Security Assessment Report

Helium Program Library

Solana Smart Contracts

March 19, 2023

Summary

The sec3 team (formerly Soteria) was engaged to do a thorough security analysis of the helium-program-library Solana smart contract programs. The artifact of the audit was the source code of the smart contracts excluding tests in https://github.com/helium/helium-program-library.

The initial audit was done on commit ed3f1e71a5fcad6897bfba01538b755130d6f893 of the following programs and revealed 25 issues or questions.

- Program “circuit-breaker”
- Program “data-credits”
- Program “helium-entity-manager”
- Program “helium-sub-daos”
- Program “lazy-distributor”
- Program “lazy-transactions”
- Program “shared-utils”
- Program “treasury-management”
- Program “voter-stake-registry”

The post-audit review was done on commit 5a1567323c5b665112244dff62f1e7704be6fbd8 to check if the reported issues have been addressed.

This report describes the findings and resolutions in detail.

Table of Contents

Methodology and Scope of Work

The sec3 (formerly Soteria) audit team, which consists of Computer Science professors and industrial researchers with extensive experience in Solana smart contract security, program analysis, testing, and formal verification, performed a comprehensive manual code review, static analysis, and penetration testing.

Assisted by the sec3 Scanner developed in-house, the audit team particularly focused on the following work items:

- Check common security issues.
    - Missing ownership checks
    - Missing signer checks
    - Signed invocation of unverified programs
    - Solana account confusions
    - Arithmetic over- or underflows
    - Numerical precision errors
    - Loss of precision in calculation
    - Insufficient SPL-Token account verification
    - Missing rent exemption assertion
    - Casting truncation
    - Did not follow security best practices
    - Outdated dependencies
    - Redundant code
    - Unsafe Rust code
- Check program logic implementation against available design specifications.
- Check poor coding practices and unsafe behavior.
- The soundness of the economics design and algorithm is out of scope of this work

Result Overview

Issue	Impact	Status
1. CIRCUIT-BREAKER		
[L-1-1] Token account close authority not cleared	Low	Resolved
[I-1-1] Missing argument check	Informational	Resolved
[I-1-2] Inconsistent Anchor discriminator and padding space	Informational	Resolved
2. TREASURY-MANAGEMENT		
[I-2-1] Block the creation of treasury_management (DoS)	Informational	Resolved
[I-2-2] Validate mint precision	Informational	Resolved
3. SHARED-UTILS		
[I-3-1] Floor of negatives	Informational	Resolved
4. LAZY-TRANSACTIONS		
[I-4-1] Miss max_depth range check	Informational	Resolved
5. VOTER-STAKE-REGISTRY		
[H-5-1] Incomplete transfer_v0	High	Resolved
[H-5-2] Self-transfer accounting	High	Resolved
[H-5-3] Permissionless update_voter_weight_record_v0	High	Resolved
[H-5-4] Manipulate "num_active_votes" in position	High	Resolved
[M-5-1] Nonfunctional unique NFT mint checker	Medium	Resolved
[M-5-2] Multiple position authorities	Medium	Resolved
[I-5-1] Unused accounts in context	Informational	Resolved
6. LAZY-DISTRIBUTOR		
[C-6-1] Unchecked reward receiver in distribute_compression_rewards_v0	Critical	Resolved
[M-6-1] Validate recipient current_config_version when distributing rewards	Medium	Resolved
[L-6-1] Argument range check	Low	Resolved
7. HELIUM-SUB-DAOS		
[M-7-1] calculate_utility_score_v0 is permissionless and time-sensitive	Medium	Resolved
[M-7-2] Inconsistent formula implementation	Medium	Resolved
[L-7-1] Panic due to integer overflow	Low	Resolved
[I-7-1] Validate the input size	Informational	Resolved
8. DATA-CREDITS		
[M-8-1] Insufficient Pyth price feed validation	Medium	Resolved

9. HELIUM-ENTITY-MANAGER		
[L-9-1] Incorrect owner/delegate keys in compressed NFT validation	Low	Resolved
[I-9-1] Onboarding location payment and accounting	Informational	Resolved
[I-9-2] Improvements in iot and mobile updates	Informational	Resolved

Findings in Detail

CIRCUIT-BREAKER

[L-1-1] Token account close authority not cleared

The close authority of the token account is not cleared for non-native accounts. The circuit breaker cannot prevent the account being closed by the closure authority.

```
/* circuit-breaker/src/instructions/initialize_account_windowed_breaker_v0.rs */
038 | pub fn handler(
039 |   ctx: Context<InitializeAccountWindowedBreakerV0>,
040 |   args: InitializeAccountWindowedBreakerArgsV0,
041 | ) -> Result<()> {
057 |   set_authority(
065 |     AuthorityType::AccountOwner,
066 |     Some(ctx.accounts.circuit_breaker.key()),
067 |   )?;
070 | }

/* spl-token-3.5.0/src/processor.rs */
417 | pub fn process_set_authority(
422 | ) -> ProgramResult {
427 |     if account_info.data_len() == Account::get_packed_len() {
434 |         match authority_type {
435 |             AuthorityType::AccountOwner => {
443 |                 if let COption::Some(authority) = new_authority {
444 |                     account.owner = authority;
445 |                 } else {
446 |                     return Err(TokenError::InvalidInstruction.into());
447 |                 }
449 |                 account.delegate = COption::None;
450 |                 account.delegated_amount = 0;
452 |                 if account.is_native() {
453 |                     account.close_authority = COption::None;
454 |                 }
455 |             }
```

Resolution

The account closure authority has been transferred to circuit breaker too. This issue has been fixed.

CIRCUIT-BREAKER

[I-1-1] Missing argument check

config.window_size_seconds is used as the divisor and should not be zero. Otherwise, the contract will panic.

```
/* circuit-breaker/src/window.rs */
004 | pub fn time_decay_previous_value(
005 |   config: &WindowedCircuitBreakerConfigV0,
008 | ) -> Option<u64> {
010 |   u64::try_from(
020 |       .checked_div(u128::from(config.window_size_seconds))?,
```

Consider validating user-provided window_config.

The following is an example.

```
/* treasury-management/src/instructions/initialize_treasury_management_v0.rs */
058 | pub fn handler(
059 |   ctx: Context<InitializeTreasuryManagementV0>,
060 |   args: InitializeTreasuryManagementArgsV0,
061 | ) -> Result<()> {
062 |   initialize_account_windowed_breaker_v0(
079 |     InitializeAccountWindowedBreakerArgsV0 {
081 |       config: args.window_config.into(),
083 |     },
084 |   )?;
```

Resolution

The window size parameter needs to pass validations now. This issue has been fixed.

CIRCUIT-BREAKER

[I-1-2] Inconsistent Anchor discriminator and padding space

The space discriminator space 8 is inconsistently reserved.

```
/* circuit-breaker/src/instructions/initialize_mint_windowed_breaker_v0.rs */
016 | pub struct InitializeMintWindowedBreakerV0<'info> {
017 |   #[account(mut)]
018 |   pub payer: Signer<'info>,
019 |   #[account(
020 |     init,
021 |     payer = payer,
022 |     space = 60 + std::mem::size_of::<MintWindowedCircuitBreakerV0>(),
023 |     seeds = ["mint_windowed_breaker".as_bytes(), mint.key().as_ref()],
024 |     bump
025 |   )]
026 |   pub circuit_breaker: Box<Account<'info, MintWindowedCircuitBreakerV0>>,


/* treasury-management/src/instructions/initialize_treasury_management_v0.rs */
020 | #[derive(Accounts)]
021 | #[instruction(args: InitializeTreasuryManagementArgsV0)]
022 | pub struct InitializeTreasuryManagementV0<'info> {
023 |   #[account(mut)]
024 |   pub payer: Signer<'info>,
025 |   #[account(
026 |     init,
027 |     payer = payer,
028 |     space = 8 + std::mem::size_of::<TreasuryManagementV0>() + 60,
029 |     seeds = ["treasury_management".as_bytes(), supply_mint.key().as_ref()],
030 |     bump,
031 |   )]
032 |   pub treasury_management: Box<Account<'info, TreasuryManagementV0>>,
```

Resolution

This issue has been fixed.

TREASURY-MANAGEMENT

[I-2-1] Block the creation of treasury_management (DoS)

```
/* treasury-management/src/instructions/initialize_treasury_management_v0.rs */
022 | pub struct InitializeTreasuryManagementV0<'info> {
023 |   #[account(mut)]
024 |   pub payer: Signer<'info>,
025 |   #[account(
026 |     init,
027 |     payer = payer,
028 |     space = 8 + std::mem::size_of::<TreasuryManagementV0>() + 60,
029 |     seeds = ["treasury_management".as_bytes(), supply_mint.key().as_ref()],
030 |     bump,
031 |   )]
032 |   pub treasury_management: Box<Account<'info, TreasuryManagementV0>>,
033 |   // HNT
034 |   pub treasury_mint: Box<Account<'info, Mint>>,
035 |   // IOT, MOBILE, etc
036 |   pub supply_mint: Box<Account<'info, Mint>>,


/* treasury-management/src/state.rs */
018 | pub struct TreasuryManagementV0 {
019 |   pub treasury_mint: Pubkey,
020 |   pub supply_mint: Pubkey,
```

The PDA seeds of `treasury_management` only contain `supply_mint`. If someone creates the PDA with the correct `supply_mint` and a different `treasury_mint`, it's not possible to create the `treasury_management` for the `supply_mint` and HNT mint anymore.

Consider including `treasury_mint` in the PDA seeds for `treasury_management`, or requiring it to be the HNT mint.

Resolution

The mint authority of the supply mint has to sign, which will prevent others create the treasury management with an incorrect treasury mint. This issue has been fixed.

TREASURY-MANAGEMENT

[I-2-2] Validate mint precision

The precision of the user-provided mint should be smaller or equal to 12. It's better to reject user inputs than panic.

```
/* treasury-management/src/utils.rs */
049 | pub fn precise_supply_amt(amt: u64, mint: &Mint) -> PreciseNumber {
050 |   PreciseNumber {
051 |     value: InnerUint::from(amt)
052 |       .checked_mul(InnerUint::from(get_u128_pow_10(12_u8 - mint.decimals)))
053 |       .unwrap()
054 |       .checked_mul(InnerUint::from(1_000_000u64)) // Add 6 precision
055 |       .unwrap(),
056 |   }
057 | }
```

Resolution

The team stated that a panic is ok when mints have more than 12 decimals. In practice, this will not happen. Issue resolved.

SHARED-UTILS

[I-3-1] Floor of negatives

The common definition of floor(x) is the largest integer value less than or equal to x. When x is negative, it should be the negative of the ceiling. E.g. `floor(-2.1) = -3`.

```
/* shared-utils/src/signed_precise_number.rs */
147 | pub fn floor(&self) -> Option<SignedPreciseNumber> {
148 |   Some(Self {
149 |     value: self.value.floor()?,
150 |     is_negative: self.is_negative,
151 |   })
152 | }
```

However, this `floor()` is only called in `exp()`, which is modified from `e_exp.c`.

```
/* shared-utils/src/signed_precise_number.rs */
239 | k = INVLN2
240 |   .signed()
241 |   .checked_mul(self)?
242 |   .checked_add(&Self {
243 |     value: HALF,
244 |     is_negative: self.is_negative,
245 |   })?
246 |   .floor()?;
```

In `e_exp.c`, it is supposed to do the explicit int conversion.

```
/* https://github.com/JuliaMath/openlibm/blob/master/src/e_exp.c#L140 */
k  = (int)(invln2*x+halF[xsb]);
```

Although the `floor()` implantation doesn't follow the standard semantics of flooring negative numbers, it's consistent with the type cast in `e_exp.c`.

Therefore, the usage here is correct. If `floor()` will be used in other places in the future, it could be confusing.

Resolution

The team acknowledged the reminder. Issue resolved.

LAZY-TRANSACTIONS

[I-4-1] Missing max_depth argument check

The constraints between the user-provided max_depth and the size of the canopy is not checked. Once max_depth is set during initialization, it cannot be updated.

```
/* lazy-transactions/src/instructions/initialize_lazy_transactions_v0.rs */
004 | #[derive(AnchorSerialize, AnchorDeserialize, Clone, Default)]
005 | pub struct InitializeLazyTransactionsArgsV0 {
009 |   pub max_depth: u32,
010 | }
011 |
012 | #[derive(Accounts)]
013 | #[instruction(args: InitializeLazyTransactionsArgsV0)]
014 | pub struct InitializeLazyTransactionsV0<'info> {
025 |   /// CHECK: Account to store the canopy, the size will determine the size of the
canopy
026 |   #[account(
027 |     mut,
028 |     owner = id(),
029 |     constraint = canopy.data.borrow()[0] == 0,
030 |     constraint = check_canopy_bytes(&canopy.data.borrow()[1..]).is_ok(),
031 |   )]
032 |   pub canopy: AccountInfo<'info>,
034 | }
035 |
036 | pub fn handler(
037 |   ctx: Context<InitializeLazyTransactionsV0>,
038 |   args: InitializeLazyTransactionsArgsV0,
039 | ) -> Result<()> {


/* lazy-transactions/src/canopy.rs */
050 | if closest_power_of_2 > (1 << (max_depth + 1)) {
051 |   msg!(
052 |     "Canopy size is too large. Size: {}. Max size: {}",
053 |     closest_power_of_2 - 2,
054 |     (1 << (max_depth + 1)) - 2
055 |   );
```

Resolution

The team acknowledged the finding. It's not exploitable. This issue has been resolved.

VOTER-STAKE-REGISTRY

[H-5-1] Incomplete transfer_v0

In instruction `transfer_v0`, the implementation is incomplete.

The processor only makes changes to `amount_deposited_native` of the source and target positions. However, the actually token transfer is missing.

Resolution

The token transfer function has been added. This issue has been resolved.

VOTER-STAKE-REGISTRY

[H-5-2] Self-transfer accounting

```
/* voter-stake-registry/src/instructions/transfer_v0.rs */
010 | pub struct TransferV0<'info> {
019 |   #[account(
020 |     mut,
021 |     seeds = [b"position".as_ref(), mint.key().as_ref()],
022 |     bump = source_position.bump_seed,
023 |     constraint = source_position.num_active_votes == 0 @ VsrError::ActiveVotesExist,
024 |     has_one = registrar,
025 |     has_one = mint
026 |   )]
027 |   pub source_position: Box<Account<'info, PositionV0>>,

036 |   #[account(
037 |     mut,
038 |     has_one = registrar,
039 |   )]
040 |   pub target_position: Box<Account<'info, PositionV0>>,
056 | }


/* voter-stake-registry/src/instructions/transfer_v0.rs */
073 | pub fn handler(ctx: Context<TransferV0>, args: TransferArgsV0) -> Result<()> {
076 |   let source_position = &mut ctx.accounts.source_position;
077 |   let target_position = &mut ctx.accounts.target_position;
084 |   source_position.amount_deposited_native = source_position
085 |     .amount_deposited_native
086 |     .checked_sub(amount)
087 |     .unwrap();
119 |   // Add target amounts
120 |   target_position.amount_deposited_native = target_position
121 |     .amount_deposited_native
122 |     .checked_add(amount)
123 |     .unwrap();
124 |   Ok(())
125 | }
```

It's possible to use the same position account as the `source_position` and `target_position`.

Since `source_position.amount_deposited_native` and `target_position.amount_deposited_native` are two separate variables in the program, the accounting will be off.

When saving the accounts, the same position account will be written twice. As a result, the `amount_deposited_native` will be inconsistent instead of staying unchanged.

Poc

```
it("Attack move tokens to same position with a greater or equal lockup", async () => {
    const { position: newPos } = await createAndDeposit(10, 185);
    await program.methods
    .transferV0({ amount: toBN(10, 8) })
    .accounts({
        sourcePosition: position,
        targetPosition: position,  // Attack here!
        depositMint: hntMint,
    })
    .rpc({ skipPreflight: true });

    const newPosAcc = await program.account.positionV0.fetch(newPos);
    const oldPosAcc = await program.account.positionV0.fetch(position);
    expect(newPosAcc.amountDepositedNative.toNumber()).to.equal(
    toBN(10, 8).toNumber()
    );
    expect(oldPosAcc.amountDepositedNative.toNumber()).to.equal(
    toBN(110, 8).toNumber() // before-transfer: 100; correct after-transfer amount 100.
    );
});
```

Result

```
Program version (simulation) { major: 3, minor: 1, patch: 0 }
Latency
49RMkMu46EsfP68ZpQS2FgQzouTUBoF4F1icSaapZMhG4zTf65fPayYyFuwDSzwEToTSx4c1wURZMteEFAimQjq5
0.3810000419616699
Latency
4cMWhRuT8iUQ4zez5Usvhdt3ytgQDEvDJNMJ7ec3Rmi8NuxouqfBfthvMvBNz51z1Dgd9rESj8hTJr5CMSTaiLLM
0.3689999580383301
Latency
4zuYbiKdbF3X2MSgAwYgvW71nWoAfV2cC2YxAFN6hj5wo9FAVXVocHPY2SS9Xi3jHWpY7qrGXAeDuLAQerF6w9Cb
0.36299991607666016
      ✔ Attack move tokens to same position with a greater or equal lockup (817ms)
```

Resolution

The source and destination accounts have been checked. This issue has been resolved.

VOTER-STAKE-REGISTRY

[H-5-3] Permissionless update_voter_weight_record_v0

```
/* voter-stake-registry/src/instructions/update_voter_weight_record_v0.rs */
023 | pub struct UpdateVoterWeightRecordV0<'info> {
024 |   #[account(mut)]
025 |   pub payer: Signer<'info>,
026 |   pub registrar: Box<Account<'info, Registrar>>,
027 |
028 |   #[account(
029 |     init_if_needed,
030 |     payer = payer,
031 |     space = 8 + size_of::<VoterWeightRecord>(),
032 |     seeds = [registrar.key().as_ref(), b"voter-weight-record".as_ref(),
args.owner.as_ref()],
033 |     bump,
034 |   )]
035 |   pub voter_weight_record: Account<'info, VoterWeightRecord>,
036 |   pub system_program: Program<'info, System>,
037 | }
```

Instruction `update_voter_weight_record_v0` only requires a `payer` as the signer. Anyone may call this instruction and update the voter weight record on the behalf of `args.owner` (a public key). For example, it's possible to update an existing record with `weight_action` action to other actions and clear the target.

Resolution

The voter authority check has been added. This issue has been resolved.

VOTER-STAKE-REGISTRY

[H-5-4] Manipulate "num_active_votes" in position

The `position` accounts loaded from `remaining_accounts` are not sufficiently validated. Currently, it only checks the account was created by the `voter-state-registry` program and its type discriminator.

However, it doesn't check the associated mint to see if it's relevant to the `token_account`. Therefore, it's possible to provide other unrelated `position` accounts and manipulate the calculation.

```
/* voter-stake-registry/src/instructions/update_voter_weight_record_v0.rs */
062 | let mut voter_weight = 0u64;
067 | for (token_account, position) in ctx.remaining_accounts.iter().tuples() {
068 |   let nft_vote_weight = resolve_vote_weight(
069 |     registrar,
070 |     &governing_token_owner,
071 |     token_account,
072 |     position,
073 |     &mut unique_nft_mints,
074 |   )?;
075 |
076 |   voter_weight = voter_weight.checked_add(nft_vote_weight).unwrap();
077 | }
083 | voter_weight_record.voter_weight = voter_weight;
```

In particular, in `update_voter_weight_record_v0`, it may use incorrect voter mint configurations from other positions and manipulate the `voter_weight` result.

```
/* voter-stake-registry/src/instructions/cast_vote_v0.rs */
087 | for (token_account, position, nft_vote_record_info) in
ctx.remaining_accounts.iter().tuples() {
088 |   let nft_vote_weight = resolve_vote_weight(
089 |     registrar,
090 |     &args.owner,
091 |     token_account,
092 |     position,
093 |     &mut unique_nft_mints,
094 |   )?;
095 |
096 |   voter_weight = voter_weight.checked_add(nft_vote_weight).unwrap();
097 |
```

```
098 |   // Increase num active votes
099 |   let position_acc: &mut Account<PositionV0> = &mut Account::try_from(position)?;
100 |   position_acc.num_active_votes += 1;
101 |   position_acc.exit(&crate::ID)?;
102 |   require!(position.is_writable, VsrError::PositionNotWritable);
107 |   require!(
108 |     nft_vote_record_info.data_is_empty(),
109 |     VsrError::NftAlreadyVoted
110 |   );
138 | }
```

In `cast_vote_v0`, it may increase the `num_active_votes` of an unrelated position and affect the result of the `voter_weight`.

Resolution

The mint consistency check between the position and the token account has been added. This issue has been resolved.

VOTER-STAKE-REGISTRY

[M-5-1] Nonfunctional unique NFT mint checker

```
/* voter-stake-registry/src/util.rs */
009 | pub fn resolve_vote_weight(
010 |   registrar: &Registrar,
011 |   governing_token_owner: &Pubkey,
012 |   token_account: &AccountInfo,
013 |   position: &AccountInfo,
014 |   unique_nft_mints: &mut Vec<Pubkey>,
015 | ) -> Result<u64> {
033 |   require!(
034 |     !unique_nft_mints.contains(&token_account.key()),
035 |     VsrError::DuplicatedNftDetected
036 |   );
037 |
038 |   unique_nft_mints.push(token_account_acc.mint);
```

At line 34, `token_account` instead of `token_account_acc.mint` is incorrectly used to check for duplicated NFTs.

Resolution

Fixed. Now it checks the mint.

VOTER-STAKE-REGISTRY

[M-5-2] Multiple position authorities

The owner of the token minted from `mint` will be the authority of the position.

```
/* voter-stake-registry/src/instructions/initialize_position_v0.rs */
030 | #[derive(Accounts)]
031 | pub struct InitializePositionV0<'info> {
055 |   #[account(
056 |     init,
057 |     payer = payer,
058 |     seeds = [b"position".as_ref(), mint.key().as_ref()],
059 |     bump,
060 |     space = 8 + size_of::<PositionV0>() + 60,
061 |   )]
062 |   pub position: Box<Account<'info, PositionV0>>,
063 |   #[account(
064 |     mut,
065 |     mint::decimals = 0,
066 |     mint::authority = position,
067 |     mint::freeze_authority = position,
068 |   )]
069 |   pub mint: Box<Account<'info, Mint>>,


137 | pub fn handler(ctx: Context<InitializePositionV0>, args: InitializePositionArgsV0) ->
Result<()> {
169 |   token::mint_to(ctx.accounts.mint_ctx().with_signer(signer_seeds), 1)?;
171 |   token::freeze_account(ctx.accounts.freeze_ctx().with_signer(signer_seeds))?;
```

When creating the `position`, a token will be minted to `position_token_account` owned by the `recipient`. And then, the token account `position_token_account` will be frozen. The owner of the token will be the authority of the position.

However, the instruction doesn't check if `mint.supply` is `0`. If there are exiting tokens of `mint`, their owners will be the position authorities.

Consider the following scenario:

- The `mint` is created with non-PDA mint authority before initializing the position.
- Mint a few tokens

- Assign `position` as the mint authority
- Initialize the position and create the official position authority.

The setting of multiple position authorities can be confusing. The authorities who obtained the token before initializing the position can withdraw without approval from the authority created during the initialization.

Resolution

The mint supply validation was added. This issue has been resolved.

VOTER-STAKE-REGISTRY

[I-5-1] Unused accounts in context

collection_metadata, collection and metadata are not used in the context of close_position_v0.

```
/* voter-stake-registry/src/instructions/close_position_v0.rs */
007 | pub struct ClosePositionV0<'info> {
028 |   pub collection: Box<Account<'info, Mint>>,
029 |   /// CHECK: Handled by cpi
030 |   #[account(
031 |     mut,
032 |     seeds = ["metadata".as_bytes(), token_metadata_program.key().as_ref(),
collection.key().as_ref()],
033 |     seeds::program = token_metadata_program.key(),
034 |     bump,
035 |   )]
036 |   pub collection_metadata: UncheckedAccount<'info>,
037 |   #[account(mut)]
038 |   pub mint: Box<Account<'info, Mint>>,
039 |   #[account(
040 |     mut,
041 |     seeds = ["metadata".as_bytes(), token_metadata_program.key().as_ref(),
mint.key().as_ref()],
042 |     seeds::program = token_metadata_program.key(),
043 |     bump,
044 |   )]
045 |   /// CHECK: Checked by cpi
046 |   pub metadata: UncheckedAccount<'info>,
```

Resolution

Unused code snippets have been removed. This issue has been resolved.

LAZY-DISTRIBUTOR

[C-6-1] Unchecked reward receiver in distribute_compression_rewards_v0

```
/* lazy-distributor/src/instructions/distribute/distribute_compression_rewards_v0.rs */
032 | pub fn handler<'info>(
033 |   ctx: Context<'_, '_, '_, 'info, DistributeCompressionRewardsV0<'info>>,
034 |   args: DistributeCompressionRewardsArgsV0,
035 | ) -> Result<()> {
036 |   verify_compressed_nft(VerifyCompressedNftArgs {
037 |     hash: args.hash,
038 |     root: args.root,
039 |     index: args.index,
040 |     compression_program: ctx.accounts.compression_program.to_account_info(),
041 |     merkle_tree: ctx.accounts.merkle_tree.to_account_info(),
042 |     owner: ctx.accounts.common.owner.key(),
043 |     delegate: ctx.accounts.common.owner.key(),
044 |     proof_accounts: ctx.remaining_accounts.to_vec(),
045 |   })?;
047 |   require_eq!(
048 |     ctx.accounts.common.recipient.asset,
049 |     get_asset_id(&ctx.accounts.merkle_tree.key(), args.index.into()),
050 |     ErrorCode::InvalidAsset
051 |   );
054 | }

/* shared-utils/src/compressed_nfts.rs */
018 | pub fn verify_compressed_nft(args: VerifyCompressedNftArgs) -> Result<()> {
019 |   let verify_ctx = CpiContext::new(
020 |     args.compression_program,
021 |     VerifyLeaf {
022 |       merkle_tree: args.merkle_tree,
023 |     },
024 |   )
025 |   .with_remaining_accounts(args.proof_accounts);
027 |   verify_leaf(verify_ctx, args.root, args.hash, args.index)
028 | }
```

The reward receiver `ctx.accounts.common.owner` is not validated,

In particular, since this is a permissionless instruction, anyone may directly call it with any the user-provided `args.hash` and `ctx.accounts.common.owner`. However, via `verify_leaf()` , it

only checks if ctx.accounts.merkle_tree, args.hash, args.root and args.index are consistent and valid. It doesn't verify if the owner is indeed holding the NFT token.

As a result, it's possible to provide an unrelated account as the `ctx.accounts.common.owner` and steal rewards.

Resolution

The owner is verified using the Merkle Tree. This issue has been resolved.

LAZY-DISTRIBUTOR

[M-6-1] Validate recipient current_config_version in distribute

Similar to the behavior in `set_current_rewards_v0`, the current rewards should be cleared too when `lazy_distributor.version` and `recipient.current_config_version` are different.

Resolution

This issue has been resolved.

LAZY-DISTRIBUTOR

[L-6-1] Argument range check

```
/* lazy-distributor/src/instructions/set_current_rewards_v0.rs */
011 | #[derive(Accounts)]
012 | #[instruction(args: SetRewardsArgsV0)]
013 | pub struct SetRewardsV0<'info> {
022 |   #[account(
023 |     constraint = oracle.key() ==
lazy_distributor.oracles[usize::try_from(args.oracle_index).unwrap()].oracle
024 |   )]
025 |   pub oracle: Signer<'info>,
027 | }
028 |
029 | pub fn handler(ctx: Context<SetRewardsV0>, args: SetRewardsArgsV0) -> Result<()> {
036 |   ctx.accounts.recipient.current_rewards[usize::try_from(args.oracle_index).unwrap()]
=
037 |     Some(args.current_rewards);
046 | }
```

When `args.oracle_index` is larger than the size of `lazy_distributor.oracles`, the program will panic. It's better to validate `args.oracle_index` first.

Resolution

This issue has been resolved.

HELIUM-SUB-DAOS

[M-7-1] calculate_utility_score_v0 is permissionless and time-sensitive

calculate_utility_score_v0 can be invoked only once per epoch by checking the status of sub_dao_epoch_info.utility_score and is supposed to be permissionless.

However, the value of dao_epoch_info.total_rewards can be affected by when the instruction is called (Clock::get() at line 87), which is incompatible with the permissionless design.

```
/* helium-sub-daos/src/instructions/calculate_utility_score_v0.rs */
067 | pub fn handler(
068 |   ctx: Context<CalculateUtilityScoreV0>,
069 |   args: CalculateUtilityScoreArgsV0,
070 | ) -> Result<()> {
083 |   ctx.accounts.dao_epoch_info.total_rewards = ctx
084 |     .accounts
085 |     .dao
086 |     .emission_schedule
087 |     .get_emissions_at(Clock::get()?.unix_timestamp)
088 |     .unwrap()
089 |     .checked_add(std::cmp::min(
090 |       prev_supply.saturating_sub(curr_supply),
091 |       ctx.accounts.dao.net_emissions_cap,
092 |     ))
093 |     .unwrap();
```

Resolution

Fixed in PR #98.

HELIUM-SUB-DAOS

[M-7-2] Inconsistent formula implementation

The TWO_PREC and FOUR_PREC should be swapped.

```
/* helium-sub-daos/src/instructions/calculate_utility_score_v0.rs */
129 | // D = max(1, sqrt(DCs burned in USD)). 1 DC = $0.00001.
164 | // sqrt(x) = e^(ln(x)/2)
167 | let d = if epoch_info.dc_burned > 0 {
168 |   std::cmp::max(
169 |     one.clone(),
170 |     dc_burned
171 |       .log()
172 |       .or_arith_error()?
173 |       .checked_div(&FOUR_PREC.clone().signed())  // should be TWO_PREC
174 |       .or_arith_error()?
175 |       .exp()
176 |       .or_arith_error()?,
177 |   )
178 | } else {

130 | // A = max(1, fourth_root(Total active device count * device activation fee)).
165 | // x^1/4 = e^(ln(x)/4))
189 | let a = if total_devices_u64 > 0 {
190 |   std::cmp::max(
191 |     one,
192 |     devices_with_fee
193 |       .log()
194 |       .or_arith_error()?
195 |       .checked_div(&TWO_PREC.clone().signed())  // should be FOUR_PREC
196 |       .or_arith_error()?
197 |       .exp()
198 |       .or_arith_error()?,
199 |   )
200 | } else {
```

Resolution

This issue has been resolved.

HELIUM-SUB-DAOS

[L-7-1] Panic due to integer overflow

```
/* helium-program-library/programs/helium-sub-daos/src/state.rs */
059 | fn get_emissions_at(&self, unix_time: i64) -> Option<u64> {
060 |   if self.is_empty() {
061 |     return None;
062 |   }
063 |
064 |   let mut ans: Option<u64> = None;
065 |   let mut low: usize = 0;
066 |   let mut high: usize = self.len() - 1;
067 |
068 |   while low <= high {
069 |     let middle = (high + low) / 2;
070 |     if let Some(current) = self.get(middle) {
071 |       // Move to the right side if target time is greater
072 |       if current.start_unix_time <= unix_time {
073 |         ans = Some(current.emissions_per_epoch);
074 |         low = middle + 1;
075 |       } else {
076 |         // move left side
077 |         high = middle - 1;
078 |       }
079 |     } else {
080 |       break;
081 |     }
082 |   }
083 |
084 |   ans
085 | }
```

When there are 1 or 2 items in the vector, `middle` will be `0` so that `high = 0 - 1` will trigger a panic due to overflow checks.

Resolution

This issue has been resolved.

HELIUM-SUB-DAOS

[I-7-1] Validate the input size

```
/* helium-sub-daos/src/instructions/initialize_dao_v0.rs */
028 |  #[account(
029 |    init,
030 |    payer = payer,
031 |     space = 60 + 8 + std::mem::size_of::<DaoV0>() +
(std::mem::size_of::<EmissionScheduleItem>() * args.emission_schedule.len()),
032 |     seeds = ["dao".as_bytes(), hnt_mint.key().as_ref()],
033 |     bump,
034 |   )]
035 |   pub dao: Box<Account<'info, DaoV0>>,
```

The length of `emission_schedule` is from the user-provided `args` without validating its size. As a result, the space allocated for `dao` may exceed the PDA size limit. Consider adding a constraint on `args` to constrain the length of `emission_schedule`.

Similarly, the initialization of `sub_dao` could also be improved in `helium-sub-daos/src/instructions/initialize_sub_dao_v0.rs`

Resolution

The team acknowledged the finding. This issue has been resolved.

DATA-CREDITS

[M-8-1] Insufficient Pyth price feed validation

(1) The account owner is not checked. It's possible to use a faked account created by other programs. Since the feed is provided by the mint authority, this is unlikely.

```
/* data-credits/src/instructions/initialize_data_credits_v0.rs */
022 | pub struct InitializeDataCreditsV0<'info> {
031 |   /// CHECK: Checked via load call in handler
032 |   pub hnt_price_oracle: AccountInfo<'info>,
062 | }
063 |
064 | pub fn handler(
067 | ) -> Result<()> {
073 |   // Make sure these Pyth price accounts can be loaded
074 |   load_price_feed_from_account_info(&ctx.accounts.hnt_price_oracle).map_err(|e| {
077 |   })?;

/* pyth-sdk-solana-0.7.0/src/lib.rs */
030 | pub fn load_price_feed_from_account_info(
031 |     price_account_info: &AccountInfo,
032 | ) -> Result<PriceFeed, PythError> {
033 |     let data = price_account_info
034 |         .try_borrow_data()
035 |         .map_err(|_| PythError::InvalidAccountData)?;
036 |     let price_account = load_price_account(*data)?;
038 |     Ok(price_account.to_price_feed(price_account_info.key))
039 | }
```

(2) Suppose the price feed is owned by Pyth. It may be the feed for other tokens. Consider validating the feed against hard-coded HNT feed addresses.

(3) Additionally, checking the price confidence is recommended. Please see https://docs.pyth.network/pythnet-price-feeds/best-practices for more information.

Resolution

The owner check has been added. The confidence information was incorporated. This issue has been resolved.

HELIUM-ENTITY-MANAGER

[L-9-1] Incorrect owner/delegate keys in compressed NFT validation

`hotspot_owner.owner` should be `hotspot_owner`

```
/* helium-entity-manager/src/instructions/onboard_iot_hotspot_v0.rs */
130 | pub fn handler<'info>(
131 |   ctx: Context<'_, '_, '_, 'info, OnboardIotHotspotV0<'info>>,
132 |   args: OnboardIotHotspotArgsV0,
133 | ) -> Result<()> {
        ...
141 |     owner: ctx.accounts.hotspot_owner.owner.key(),
142 |     delegate: ctx.accounts.hotspot_owner.owner.key(),
        ...
144 |   })?;


/* helium-entity-manager/src/instructions/onboard_mobile_hotspot_v0.rs */
129 | pub fn handler<'info>(
130 |   ctx: Context<'_, '_, '_, 'info, OnboardMobileHotspotV0<'info>>,
131 |   args: OnboardMobileHotspotArgsV0,
132 | ) -> Result<()> {
      ...
138 |   verify_compressed_nft(VerifyCompressedNftArgs {
144 |     owner: ctx.accounts.hotspot_owner.owner.key(),
145 |     delegate: ctx.accounts.hotspot_owner.owner.key(),
147 |   })?;
      ...
```

Resolution

Fixed by PR #98

HELIUM-ENTITY-MANAGER

[I-9-1] Onboarding location payment and accounting

(1) `rewardable_entity_config` is not validated. It's possible to set location without paying when a `MobileConfig` (`rewardable_entity_config`) is provided but `IotConfig` is expected.

(2) `onboard_iot_hotspot_v0:165-170` should be done under the same condition when the location field is set.

(3) `onboard_mobile_hotspot_v0.rs:158` should be `ConfigSettingsV0::MobileConfig`

```
/* helium-entity-manager/src/instructions/onboard_iot_hotspot_v0.rs */
148 | ctx.accounts.iot_info.set_inner(IotHotspotInfoV0 {
151 |   location: args.location,
156 | });
158 | if let ConfigSettingsV0::IotConfig {
159 |   full_location_staking_fee,
161 | } = ctx.accounts.rewardable_entity_config.settings
162 | {
163 |   dc_fee = full_location_staking_fee.checked_add(dc_fee).unwrap();
165 |   ctx.accounts.iot_info.num_location_asserts = ctx
166 |     .accounts
167 |     .iot_info
168 |     .num_location_asserts
169 |     .checked_add(1)
171 | }

/* helium-entity-manager/src/instructions/onboard_mobile_hotspot_v0.rs */
158 | if let ConfigSettingsV0::IotConfig {
159 |   full_location_staking_fee,
161 | } = ctx.accounts.rewardable_entity_config.settings
162 | {
163 |   dc_fee = full_location_staking_fee.checked_add(dc_fee).unwrap();
165 |   ctx.accounts.mobile_info.num_location_asserts = ctx
166 |     .accounts
167 |     .mobile_info
168 |     .num_location_asserts
169 |     .checked_add(1)
171 | }
```

Resolution

The implementation has been refactored. The issues have been resolved.

HELIUM-ENTITY-MANAGER

[I-9-2] Improvements in iot and mobile updates

(1) If providing the same location in `update_iot_info_v0` and `update_mobile_info_v0`, it will do the updates and charge the fees. It seems better to reject such location updates without charging the fees.

(2) The iot hotspot owner can change the elevation and gain freely. It's not clear if this is the design as they are not used in the contracts.

Resolution

The implementation has been refactored. The issue with the same address updates has been resolved.

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a sec3 (the "Company") and Decentralized Wireless Foundation, Inc. d/b/a Helium Foundation (the "Client"). This Report solely includes the results of a technical assessment of a specific build and/or version of the Client's code specified in the Report ("Assessed Code") by the Company. The sole purpose of the Report is to provide the Client with the results of the technical assessment of the Assessed Code. The Report does not apply to any other version and/or build of the Assessed Code. Regardless of the contents of the Report, the Report does not (and should not be interpreted to) provide any warranty, representation or covenant that the Assessed Code: (i) is error and/or bug free, (ii) has no security vulnerabilities, and/or (iii) does not infringe any third-party rights. Moreover, the Report is not, and should not be considered, an endorsement by the Company of the Assessed Code and/or of the Client. Finally, the Report should not be considered investment advice or a recommendation to invest in the Assessed Code and/or the Client.

This Report is considered null and void if the Report (or any portion thereof) is altered in any manner.

ABOUT

Founded by leading academics in the field of software security and senior industrial veterans, sec3 (formerly Soteria) is a leading blockchain security company. We are also building sophisticated security tools that incorporate static analysis, penetration testing, and formal verification.

At sec3, we identify and eliminate security vulnerabilities through the most rigorous process and aided by the most advanced analysis tools.

For more information, check out our website and follow us on twitter.