2ZbvDpijjwxJ7rZo1i1ErU
keCBpJJzx5o9Vq3B9U7mNsxiMT1PoZ69TQV?cluster=custom
Double Beneficiary Balance: 16666666
    ✔ Test Double Claim (471ms)
```

Recommendation

Ensure that `confirm_round` is updated correctly after each reward claim to prevent users from claiming rewards multiple times.

Alleviation

[Lucia Team, 06/14/2024]: The team heeded the advice and resolved the issue in commit: 5317945d14acd3ef988b8ca0b819f0a8c6a8e7f7.

LCD-01 | MISSING `initialized_at` VARIABLE INITIALIZATION

Category	Severity	Location	Status
Logical Issue	Major	programs/lucia_vesting/src/lib.rs (v1): 22	Resolved

Description

lucia_vesting/src/lib.rs

```
 268  #[account]
 269  #[derive(Default)]
 270  pub struct DataAccount {
 271
// Space in bytes: 8 + 1 + 8 + 32 + 32 + 32 + 8 + 1 + (4 + (100 * (32 + 8 + 8 + 8 +
8 + 8)))

 272      pub state: u8, // 1
 273      pub token_amount: u64, // 8
 274      pub initializer: Pubkey, // 32
 275      pub escrow_wallet: Pubkey, // 32
 276      pub token_mint: Pubkey, // 32
 277  @>  pub initialized_at: u64, // 8
 278      pub beneficiaries: Vec<Beneficiary>,
// (4 + (n * (32 + 8 + 8 + 8 + 8 + 8)))
 279      pub decimals: u8, // 1
 280  }
```

The `initialized_at` variable denotes the start time of the `LUX` token asset lockup and is the key parameter for calculating the release amount of the `LUX` token assets. The current issue with the code is that the beneficiary can claim assets even if the lockup period hasn't expired.

The following code snippets are used to verify if the lockup period has expired:

1. `data_account.initialized_at` is zero, which is the default value of the `u64` type, due to the absence of its value assignment within the `initialize()` function.
2. Based on the test code within `lucia_vesting.ts`, both the `beneficiary.lockup_period` and `beneficiary.unlock_duration` are calculated as $12 \times 30 \times 24 \times 60 \times 60 = 31,104,000$ seconds, which corresponds to 12 months.

lucia_vesting/src/lib.rs

```
 81          // Check if the lockup period has expired
 82          let current_time = Clock::get()?.unix_timestamp as u64;
 83          let lockup_end_time = data_account.initialized_at + beneficiary.
lockup_period;
 84          msg!("lockup_end_time : {}", lockup_end_time);
 85
 86          // 락업 기간이 지나지 않으면 실행하지 않음
 87          if current_time < lockup_end_time {
 88              msg!("Lockup period has not expired");
 89              return Err(VestingError::LockupNotExpired.into());
 90          }
 91
 92
// Calculate the unlockable tokens based on the unlock duration and unlockTge
 93          let time_since_lockup_end = current_time - lockup_end_time;
 94          msg!("time lockup : {}", time_since_lockup_end);
 95          msg!("unlock duration : {}", beneficiary.unlock_duration);
 96
 97          // Calculate the unlockable tokens
 98          let mut unlockable_tokens: u64 = 0;
 99
100          if time_since_lockup_end >= beneficiary.unlock_duration {
101              ....
102          }
```

The `lockup_end_time` is calculated as $12 \times 30 \times 24 \times 60 \times 60 = 31,104,000$ seconds, which is certainly much less than `current_time`, representing the approximate real-world time of the current slot. Therefore, the condition `current_time < lockup_end_time` evaluates to false.

The `time_since_lockup_end`, calculated as `current_time - lockup_end_time`, is guaranteed to be greater than `beneficiary.unlock_duration`.

As a result, the beneficiary can claim the locked-up assets, bypassing the lockup period.

Proof of Concept

Below is an example test code in Js for the described finding:

```
  it("Test Initialize", async () => {
    // Send initialize transaction
    const initTx = await program.methods.initialize(beneficiaryArray, new
anchor.BN(1000000000), decimals).accounts({
      dataAccount: dataAccount,
      escrowWallet: escrowWallet,
      walletToWithdrawFrom: senderATA,
      tokenMint: mintAddress,
      sender: sender.publicKey,
      systemProgram: anchor.web3.SystemProgram.programId,
      tokenProgram: spl.TOKEN_PROGRAM_ID,
    }).signers([sender]).rpc();

    let accountAfterInit = await program.account.dataAccount.fetch(dataAccount);

    console.log(`init TX: https://explorer.solana.com/tx/${initTx}?cluster=custom`)
    console.log(`initializedAt: ${accountAfterInit.initializedAt}`)

  });
```

```
init TX:
https://explorer.solana.com/tx/yXHaecrzLE3CdMmJadhAZJL75wcyf1fgyqHgaBX5yenLxQFyGiGc7
PEXveuAm815gtMfkCRbUSmTQ5P4adeBetn?cluster=custom
initializedAt: 0
    ✔ Test Initialize (481ms)
```

Recommendation

Modify the `initialize` function to assign the `initialized_at` variable, ensuring accurate reward release calculations. A potential solution is to set `initialized_at` to the current block timestamp while the `initialize()` instruction is invoked.

Alleviation

[Lucia Team, 06/11/2024]: The team heeded the advice and resolved the issue in commit: 99d471ccf823819ab91f822aa1d85a265599259c.

LCD-02 | INCORRECT REWARD CALCULATION

Category	Severity	Location	Status
Logical Issue	Major	programs/lucia_vesting/src/lib.rs (v1): 102	Resolved

Description

The code below calculates the amount of lockup assets that can be unlocked for a single month. The current flaw is that it only unlocks the `LUX` token assets for the first month.

lucia_vesting/src/lib.rs

```
 97          // Calculate the unlockable tokens
 98          let mut unlockable_tokens: u64 = 0;
 99
100          if time_since_lockup_end >= beneficiary.unlock_duration {
101              // Calculate the number of months since lockup ended
102      @>      let months_passed = (time_since_lockup_end / 2592000) as f64;
// Assuming unlock_duration is in seconds
103              msg!("months_passed : {}", months_passed);
104
105              // Calculate the unlockable tokens for each month
106              let tokens_per_month = (beneficiary.allocated_tokens as f64) / 12.0
;
107              msg!("tokens_per_month : {}", tokens_per_month);
108
109
// Calculate additional tokens for the first month based on the unlock TGE
percentage

110              let unlock_tge_percentage = beneficiary.unlock_tge as f64;
111              let additional_tokens_first_month =
112                  (unlock_tge_percentage / 100.0) * (beneficiary.
allocated_tokens as f64);
113              msg!("additional_tokens_first_month : {}",
 additional_tokens_first_month);
114
115              // Calculate total unlockable tokens
116      @>      unlockable_tokens = (tokens_per_month +
 additional_tokens_first_month) as u64;
117          }
118
119          // Calculate the amount to transfer
120      @>  let amount_to_transfer = unlockable_tokens.saturating_sub(beneficiary.
claimed_tokens);
121          ....
122
123          // Check if the claimed amount exceeds the previously claimed amount
124      @>  require!(amount_to_transfer > 0, VestingError::ClaimNotAllowed);
125          ....
126          // Update the claimed tokens for the beneficiary
127      @>  data_account.beneficiaries[index].claimed_tokens += amount_to_transfer;
```

The `months_passed` variable represents the number of months since the lockup period ended. However, the formula used at line 116 to calculate the total unlockable tokens, `unlockable_tokens`, does not take `months_passed` into account. As a result, the total unlockable tokens always equal the value of `additional_tokens_first_month` when calculating the unlockable `LUX` assets.

After the beneficiary claims the first month's unlockable assets by invoking the `claim()` instruction, `beneficiary.claimed_tokens` is set to `additional_tokens_first_month`.

Consequently, the `amount_to_transfer`, calculated by `unlockable_tokens.saturating_sub(beneficiary.claimed_tokens);`, always equals zero, and the boolean expression

`amount_to_transfer > 0` always returns false after the beneficiary invokes the `claim()` instruction for the first time.

As a result, the beneficiary can only receive the unlockable assets for the first month.

Proof of Concept

Below is an example test code in Js for the described finding:

```
  it("Test Second Claim", async () => {
    try {

      const claimTx1 = await program.methods.claimLux(dataBump,
escrowBump).accounts({
        dataAccount: dataAccount,
        escrowWallet: escrowWallet,
        sender: beneficiary.publicKey,
        tokenMint: mintAddress,
        walletToDepositTo: beneficiaryATA,
        associatedTokenProgram: spl.ASSOCIATED_TOKEN_PROGRAM_ID,
        tokenProgram: spl.TOKEN_PROGRAM_ID,
        systemProgram: anchor.web3.SystemProgram.programId
      }).signers([beneficiary]).rpc();

      console.log(`claim TX1: https://explorer.solana.com/tx/${claimTx1}?
cluster=custom`);

      const ONE_MONTH_IN_SECONDS = 2592000;
      await provider.connection.confirmTransaction(
        await provider.connection.transaction().setTimestamp(Date.now() +
ONE_MONTH_IN_SECONDS).sign(provider.wallet)
      );
      const claimTx2 = await program.methods.claimLux(dataBump,
escrowBump).accounts({
        dataAccount: dataAccount,
        escrowWallet: escrowWallet,
        sender: beneficiary.publicKey,
        tokenMint: mintAddress,
        walletToDepositTo: beneficiaryATA,
        associatedTokenProgram: spl.ASSOCIATED_TOKEN_PROGRAM_ID,
        tokenProgram: spl.TOKEN_PROGRAM_ID,
        systemProgram: anchor.web3.SystemProgram.programId
      }).signers([beneficiary]).rpc();

    } catch (err) {
      assert.equal(err instanceof AnchorError, true);
      assert.equal(err.error.errorCode.code, "ClaimNotAllowed");
    }
  });
```

```
✔ Test Second Claim
```

Recommendation

Implement the necessary changes to integrate the `months_passed` variable into the token claim calculation, ensuring users receive the correct rewards based on elapsed time.

Alleviation

[Lucia Team, 06/11/2024]: The team heeded the advice and resolved the issue in commit: 99d471ccf823819ab91f822aa1d85a265599259c.

LCD-04 | CENTRALIZATION RELATED RISKS AND UPGRADABILITY

Category	Severity	Location	Status
Centralization	**Major**	**programs/lucia_vesting/src/lib.rs (v1): 49**	**Acknowledged**

Description

In the contract **lucia_vesting**, the role **initializer** has authority over the functions shown below.

- function release_lucia_vesting()

Any compromise to the **initializer** account may allow a hacker to take advantage of this authority and update the state.

Also, 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

[Lucia Team, 06/11/2024]: The team acknowledged this issue.

[CertiK, 06/11/2024]: 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.

CDL-01 | INCOMPLETE PRODUCTION CODE

Category	Severity	Location	Status
Volatile Code	Medium	programs/lucia_vesting/src/lib.rs (v4): 25	Resolved

Description

In the new code commit `5317945d14acd3ef988b8ca0b819f0a8c6a8e7f7`, we observed that a portion of the code was commented out, which should be included within production codes.

programs/lucia_vesting/src/lib.rs

```
24          // // LCD - 05
25          // if data_account.is_initialized == 1 {
26          //     return Err(VestingError::AlreadyInitialized.into());
27          // }
28
29          // // // LCD - 04
30
// // data_account.time_lock_end = Clock::get()?.unix_timestamp + 48 * 60 * 60;
31
32
// // msg!("Initializing data account with amount: {}, decimals: {}", amount,
decimals);

33          // // msg!("Beneficiaries: {:?}", beneficiaries);
```

1. If `data_account.is_initialized` is commented out, then the issue described in **LCD-05** still exists.
2. If `data_account.time_lock_end = Clock::get()? .unix_timestamp + 48 * 60 * 60;` is commented out, then the function `release_lucia_vesting()` can be invoked immediately to update the `state` variable value.

Recommendation

Recommend incorporating the commented code into the production environment.

Alleviation

[Lucia Team, 06/17/2024]: The team heeded the advice and resolved the issue in commit: a237595328478b7171fed4ce1782a3b2011ab3ef.

LCD-03 | MISSING STATE VERIFICATION IN `claim_lux()` FUNCTION

Category	Severity	Location	Status
Logical Issue	Medium	programs/lucia_vesting/src/lib.rs (v1): 58	Resolved

Description

The `claim_lux()` function, designed for users to claim tokens, lacks any verification of the `state` variable. This variable, intended to mark the token claim status of the contract and updated by `release_lucia_vesting()`, is crucial for the correct execution of contract logic. Without proper checks, the `state` variable fails to serve its purpose.

lucia_vesting/src/lib.rs

```
 49      pub fn release_lucia_vesting(ctx: Context<Release>, _data_bump: u8, state:
u8) -> Result<()> {
 50          let data_account: &mut Account<DataAccount> = &mut ctx.accounts.
data_account;
 51
 52  @>      data_account.state = state;
 53          msg!("Vesting Start - state : {}", state);
 54
 55          Ok(())
 56      }
```

Proof of Concept

Below is an example test code in Js for the described finding:

```
it("Test Claim", async () => {
  dataAccount = _dataAccountAfterRelease;
  const claimTx = await program.methods.claimLux(dataBump, escrowBump).accounts({
    dataAccount: dataAccount,
    escrowWallet: escrowWallet,
    sender: beneficiary.publicKey,
    tokenMint: mintAddress,
    walletToDepositTo: beneficiaryATA,
    associatedTokenProgram: spl.ASSOCIATED_TOKEN_PROGRAM_ID,
    tokenProgram: spl.TOKEN_PROGRAM_ID,
    systemProgram: anchor.web3.SystemProgram.programId
  }).signers([beneficiary]).rpc();

  await provider.connection.confirmTransaction(claimTx);
  const beneficiaryBalance = await getTokenBalanceWeb3(beneficiaryATA, provider);
  console.log('Balance:', beneficiaryBalance.toString());
});
```

```
Balance: 18333333
   ✔ Test Claim
```

Recommendation

Recommend refactoring the code to check the `state` variable in the `claim_lux()` function, ensuring the `data_account.state` is validated before allowing token claims.

Alleviation

[Lucia Team, 06/11/2024]: The team heeded the advice and resolved the issue in commit: 99d471ccf823819ab91f822aa1d85a265599259c.

LCD-05 | UNRESTRICTED ACCESS TO `initialize()` FUNCTION

Category	Severity	Location	Status
Logical Issue	Medium	programs/lucia_vesting/src/lib.rs (v1): 11	Resolved

Description

lucia_vesting/src/lib.rs

```
 11      pub fn initialize(
 12          ctx: Context<Initialize>,
 13          beneficiaries: Vec<Beneficiary>,
 14          amount: u64,
 15          decimals: u8
 16      ) -> Result<()> {
 17          let data_account: &mut Account<DataAccount> = &mut ctx.accounts.
data_account;
 18
 19          msg!("Initializing data account with amount: {}, decimals: {}", amount,
 decimals);
 20          msg!("Beneficiaries: {:?}", beneficiaries);
 21
 22          data_account.beneficiaries = beneficiaries;
 23          data_account.state = 0;
 24          data_account.token_amount = amount;
 25          data_account.decimals = decimals; // b/c bpf does not have any floats
 26          data_account.initializer = ctx.accounts.sender.to_account_info().key();
 27          data_account.escrow_wallet = ctx.accounts.escrow_wallet.to_account_info
().key();
 28          data_account.token_mint = ctx.accounts.token_mint.to_account_info().key
();
```

The `initialize()` function is designed to set up the initial program state. However, there is a critical flaw in the current implementation: any user can call `initialize()` without restrictions, which could result in multiple initializations. Additionally, any user can use the `lucia_vesting` program to issue their own token assets because there are no restrictions on the `token_mint` account.

lucia_vesting/src/lib.rs

```
165 #[derive(Accounts)]
166 pub struct Initialize<'info> {
167     #[account(
168         init,
169         payer = sender,
170         space = 8 + 1 + 8 + 32 + 32 + 32 + 8 + 1 + (4 + 50 * (32 + 8 + 8 + 8 +
4 + 8) + 1), // Can take 50 accounts to vest to
171         seeds = [b"data_account", token_mint.key().as_ref()],
172         bump
173     )]
174     pub data_account: Account<'info, DataAccount>,
175
176     #[account(
177         init,
178         payer = sender,
179         seeds = [b"escrow_wallet".as_ref(), token_mint.key().as_ref()],
180         bump,
181         token::mint = token_mint,
182         token::authority = data_account
183     )]
184     pub escrow_wallet: Account<'info, TokenAccount>,
185
186     #[account(
187         mut,
188         constraint=wallet_to_withdraw_from.owner == sender.key(),
189         constraint=wallet_to_withdraw_from.mint == token_mint.key()
190     )]
191     pub wallet_to_withdraw_from: Account<'info, TokenAccount>,
192
193 @>  pub token_mint: Account<'info, Mint>,
194
195     #[account(mut)]
196     pub sender: Signer<'info>,
197
198     pub system_program: Program<'info, System>,
199
200     pub token_program: Program<'info, Token>,
201 }
```

Within the `Initialize` instruction:

1. The addresses for `data_account` and `escrow_wallet` are derived from the `token_mint` address. Different `token_mint` addresses result in different `data_account` and `escrow_wallet` accounts.
2. The `wallet_to_withdraw_from` account is created outside the `lucia_vesting` program and is provided by the instruction signer.

Based on the above description, all accounts within the `Initialize` instruction can be customized by the instruction signer, the `sender`. Consequently, any user can use the `lucia_vesting` program to issue their own token assets.

Recommendation

Recommend refactoring codes to ensure that program can only be initialized once time and only correct `token_mint` is used.

Alleviation

[Lucia Team, 06/14/2024]: The team heeded the advice and resolved the issue in commit: 6f3465487f1adc22f7a06ef63973024799c2e803.

LCD-06 UNRESTRICTED REWARD CLAIMS IN `claim_lux()` FUNCTION

Category	Severity	Location	Status
Logical Issue	Medium	programs/lucia_vesting/src/lib.rs (v1): 116	Resolved

Description

In the smart contract, the `claim_lux()` function is responsible for distributing monthly rewards to users. A security review of the code revealed that at line 102, the function calculates how many months have passed but does not enforce a cap on the number of months for which rewards can be claimed. Consequently, without this check, users might be able to claim rewards for more than the intended 12-month period, leading to potential exploitation and undesired depletion of reward funds.

lucia_vesting/src/lib.rs

```
101         // Calculate the number of months since lockup ended
102         let months_passed = (time_since_lockup_end / 2592000) as f64;
// Assuming unlock_duration is in seconds
103         msg!("months_passed : {}", months_passed);
104
105         // Calculate the unlockable tokens for each month
106         let tokens_per_month = (beneficiary.allocated_tokens as f64) / 12.0;
107         msg!("tokens_per_month : {}", tokens_per_month);
108
```

Recommendation

Implement a check in the `claim_lux()` function to ensure that the total number of reward months claimed by a user does not exceed 12.

Alleviation

[Lucia Team, 06/11/2024]: The team heeded the advice and resolved the issue in commit: 99d471ccf823819ab91f822aa1d85a265599259c.

LCD-11 | INCORRECT SPACE SIZE CONSTRAINT FOR `data_account`

Category	Severity	Location	Status
Volatile Code	Medium	programs/lucia_vesting/src/lib.rs (v1): 170	Resolved

Description

The `DataAccount` account storage space size is composed of three parts:

1. The account discriminator size, which is 8 bytes for accounts owned by anchor programs.
2. The size of the `DataAccount` struct instance, excluding the `Vec<Beneficiary>`, is calculated as $1 + 8 + 32 + 32 + 32 + 8 + 1 + 1 + 1 = 116$ bytes.
3. The size of the `Vec<Beneficiary>` instance, which contains 50 items, is calculated as $4 + 50 \times (32 + 8 + 8 + 4 + 8 + 8 + 8 + 1) = 3854$ bytes.

Therefore, the total storage space for the `DataAccount` account is $8 + 116 + 3854 = 3978$ bytes.

```
298 #[derive(Default, Copy, Clone, AnchorSerialize, AnchorDeserialize, Debug)]
299 pub struct Beneficiary {
300     pub key: Pubkey, // Beneficiary's public key  32bytes
301     pub allocated_tokens: u64, // Tokens allocated to the beneficiary  8bytes
302     pub claimed_tokens: u64, // Tokens claimed by the beneficiary  8bytes
303     pub unlock_tge: f32,
// Unlock percentage at TGE (Token Generation Event)  4bytes
304     pub lockup_period: i64, // Lockup period in seconds  8bytes
305     pub unlock_duration: u64, // Unlock duration in seconds  8bytes
306     pub vesting_end_month: u64, // Vesting end month  8bytes
307     pub confirm_round: u8, // Confirmation round  1byte
308 }
309
310 #[account]
311 #[derive(Default)]
312 pub struct DataAccount {
313     pub state: u8, // State of the vesting contract
314     pub token_amount: u64, // Total token amount
315     pub initializer: Pubkey, // Public key of the initializer
316     pub escrow_wallet: Pubkey, // Public key of the escrow wallet
317     pub token_mint: Pubkey, // Public key of the token mint
318     pub initialized_at: u64, // Initialization timestamp
319     pub beneficiaries: Vec<Beneficiary>, // List of beneficiaries
320     pub decimals: u8, // Token decimals
321     pub is_initialized: u8, // Flag to check if account is initialized
322     pub contract_end_month: u8, // Contract end month
323 }
```

However, the space constraint for `data_account` equals 3527 bytes which is less than actually needed 3978 bytes space.

lucia_vesting/src/lib.rs

```
202  pub struct Initialize<'info> {
203      #[account(
204          init,
205          payer = sender,
206          space = 8 + 1 + 8 + 32 + 32 + 32 + 8 + 1 + (4 + 50 * (32 + 8 + 8 + 8 +
4 + 8) + 1),
207          seeds = [b"data_account", token_mint.key().as_ref()],
208          bump
209      )]
210      pub data_account: Account<'info, DataAccount>,
// Data account to initialize
```

Recommendation

Review and adjust the account storage allocation to ensure it is sufficient.

Alleviation

[Lucia Team, 06/16/2024]: The team heeded the advice and resolved the issue in commit: da68c4ef69c5abd69003afacc1406a0d702c8cc9.

LIB-01 | INVALID AMOUNT CHECK CONDITION

Category	Severity	Location	Status
Coding Issue	Medium	programs/lucia_vesting/src/lib.rs (v5): 45~47, 86~89	Resolved

Description

The following code checks if the vesting assets within `wallet` are sufficient. However, the current issue is that the boolean expression at line 45 always returns false, making its validation ineffective.

```
if amount > ctx.accounts.wallet_to_withdraw_from.amount {
    return Err(VestingError::InsufficientFunds.into());
}
```

Based on the transfer code below, we know that `amount` does not include decimals because the `transfer` function requires the transferred amount to include decimals. In contrast, the value returned by `ctx.accounts.wallet_to_withdraw_from.amount` includes decimals.

```
data_account.token_amount = amount;
...
token::transfer(
    cpi_ctx,
    data_account.token_amount * u64::pow(10, decimals as u32),
)?;
```

It is highly likely that the `amount`, which lacks decimals, is less than `ctx.accounts.wallet_to_withdraw_from.amount`, which includes decimals. As a result, the check at line 45 is invalid.

Recommendation

Recommend refactoring the code to avoid invalid amount check conditions. A potential solution is to pass the `amount` value as a decimal argument and replace the following code:

```
86          token::transfer(
87              cpi_ctx,
88              data_account.token_amount * u64::pow(10, decimals as u32),
89          )?;
```

with

```
86          token::transfer(
87              cpi_ctx,
88              data_account.token_amount,
89          )?;
90  ```.
```

Alleviation

[Lucia Team, 06/25/2024]: The team heeded the advice and resolved the issue in commit: c1996e20ac0e6d8900f30c5af83702df011397a6.

CAL-01 | LACK OF CHECK FOR `confirm_round`

Category	Severity	Location	Status
Logical Issue	Minor	programs/lucia_vesting/src/calculate.rs (v4): 8	Resolved

Description

programs/lucia_vesting/src/calculate.rs

```
 2  pub fn calculate_schedule(
 3      start_time: i64,
 4      vesting_end_month: i64,
 5      unlock_duration: i64,
 6      allocated_tokens: i64,
 7      unlock_tge: f32,
 8      confirm_round: u8
 9  ) -> Vec<(String, i64, f64, f64)> {
10      let mut schedule = Vec::new();
11      // LCD - 11
12   @> let start_round = 1;
13
14      ......
15
16   @> for i in start_round..=vesting_end_month {
17          ....
18   @>     if (confirm_round as i64) == i {
19              schedule.push(schedule_item);
20          }
21      }
```

The new code commit `5317945d14acd3ef988b8ca0b819f0a8c6a8e7f7` introduces a variable `start_round` which is set to 1. In the for loop on line 37, the variable `i` starts from `start_round` . For the condition on line 53, `if (confirm_round as i64) == i` , to be satisfied, the variable `confirm_round` must also be initialized to 1.

If the condition on line 53 is never met, the `schedule_item` will not be added to the schedule vector, resulting in the `schedule` variable being empty. Consequently, the `can_claim` variable in the `lib.rs` file will always be false, causing an error in the `claim_lux` function and preventing users from claiming LUX assets.

programs/lucia_vesting/src/lib.rs

```
141     @>   let schedule = calculate_schedule(
142              lockup_end_time,
143              vesting_end_month as i64,
144              beneficiary.unlock_duration as i64,
145              allocated_tokens as i64,
146              unlock_tge,
147              confirm_round
148          );
149
150          ...
151
152     @>   for item in schedule {
153              let round_num = item.0.split(": ").nth(1).unwrap().parse::<u64>().
unwrap();
154
155
// Check if the current time is greater than or equal to the unlock time and
round_num is valid

156              if current_time >= item.1 && (confirm_round as u64) <= round_num {
157                  ...
158
159      @>          can_claim = true;
160              } else {
161                  ...
162              }
163              ...
164          }
165
166      @> if !can_claim {
167              return Err(VestingError::ClaimNotAllowed.into());
168          }
```

Recommendation

Recommend adding a check to confirm that `confirm_round` is initialized to 1.

Alleviation

[Lucia Team, 06/18/2024]: The team heeded the advice and resolved the issue in commit: 69edcbde64ea6b1d831d3ef362f3648562e55391.

LCD-07 | INADEQUATE INITIALIZATION CHECKS

Category	Severity	Location	Status
Logical Issue	Minor	programs/lucia_vesting/src/lib.rs (v1): 11	Partially Resolved

Description

The contract's `initialize()` function is crucial for setting initial state variables, yet it does not adequately validate the supplied parameters.

1. The absence of a check to confirm that the `token_mint` decimals align with the input parameter `decimals` can lead to incorrect token transfer amounts.
2. The contract does not verify that the `token_amount` is sufficient for user claims, which could result in either an excess, causing tokens to be locked within the contract, or a deficit, preventing users from claiming their due tokens.
3. The function does not restrict the `beneficiaries` array to 50 entries, potentially causing unexpected behavior.

lucia_vesting/src/lib.rs

```
 11      pub fn initialize(
 12          ctx: Context<Initialize>,
 13          beneficiaries: Vec<Beneficiary>,
 14          amount: u64,
 15          decimals: u8
 16      ) -> Result<()> {
 17          let data_account: &mut Account<DataAccount> = &mut ctx.accounts.
data_account;
 18
 19          msg!("Initializing data account with amount: {}, decimals: {}", amount,
 decimals);
 20          msg!("Beneficiaries: {:?}", beneficiaries);
 21
 22  @>      data_account.beneficiaries = beneficiaries;
 23          data_account.state = 0;
 24  @>      data_account.token_amount = amount;
 25  @>      data_account.decimals = decimals; // b/c bpf does not have any floats
 26          data_account.initializer = ctx.accounts.sender.to_account_info().key();
 27          data_account.escrow_wallet = ctx.accounts.escrow_wallet.to_account_info
().key();
 28          data_account.token_mint = ctx.accounts.token_mint.to_account_info().key
();
```

Recommendation

Update the `initialize()` function to include validation checks that confirm `token_mint` decimals are as expected, `token_amount` is enough to cover claims, and the `beneficiaries` array does not exceed 50 entries.

Alleviation

[Lucia Team, 06/27/2024]: This part is based on Lucia tokenomics, which limits the total issuance to 1 billion tokens and allocates initial shares to beneficiary accounts based on their stake in the tokenomics. Therefore, there should be no instances of exceeding this limit.

[CertiK, 06/27/2024]: The team heeded the advice and partially resolved the issue in commit: 99d471ccf823819ab91f822aa1d85a265599259c.

The audit team recommends adding code to verify that the amount received by the `escrow_wallet` equals the sum of all allocated tokens for the beneficiaries. If the received amount exceeds the total allocated amount, any leftover vesting assets will remain locked in the `escrow_wallet`. This is because the assets in the `escrow_wallet` can only be transferred by the `data_account` PDA, which cannot sign transactions on its own.

LCD-08 | OUT-OF-SCOPE DEPENDENCIES

Category	Severity	Location	Status
Volatile Code	Minor	programs/lucia_vesting/src/lib.rs (v1): 28	Acknowledged

Description

The contract is serving as the underlying entity to interact with the `token_mint` contract, which is an out-of-scope dependency. The scope of the audit treats out-of-scope entities as black boxes and assumes their functional correctness. However, in the real world, those dependencies can be compromised and this may lead to lost or stolen assets.

Recommendation

We recommend that the project team constantly monitor the functionality of the out-of-scope contracts to mitigate any side effects that may occur when unexpected changes are introduced.

Alleviation

[Lucia Team, 06/27/2024]: Issue acknowledged. We will continuously monitor the contract.

LCD-09 | UNSAFE INTEGER CAST

Category	Severity	Location	Status
Incorrect Calculation	Minor	programs/lucia_vesting/src/lib.rs (v1): 116	Resolved

Description

Type casting refers to changing an variable of one data type into another. The code contains an unsafe cast between integer types, which may result in unexpected truncation or sign flipping of the value.

The `tokens_per_month` and `additional_tokens_first_month` are both `f64` data type.

lucia_vesting/src/lib.rs

```
116         unlockable_tokens = (tokens_per_month + additional_tokens_first_month)
as u64;
```

Recommendation

Recommended to check the bounds of integer values before casting.

Alleviation

[Lucia Team, 06/14/2024]: The team heeded the advice and resolved the issue in commit: 6f3465487f1adc22f7a06ef63973024799c2e803.

CDL-02 POTENTIAL FRONT-RUNNING RISK WITH INITIALIZE INSTRUCTIONS

Category	Severity	Location	Status
Logical Issue, Volatile Code	Informational	programs/lucia_vesting/src/lib.rs (v4): 16	Acknowledged

Description

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

However, a malicious party can invoke both `initialize` instructions, which will affect both programs.

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

[Lucia Team, 06/17/2024]: The team acknowledged the finding and decided not to change the current codebase.

OPTIMIZATIONS | LUCIA - AUDIT

ID	Title	Category	Severity	Status
LCD-10	Unnecessary Use Of `as_ref()`	Volatile Code	Optimization	Resolved
LCL-01	Redundant Code Execution In `calculate_schedule()`	Volatile Code	Optimization	Resolved
LCL-02	Unnecessary Conversion Between UTC And UnixTimeStamp	Code Optimization	Optimization	Resolved
LDL-01	Excessive Account Space Allocation	Storage Optimization	Optimization	Resolved

LCD-10 | UNNECESSARY USE OF `as_ref()`

Category	Severity	Location	Status
Volatile Code	Optimization	programs/lucia_vesting/src/lib.rs (v1): 234	Resolved

Description

In the smart contract's `Claim` struct, the seed value is derived using `b"escrow_wallet".as_ref()` . This is a redundant operation because the byte string literal `b"escrow_wallet"` is already a byte array, which is the expected format for a seed value. Using `.as_ref()` on a byte string literal adds an unnecessary layer of complexity and can lead to potential misunderstandings in the contract's execution.

lucia_vesting/src/lib.rs

```
232      #[account(
233          mut,
234  @>      seeds = [b"escrow_wallet".as_ref(), token_mint.key().as_ref()],
235          bump = wallet_bump,
236      )]
237      pub escrow_wallet: Account<'info, TokenAccount>,
```

Recommendation

Remove the unnecessary `.as_ref()` conversion and use the byte string literal `b"escrow_wallet"` as the seed value to simplify the code and enhance clarity.

Alleviation

[Lucia Team, 06/11/2024]: The team heeded the advice and resolved the issue in commit: 99d471ccf823819ab91f822aa1d85a265599259c.

LCL-01 | REDUNDANT CODE EXECUTION IN `calculate_schedule()`

Category	Severity	Location	Status
Volatile Code	Optimization	programs/lucia_vesting/src/calculate.rs (v2): 22~30	Resolved

Description

The `claimable_token` variable represents the fixed reward for a single month. The current code issue is that this fixed value is repeatedly calculated multiple times.

programs/lucia_vesting/src/calculate.rs

```
 22   for i in start_round..vesting_end_month + 1 {
 23           ....
 24
 25           // LCD - 09
 26
// Ensure allocated_tokens and vesting_end_month are positive to avoid unexpected
behavior

 27           if allocated_tokens < 0 || vesting_end_month <= 0 {
 28               panic!("Invalid allocated_tokens or vesting_end_month");
 29           }
 30
 31           // Check for overflow before casting
 32    @>     let claimable_token = (allocated_tokens as f64) / (vesting_end_month as
 f64);
 33           if claimable_token < 0.0 || claimable_token.is_infinite() ||
 claimable_token.is_nan() {
 34               panic!("Invalid claimable_token calculated");
 35           }
 36           ....
 37       }
```

The value of the `claimable_token` variable, determined by `allocated_tokens` and `vesting_end_month`, is fixed after the `lucia_vesting` program initialization. However, within the for-loop, this value is recalculated repeatedly, leading to unnecessary gas consumption.

Recommendation

Optimize the `calculate_schedule()` function by moving the repeated calculation outside of the for-loop to save on gas costs.

Alleviation

[Lucia Team, 06/14/2024]: The team heeded the advice and resolved the issue in commit: 6f3465487f1adc22f7a06ef63973024799c2e803.

LCL-02 UNNECESSARY CONVERSION BETWEEN UTC AND UNIXTIMESTAMP

Category	Severity	Location	Status
Code Optimization	Optimization	programs/lucia_vesting/src/calculate.rs (v2): 17~18, 33	Resolved

Description

Calculating the reward for a single month is based on the end time of a complete month, which is measured in Unix timestamp (seconds), starting from the lockup end time, `start_time`. The current issue with the code is that the conversion between UTC and Unix Timestamp is redundant.

programs/lucia_vesting/src/calculate.rs

```
  4  pub fn calculate_schedule(
  5      start_time: i64,
  6      vesting_end_month: i64,
  7      unlock_duration: i64,
  8      allocated_tokens: i64,
  9      confirm_round: u8
 10  ) -> Vec<(String, i64, f64)> {
 11      ...
 12  @>  let start_date = Utc.timestamp_opt(start_time, 0).single().expect(
"Invalid timestamp");
 13
 14      ...
 15
 16      for i in start_round..vesting_end_month + 1 {
 17  @>      let unlock_time =
 18              start_date + Duration::seconds((unlock_duration * (i as i64)) /
 vesting_end_month);
 19
 20          ....
 21  @>      let time_round = unlock_time.timestamp();
 22          let schedule_item = (claim_token_round, time_round, claimable_token as
f64);
 23
 24          schedule.push(schedule_item);
 25      }
 26      return schedule;
 27  }
```

`unlock_time`, the end time of a single month, is ultimately converted to a Unix timestamp. And we can directly use `start_time + (unlock_duration * (i as i64)) / vesting_end_month` to calculate `unlock_time`. Therefore, it is unnecessary to convert the Unix timestamp `start_time` to the UTC `start_date` at the beginning.

Recommendation

Recommend refactoring the code to avoid redundant code for converting Unix timestamps and UTC.

Alleviation

[Lucia Team, 06/14/2024]: The team heeded the advice and resolved the issue in commit: 6f3465487f1adc22f7a06ef63973024799c2e803.

LDL-01 | EXCESSIVE ACCOUNT SPACE ALLOCATION

Category	Severity	Location	Status
Storage Optimization	Optimization	programs/lucia_vesting/src/lib.rs (v2): 303, 323	Resolved

Description

The two fields, `unlock_tge` (4 bytes) and `contract_end_month` (1 byte), are not used in the codebase. However, when creating a `DataAccount`, an additional 1 byte + 50 * 4 bytes = 201 bytes of storage space, which will never be accessed, is allocated.

programs/lucia_vesting/src/lib.rs

```
297 // Struct to represent each beneficiary
298 #[derive(Default, Copy, Clone, AnchorSerialize, AnchorDeserialize, Debug)]
299 pub struct Beneficiary {
300     ....
301     pub unlock_tge: f32, // Unlock percentage at TGE (Token Generation Event)
302     ....
303 }
304
305 // Struct to represent the data account
306 #[account]
307 #[derive(Default)]
308 pub struct DataAccount {
309     ....
310     pub contract_end_month: u8, // Contract end month
311 }
```

Recommendation

Recommend removing unnecessary fields from account data to optimize storage space usage.

Alleviation

[Lucia Team, 06/14/2024]: The team heeded the advice and resolved the issue in commit: 6f3465487f1adc22f7a06ef63973024799c2e803.

APPENDIX | LUCIA - AUDIT

Finding Categories

Categories	Description
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
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.

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.

CertiK | Securing the Web3 World

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.