ExVul

SMART CONTRACT AUDIT REPORT

TNTX solana Smart Contract

OCTOBER 2025

Contents

1. EXECUTIVE SUMMARY

ExVul Web3 Security was engaged by **TNTX solana** to review smart contract implementation. The assessment was conducted in accordance with our systematic approach to evaluate potential security issues based upon customer requirement. The report provides detailed recommendations to resolve the issue and provide additional suggestions or recommendations for improvement.

The outcome of the assessment outlined in chapter 3 provides the system's owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendations that will resolve the underlying technical issue.

1.1 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [10] which is the gold standard in risk assessment using the following risk models:

- **Likelihood**: represents how likely a particular vulnerability is to be uncovered and exploited in the wild.
- **Impact**: measures the technical loss and business damage of a successful attack.
- **Severity**: determine the overall criticality of the risk.

Likelihood can be: High, Medium and Low and impact are categorized into: High, Medium, Low, Informational. Severity is determined by likelihood and impact and can be classified into five categories accordingly: Critical, High, Medium, Low, Informational shown in table 1.1.

Likelihood \ IMPACT	Informational	Low	Medium	High
High	INFO	MEDIUM	HIGH	CRITICAL
Medium	INFO	LOW	MEDIUM	HIGH
Low	INFO	LOW	LOW	MEDIUM

Table 1.1 Overall Risk Severity

To evaluate the risk, we will be going through a list of items, and each would be labelled with a severity category. The audit was performed with a systematic approach guided by a comprehensive assessment list carefully designed to identify known and impactful security issues. If our tool or analysis does not identify any issue, the contract can be considered safe regarding the assessed item. For any discovered issue, we might further deploy contracts on our private test environment and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.2.

- **Basic Coding Bugs**: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- **Code and business security testing**: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- **Additional Recommendations**: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Category	Assessment Item
Basic Coding Assessment	• Apply Verification Control • Authorization Access Control • Forged Transfer Vulnerability • Forged Transfer Notification • Numeric Overflow • Transaction Rollback Attack • Transaction Block Stuffing Attack • Soft Fail Attack • Hard Fail Attack • Abnormal Memo • Abnormal Resource Consumption • Secure Random Number

Advanced Source Code Scrutiny	• Asset Security • Cryptography Security • Business Logic Review • Source Code Functional Verification • Account Authorization Control • Sensitive Information Disclosure • Circuit Breaker • Blacklist Control • System API Call Analysis • Contract Deployment Consistency Check • Abnormal Resource Consumption
Additional Recommendations	• Semantic Consistency Checks • Following Other Best Practices

Table 1.2: The Full List of Assessment Items

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [14], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development.

2. FINDINGS OVERVIEW

2.1 Project Info And Contract Address

Project Name	Audit Time	Language
TNTX solana	18/09/2025 - 14/10/2025	Rust

Repository
https://github.com/Antoninw3/smartcontracts-tntx

Commit Hash
5f56253c93d5926c62d5eae4eb985d8c50910878

2.2 Summary

Severity	Found
CRITICAL	0
HIGH	6
MEDIUM	6
LOW	5
INFO	6

2.3 Key Findings

Severity	Findings Title	Status
HIGH	Calls are not subject to permission control	Acknowledge
HIGH	Stake reward double counting	Fixed
HIGH	Re-staking may result in the principal being covered	Fixed
HIGH	Double counting of expired rewards	Fixed
HIGH	Illegal unstake results in loss of rewards	Fixed
HIGH	Illegal time operation	Fixed
MEDIUM	Inconsistent lengths cause panic	Fixed
MEDIUM	Fee calculation bypass risk	Acknowledge
MEDIUM	Mistakenly closing a legitimate game	Acknowledge
MEDIUM	The game status is not updated in time	Fixed
MEDIUM	create_airdrop lacks permission control	Acknowledge
MEDIUM	Days type conversion causes calculation problems	Fixed
LOW	max_players has no minimum bound	Fixed
LOW	There is no limit on the start time of the Airdrop	Fixed
LOW	create_game's timing logic flaw	Fixed
LOW	Incorrect cancellation logic	Fixed
LOW	Lack of logical judgment of time	Fixed
INFO	Useless accounts cause waste of resources	Fixed

Severity	Findings Title	Status
INFO	Misspelling of words	Acknowledge
INFO	Non-standard PDA and ATA constraints	Fixed
INFO	ATA account constraints are not standardized	Fixed
INFO	Do not verify vault balance before transfer	Fixed
INFO	Wrong amount leads to waste of resources	Fixed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Calls are not subject to permission control

SEVERITY: HIGH **STATUS:** Acknowledge

PATH:

`solana/staking/lib.rs`

DESCRIPTION:

When notify_reward distributes the reward pool, it will check the pool. If the pool has not been initialized or no one has staked it, it will burn the tokens in temp_reward_vault, or transfer temp_reward_vault to reward_vault_official and increase the reward distribution.

The problem is that this function with sensitive operations has no permission control and lacks signer verification.

IMPACT:

By mistake, anyone can be allowed to call it. Malicious attacks may cause tokens to be burned and the reward pool to be manipulated.

RECOMMENDATIONS:

Add signer verification to ensure only authorized accounts can call this sensitive function:

```
 #[derive(Accounts)]
 pub struct NotifyReward<'info> {
-    #[account(seeds=[b"config"], bump)]
-    pub config: Account<'info, Config>,
+    #[account(
+        seeds=[b"config"],
+        bump,
+        constraint = super_authority.key() == config.super_authority
+    )]
+    pub config: Account<'info, Config>,
+    #[account(mut)]
```

```
+     pub super_authority: Signer<'info>,
      ...
  }
```

3.2 Stake reward double counting

SEVERITY: HIGH **STATUS:** Fixed

PATH:

`solana/staking/lib.rs`

DESCRIPTION:

In expire_stake, the rewards that can be claimed in the current staking_status are calculated and the status.claimable status is updated. However, the status.reward_per_share_paid status is missing, which will lead to repeated calculations in the subsequent claim_rewards, resulting in double rewards.

```
pub fn expire_stake(
    ctx: Context<ExpireStake>
) -> Result<()> {
    let status = &mut ctx.accounts.staking_status;
    let now = Clock::get()?.unix_timestamp;
    require!(now >= status.end_timestamp, StakingError::StakeStillActive);
    require!(status.status == Status::Current,
   StakingError::InvalidStatus);

    let pool = &mut ctx.accounts.pool;
    let delta = pool.reward_per_share
        .checked_sub(status.reward_per_share_paid)
        .unwrap_or(0);
    let earned = (status.amount as u128)
        .checked_mul(delta).unwrap()
        .checked_div(Q64).unwrap() as u64;
    status.claimable = status.claimable.checked_add(earned).unwrap();

    pool.total_staked =
   pool.total_staked.checked_sub(status.amount).unwrap();
    status.status = Status::Finished;
    Ok(())
}
```

IMPACT:

If status.reward_per_share_paid is not updated in time, users may receive double rewards.

RECOMMENDATIONS:

Update status.reward_per_share_paid in time to prevent double reward claims:

```
pub fn expire_stake(
    ctx: Context<ExpireStake>
) -> Result<()> {
    let status = &mut ctx.accounts.staking_status;
    let now = Clock::get()?.unix_timestamp;
    require!(now >= status.end_timestamp, StakingError::StakeStillActive);
    require!(status.status == Status::Current,
   StakingError::InvalidStatus);

    let pool = &mut ctx.accounts.pool;
    let delta = pool.reward_per_share
        .checked_sub(status.reward_per_share_paid)
        .unwrap_or(0);
    let earned = (status.amount as u128)
        .checked_mul(delta).unwrap()
        .checked_div(Q64).unwrap() as u64;
    status.claimable = status.claimable.checked_add(earned).unwrap();
+   status.reward_per_share_paid = pool.reward_per_share;

    pool.total_staked =
   pool.total_staked.checked_sub(status.amount).unwrap();
    status.status = Status::Finished;
    Ok(())
}
```

3.3 Re-staking may result in the principal being covered

SEVERITY: HIGH **STATUS:** Fixed

PATH:

solana/staking/lib.rs

DESCRIPTION:

The stake function allows re-staking with a staking_status of Finished, but the new amount will overwrite the old data. Currently, expire_stake and unstake allow the staking_status to change, but unstake closes the account after execution, while expire_stake does not. Therefore, if a user calls expire_stake and then re-stakes, the old principal may be overwritten.

```
pub fn stake(
    ctx: Context<Stake>,
    amount: u64,
    duration: i64,
) -> Result<()> {
    ...

    if !status.is_initialized {
        ...
    } else if status.status == Status::Current {
        ...
    } else {
        // Nouveau cycle apres Finished
        // claimable conserve tel quel, pas d'accrual pendant le gap
   (amount=0)
        status.amount          = amount;
        status.start_timestamp = now;
        status.end_timestamp   = new_end;
        status.status          = Status::Current;
        status.reward_per_share_paid = pool.reward_per_share;
    }

    pool.total_staked = pool.total_staked.checked_add(amount).unwrap();
    Ok(())
}
```

IMPACT:

If the user calls the expire_stake then stake sequence, the old principal will be lost.

RECOMMENDATIONS:

It is recommended to check whether the amount is 0 when re-staking:

```
    } else {
        // Nouveau cycle apres Finished
+       require!(status.amount == 0, StakingError::InvalidAmount);
        status.amount          = amount;
        status.start_timestamp = now;
        status.end_timestamp   = new_end;
        status.status          = Status::Current;
        status.reward_per_share_paid = pool.reward_per_share;
    }
```

3.4 Double counting of expired rewards

SEVERITY: HIGH **STATUS:** Fixed

PATH:

`solana/staking/lib.rs`

DESCRIPTION:

The claim_rewards function will recalculate the new rewards that can be claimed, but it does not verify whether the current stake has ended, and mistakenly allows the completed stake to claim the new rewards.

```
pub fn claim_rewards(ctx: Context<ClaimRewards>) -> Result<()> {
    let pool = &ctx.accounts.pool;
    let status = &mut ctx.accounts.staking_status;
    let delta = pool.reward_per_share
        .checked_sub(status.reward_per_share_paid)
        .unwrap_or(0);
    let earned = (status.amount as u128)
        .checked_mul(delta).unwrap()
        .checked_div(Q64).unwrap() as u64;
    let total = status.claimable.checked_add(earned).unwrap();

    require!(total > 0, StakingError::NoClaimable);

    status.claimable = 0;
    status.reward_per_share_paid = pool.reward_per_share;

    ...
    Ok(())
}
```

IMPACT:

You can still claim new rewards after expire_stake.

RECOMMENDATIONS:

Add verification of stake status to prevent claiming rewards for expired stakes:

```
pub fn claim_rewards(ctx: Context<ClaimRewards>) -> Result<()> {
    let pool = &ctx.accounts.pool;
    let status = &mut ctx.accounts.staking_status;
    let delta = pool.reward_per_share
        .checked_sub(status.reward_per_share_paid)
        .unwrap_or(0);
-   let earned = (status.amount as u128)
-       .checked_mul(delta).unwrap()
-       .checked_div(Q64).unwrap() as u64;
+   let mut earned: u64 = 0;
+   if(status.status == Status::Current) {
+       earned = (status.amount as u128)
+           .checked_mul(delta).unwrap()
+           .checked_div(Q64).unwrap() as u64;
+   }
    let total = status.claimable.checked_add(earned).unwrap();

    require!(total > 0, StakingError::NoClaimable);

    status.claimable = 0;
    status.reward_per_share_paid = pool.reward_per_share;

    ...
    Ok(())
}
```

3.5 Illegal unstake results in loss of rewards

SEVERITY: HIGH **STATUS:** Fixed

PATH:

solana/staking/lib.rs

DESCRIPTION:

If the stake has not executed expire_stake in the unstake, it will help calculate the rewards that should be received, but it will not be distributed to the user. Finally, the account will be closed directly, resulting in the loss of the user's rewards.

```
pub fn unstake(ctx: Context<Unstake>) -> Result<()> {
    let status = &mut ctx.accounts.staking_status;
    let pool   = &mut ctx.accounts.pool;
    let now    = Clock::get()?.unix_timestamp;

    // Si ca vient d'expirer mais pas encore ”expire_stake”
    if status.status == Status::Current && now >= status.end_timestamp {
        let delta = pool.reward_per_share
            .checked_sub(status.reward_per_share_paid)
            .ok_or(StakingError::MathError)?;
        let earned = (status.amount as u128)
            .checked_mul(delta).ok_or(StakingError::MathError)?
            .checked_div(Q64).ok_or(StakingError::MathError)? as u64;

        status.claimable = status.claimable.checked_add(earned)
            .ok_or(StakingError::MathError)?;
        status.reward_per_share_paid = pool.reward_per_share;

        pool.total_staked = pool.total_staked.checked_sub(status.amount)
            .ok_or(StakingError::MathError)?;
        status.status = Status::Finished;
    }

    // Ferme le compte de status
    status.close(ctx.accounts.authority.to_account_info())?;
    Ok(())
```

```
}
```

IMPACT:

If the user unstakes without calling claim_rewards, the reward will be lost.

RECOMMENDATIONS:

It is recommended to cancel the harvest calculation here, requiring users to call expire_stake and claim_rewards before unstake, ensuring that status.status is Finished and status.claimable is 0.

3.6 Illegal time operation

SEVERITY: HIGH **STATUS:** Fixed

PATH:

`solana/game_manager/lib.rs`

DESCRIPTION:

The lack of a check for the game start time in cancel_game will allow users to exit the game and refund the fee after the game starts. This may cause unexpected results in users who have already received the game rewards.

```
pub fn cancel_game(ctx: Context<CancelGame>) -> Result<()> {
    ...

    if game.players.len() >= game.max_players as usize {
        require!(
            !game.start_now && now >= game.start_date - 5 * 60,
            GameError::CancelNotAllowed
        );
    }

    ...
    game.players.retain(|x| x != &player_key);
    Ok(())
}
```

IMPACT:

Users can exit the game after starting the game and claiming the rewards, and the transaction fee can be refunded.

RECOMMENDATIONS:

Add a time limit to only allow exiting before starting:

```
pub fn cancel_game(ctx: Context<CancelGame>) -> Result<()> {
    ...

    if game.players.len() >= game.max_players as usize {
        require!(
            !game.start_now && now >= game.start_date - 5 * 60,
            GameError::CancelNotAllowed
        );
    }
+   require!(now < game.start_date, GameError::GameStarted);
    ...
    game.players.retain(|x| x != &player_key);
    Ok(())
}
```

3.7 Inconsistent lengths cause panic

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

solana/game_manager/lib.rs

DESCRIPTION:

In the submit_results function, rewards are assigned to remaining_accounts by looping through them. There may be a situation where the length of rewards is greater than the length of remaining_accounts, which will cause the program to panic.

```
pub fn submit_results<'a, 'b, 'c, 'info>(
    ctx: Context<'a, 'b, 'c, 'info, SubmitResults<'info>>,
    rewards: Vec<Reward>,
) -> Result<()> {
    let cfg = &ctx.accounts.config;
    ...

    for (i, reward) in rewards.iter().enumerate() {
        let recipient_acc = &ctx.remaining_accounts[i];
        transfer_checked(
            CpiContext::new_with_signer(
                ctx.accounts.token_program.to_account_info(),
                TransferChecked {
                    from:      ctx.accounts.token_vault.to_account_info(),
                    mint:      ctx.accounts.token_mint.to_account_info(),
                    to:        recipient_acc.to_account_info(),
                    authority:
   ctx.accounts.vault_authority.to_account_info(),
                },
                &[signer_seeds],
            ),
            reward.amount,
            decimals,
        )?;
    }
```

```
    game.processed_count = game.processed_count.checked_add(rewards.len()
    as u64)
        .ok_or(GameError::MathError)?;
    game.processed_rewards = game.processed_rewards.checked_add(batch_sum)
        .ok_or(GameError::MathError)?;

    ...
    Ok(())
}
```

IMPACT:

When the length of rewards is greater than the length of remaining_accounts, the loop will cause the program to panic.

RECOMMENDATIONS:

Perform length check before the loop to ensure rewards.len() == ctx.remaining_accounts.len():

```
pub fn submit_results<'a, 'b, 'c, 'info>(
    ctx: Context<'a, 'b, 'c, 'info, SubmitResults<'info>>,
    rewards: Vec<Reward>,
) -> Result<()> {
    let cfg = &ctx.accounts.config;
    ...
+   require!(
+       rewards.len() == ctx.remaining_accounts.len(),
+       GameError::NotEnoughAccountKeys
+   );

    for (i, reward) in rewards.iter().enumerate() {
        let recipient_acc = &ctx.remaining_accounts[i];
        transfer_checked(
            CpiContext::new_with_signer(
                ctx.accounts.token_program.to_account_info(),
                TransferChecked {
                    from:      ctx.accounts.token_vault.to_account_info(),
                    mint:      ctx.accounts.token_mint.to_account_info(),
                    to:        recipient_acc.to_account_info(),
                    authority:
    ctx.accounts.vault_authority.to_account_info(),
```

```
                },
                &[signer_seeds],
            ),
            reward.amount,
            decimals,
        )?;
    }

    game.processed_count = game.processed_count.checked_add(rewards.len()
    as u64)
        .ok_or(GameError::MathError)?;
    game.processed_rewards = game.processed_rewards.checked_add(batch_sum)
        .ok_or(GameError::MathError)?;

    ...
    Ok(())
}
```

3.8 Fee calculation bypass risk

SEVERITY: MEDIUM **STATUS:** Acknowledge

PATH:

`solana/game_manager/lib.rs`

DESCRIPTION:

The create_game function receives a fee_in_sol parameter as the fee for creating the game. However, this parameter has no restrictions and users can pass in illegal values, resulting in unexpected results that bypass the fee calculation.

IMPACT:

If the fee_in_sol passed by the user is 0, then the transaction can be conducted without paying the handling fee.

RECOMMENDATIONS:

Add a minimum value restriction to the fee_in_sol parameter of the create_game function.

3.9 Mistakenly closing a legitimate game

SEVERITY: MEDIUM **STATUS:** Acknowledge

PATH:

`solana/game_manager/lib.rs`

DESCRIPTION:

In cancel_game_server, authority_server is allowed to close the game, but there is no verification whether the game is over, which may affect the normal operation of the project.

```
pub fn cancel_game_server(ctx: Context<CancelGameServer>) -> Result<()> {
    let cfg = &ctx.accounts.config;
    require_keys_eq!(
        ctx.accounts.authority_server.key(),
        cfg.authority_server,
        GameError::Unauthorized
    );
    Ok(())
}
```

IMPACT:

If the game is in progress, authority_server can still be closed by cancel_game_server, which will affect the project operation.

RECOMMENDATIONS:

Before closing the game, check whether the finished state is completed. Add new error messages:

```
pub fn cancel_game_server(ctx: Context<CancelGameServer>) -> Result<()> {
    let cfg = &ctx.accounts.config;
    require_keys_eq!(
        ctx.accounts.authority_server.key(),
        cfg.authority_server,
        GameError::Unauthorized
```

```
    );

+   require!(ctx.accounts.game.finished, GameError::GameNotFinished);
    Ok(())
}
```

3.10 The game status is not updated in time

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`solana/game_manager/lib.rs`

DESCRIPTION:

The submit_results function distributes rewards for unfinished games, but does not update game.finished in time after execution, causing repeated calls and affecting other logical judgments.

IMPACT:

When the reward settlement is completed, failure to update the status in time will affect the judgment of other logic and affect the operation of the contract.

RECOMMENDATIONS:

Finally update the status in submit_results, game.finished = true;

3.11 create_airdrop lacks permission control

SEVERITY: MEDIUM **STATUS:** Acknowledge

PATH:

`solana/game_manager/lib.rs`

DESCRIPTION:

The create_airdrop function allows signers to use their own accounts to create short rewards. If ordinary users call this function to create an airdrop, they will not be able to withdraw it in subsequent operations due to permission issues.

IMPACT:

This will result in ordinary users' funds being frozen or unauthorized transfers by authorized accounts, affecting user experience and fund security.

RECOMMENDATIONS:

Check for permissions when calling, here it is set to the same permission as authority_server and submit_result_airdrop:

```
pub fn create_airdrop(
    ctx: Context<CreateAirdrop>,
    start_date: i64,
    max_players: u16,
    token_mint: Pubkey,
    total_airdrop: u64,
) -> Result<()> {
    let cfg = &ctx.accounts.config;
    let ad  = &mut ctx.accounts.airdrop;

+   require_keys_eq!(
+       ctx.accounts.authority.key(),
+       cfg.authority_server,
+       GameError::Unauthorized
+   );
```

```
    ...
}
```

3.12 Days type conversion causes calculation problems

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

solana/staking/lib.rs

DESCRIPTION:

The reduce_duration function allows users to pay a fee to reduce the duration of their stake. However, the externally passed “days” is of type u64, which is forcibly converted to i64 during execution. The maximum value of the u64 type is 2^64-1, and the maximum value of the i64 type is 2^63-1. If a maliciously larger number is passed, sec_reduce will return a negative number, leading to unpredictable results.

```
pub fn reduce_duration(
    ctx: Context<ReduceDuration>,
    days: u64,
) -> Result<()> {
    ...

    let sec_reduce = (days as i64) * 86400;
    require!(
        remaining.checked_sub(sec_reduce).unwrap() >= min_rem,
        StakingError::DurationTooShort
    );

    ...

    st.end_timestamp = st.end_timestamp.checked_sub(sec_reduce).unwrap();
    Ok(())
}
```

IMPACT:

A maliciously constructed days parameter may cause calculation problems and unexpected results.

RECOMMENDATIONS:

Set a range limit for the days passed in from outside and add a new Error:

```
pub fn reduce_duration(
    ctx: Context<ReduceDuration>,
    days: u64,
) -> Result<()> {
+   let max_days = (i64::MAX as u64).checked_div(86400).unwrap();
+   require!(days <= max_days, StakingError::DaysExceedMax);
    ...

    let sec_reduce = (days as i64) * 86400;
    require!(
        remaining.checked_sub(sec_reduce).unwrap() >= min_rem,
        StakingError::DurationTooShort
    );

    ...

    st.end_timestamp = st.end_timestamp.checked_sub(sec_reduce).unwrap();
    Ok(())
}
```

3.13 max_players has no minimum bound

SEVERITY: LOW **STATUS:** Fixed

PATH:

`solana/game_manager/lib.rs`

DESCRIPTION:

In both the create_airdrop and create_game functions, a max_players parameter needs to be passed in from the outside. There is no restriction here, so it can be 0.

IMPACT:

When the value of max_players passed in is 0, the project may not run normally.

RECOMMENDATIONS:

Add a limit greater than 0 to max_players.

3.14 There is no limit on the start time of the Airdrop

SEVERITY: LOW **STATUS:** Fixed

PATH:

`solana/game_manager/lib.rs`

DESCRIPTION:

In create_airdrop, a start_date parameter is allowed to be received as the start time of the airdrop, but this parameter does not check whether it is now or after. If the wrong time is passed, the airdrop will never start.

IMPACT:

Entering an incorrect time may cause the Airdrop to be locked and never start.

RECOMMENDATIONS:

Check the validity of start_date in advance and add error message:

```
pub fn create_airdrop(
    ctx: Context<CreateAirdrop>,
    start_date: i64,
    max_players: u16,
    token_mint: Pubkey,
    total_airdrop: u64,
) -> Result<()> {
    let cfg = &ctx.accounts.config;
    let ad  = &mut ctx.accounts.airdrop;

    require!(total_airdrop > 0, GameError::ExceedTotalAirdrop);
+   let current_timestamp = Clock::get()?.unix_timestamp;
+   require!(
+       start_date >= current_timestamp,
+       GameError::StartDateTooEarly
+   );
```

```
    ...
}
```

3.15 create_game's timing logic flaw

SEVERITY: LOW **STATUS:** Fixed

PATH:

`solana/game_manager/lib.rs`

DESCRIPTION:

create_game allows start_now and start_date to be passed in from outside to limit the start time, but there is an unexpected situation where start_now is false but start_date still starts.

IMPACT:

This will cause the unexpected result that start_now is false, but start_date has already started.

RECOMMENDATIONS:

Add proper time validation logic:

```
pub fn create_game(
    ctx: Context<CreateGame>,
    entry_fee: u64,
    start_now: bool,
    start_date: i64,
    max_players: u16,
    token_mint: Pubkey,
    fee_in_sol: u64,
) -> Result<()> {
    let game = &mut ctx.accounts.game;

    let need = GameState::space_for(max_players);

+   let current_timestamp = Clock::get()?.unix_timestamp;
+   if start_now {
+       require!(
+           start_date == current_timestamp,
+           GameError::StartDateMustBeNow
```

```
+        );
+    } else {
+        require!(
+            start_date > current_timestamp,
+            GameError::StartDateMustBeFuture
+        );
+    };

    ...
}
```

3.16 Incorrect cancellation logic

SEVERITY: LOW **STATUS:** Fixed

PATH:

`solana/game_manager/lib.rs`

DESCRIPTION:

In the cancel_game function, users are allowed to cancel their participation in the game if the number of participants is full but the game has not started and now greater than or equal to game.start_date minus 5 minutes. The incorrect logic here is that the game can be canceled within the first five minutes. The correct logic should be that the game cannot be canceled within the first five minutes.

```
pub fn cancel_game(ctx: Context<CancelGame>) -> Result<()> {
    ...

    require!(!game.finished, GameError::GameClosed);
    require!(game.players.contains(&player_key), GameError::NotInGame);

    if game.players.len() >= game.max_players as usize {
        require!(
            !game.start_now && now >= game.start_date - 5 * 60,
            GameError::CancelNotAllowed
        );
    }

    ...
}
```

IMPACT:

The incorrect logic allows cancellation within the 5-minute window before game start.

RECOMMENDATIONS:

Modify judgment logic to prevent cancellation within the 5-minute window:

```
pub fn cancel_game(ctx: Context<CancelGame>) -> Result<()> {
    ...

    require!(!game.finished, GameError::GameClosed);
    require!(game.players.contains(&player_key), GameError::NotInGame);

    if game.players.len() >= game.max_players as usize {
        require!(
-           !game.start_now && now >= game.start_date - 5 * 60,
+           !game.start_now && now <= game.start_date - 5 * 60,
            GameError::CancelNotAllowed
        );
    }

    ...
}
```

3.17 Lack of logical judgment of time

SEVERITY: LOW **STATUS:** Fixed

PATH:

`solana/game_manager/lib.rs`

DESCRIPTION:

When submit_results ends the game, it only checks whether it is finished and ignores the start time of the game.

```
pub fn submit_results<'a, 'b, 'c, 'info>(
    ctx: Context<'a, 'b, 'c, 'info, SubmitResults<'info>>,
    rewards: Vec<Reward>,
) -> Result<()> {
    ...

    require!(!rewards.is_empty(), GameError::NoRewards);
    require!(!game.finished, GameError::AlreadyFinished);

    ...
    Ok(())
}
```

IMPACT:

Incorrect operation will allow a game that has not yet started to distribute rewards and change its status.

RECOMMENDATIONS:

Add time verification to ensure the game has started:

```
pub fn submit_results<'a, 'b, 'c, 'info>(
    ctx: Context<'a, 'b, 'c, 'info, SubmitResults<'info>>,
    rewards: Vec<Reward>,
```

```
) -> Result<()> {
    ...
+   let now = Clock::get()?.unix_timestamp;
    require!(!rewards.is_empty(), GameError::NoRewards);
    require!(!game.finished, GameError::AlreadyFinished);
+   require!(now > game.start_date, GameError::GameNotStarted);
    ...
    Ok(())
}
```

3.18 Useless accounts cause waste of resources

SEVERITY: INFO **STATUS:** Fixed

PATH:

`solana/game_manager/lib.rs`

DESCRIPTION:

In the CreateGame struct, some accounts are not used in the handler, which will cost more useless fees.

IMPACT:

Unused accounts will cost more money, and deleting them can also improve the readability of the program.

RECOMMENDATIONS:

Delete some accounts that are not used in the handler.

3.19 Misspelling of words

SEVERITY: INFO **STATUS:** Acknowledge

PATH:

`solana/game_manager/lib.rs`

DESCRIPTION:

In game_manager, staking is misspelled as stacking.

IMPACT:

This spelling mistake may cause confusion for code reviewers and maintainers.

RECOMMENDATIONS:

Change stacking back to staking.

3.20 Non-standard PDA and ATA constraints

SEVERITY: INFO **STATUS:** Fixed

PATH:

`solana/game_manager/lib.rs`

DESCRIPTION:

In SubmitResults and SubmitAirdropResults, staking_temp_authority and reward_vault_authority will be used as the authority of the two ATA accounts temp_reward_vault and reward_vault respectively, but the PDA constraint here is not standardized.

IMPACT:

Non-standard PDA constraints may lead to security vulnerabilities or unexpected behavior.

RECOMMENDATIONS:

Modify in SubmitResults and SubmitAirdropResults to add proper PDA constraints:

```
#[derive(Accounts)]
pub struct SubmitResults<'info> {
    #[account(mut)]
    pub game: Account<'info, GameState>,
    ...

    #[account(
        init_if_needed,
        payer = authority,
        associated_token::mint = token_mint,
        associated_token::authority = staking_temp_authority,
        associated_token::token_program = token_program,
    )]
    pub temp_reward_vault: InterfaceAccount<'info, TokenAccount>,

-   #[account(mut)]
-   pub staking_temp_authority: AccountInfo<'info>,
```

```
+    #[account(
+        seeds = [b"staking_temp", game.token_mint.as_ref()],
+        bump
+    )]
+    pub staking_temp_authority: AccountInfo<'info>,

-    #[account(mut)]
-    pub reward_vault_authority: UncheckedAccount<'info>,
+    #[account(
+        seeds = [b"reward_vault", game.token_mint.as_ref()],
+        bump
+    )]
+    pub reward_vault_authority: AccountInfo<'info>,

    #[account(
        mut,
        associated_token::mint = token_mint,
        associated_token::authority = reward_vault_authority,
        associated_token::token_program = token_program,
    )]
    pub reward_vault: InterfaceAccount<'info, TokenAccount>,

    pub staking_program: Program<'info, SplStaking>,
}
```

3.21 ATA account constraints are not standardized

SEVERITY: INFO **STATUS:** Fixed

PATH:

solana/staking/lib.rs

DESCRIPTION:

In some functions such as stake, the ATA account verification is not perfect, lacking the verification of mint, authority and token_program, which is not standardized.

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

    #[account(owner = token_program.key())]
    pub mint: InterfaceAccount<'info, Mint>,

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

    #[account(mut)]
    pub user_ata: InterfaceAccount<'info, TokenAccount>,

    #[account(seeds = [b”vault”, mint.key().as_ref()], bump)]
    pub vault_authority: UncheckedAccount<'info>,

    #[account(
        init_if_needed,
        payer = authority,
        associated_token::mint = mint,
        associated_token::authority = vault_authority,
        associated_token::token_program = token_program
    )]
    pub vault_ata: InterfaceAccount<'info, TokenAccount>,

    ...
```

```
    pub token_program: Interface<'info, TokenInterface>,
    pub associated_token_program: Program<'info, AssociatedToken>,
    pub system_program: Program<'info, System>,
    pub rent: Sysvar<'info, Rent>,
}
```

IMPACT:

Missing constraints may allow incorrect token accounts to be used, leading to security issues.

RECOMMENDATIONS:

Improve and standardize ATA checksum. The same problem also occurs with NotifyReward, ClaimRewards, and Unstake:

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

    #[account(owner = token_program.key())]
    pub mint: InterfaceAccount<'info, Mint>,

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

-   #[account(mut)]
+   #[account(
+       mut,
+       associated_token::mint = mint,
+       associated_token::authority = authority,
+       associated_token::token_program = token_program
+   )]
    pub user_ata: InterfaceAccount<'info, TokenAccount>,

    ...

    pub token_program: Interface<'info, TokenInterface>,
    pub associated_token_program: Program<'info, AssociatedToken>,
    pub system_program: Program<'info, System>,
    pub rent: Sysvar<'info, Rent>,
}
```

3.22 Do not verify vault balance before transfer

SEVERITY: INFO **STATUS:** Fixed

PATH:

solana/staking/lib.rs

DESCRIPTION:

When executing claim_rewards, the rewards that the user can claim are calculated and transferred from reward_vault to the user. There is no verification of whether there are enough rewards in the vault, which may cause the transaction to fail.

```
pub fn claim_rewards(ctx: Context<ClaimRewards>) -> Result<()> {
    ...
    let total = status.claimable.checked_add(earned).unwrap();
    require!(total > 0, StakingError::NoClaimable);

    ...

    transfer_checked(
        CpiContext::new_with_signer(
            ctx.accounts.token_program.to_account_info(),
            TransferChecked {
                from:      ctx.accounts.reward_vault.to_account_info(),
                mint:      ctx.accounts.mint.to_account_info(),
                to:        ctx.accounts.user_ata.to_account_info(),
                authority:
   ctx.accounts.reward_vault_authority.to_account_info(),
            },
            &[&seeds[..]],
        ),
        total,
        decimals,
    )?;
    Ok(())
}
```

IMPACT:

When the reward_vault balance is insufficient, the transaction may fail, affecting the user experience.

RECOMMENDATIONS:

Add a balance check before executing the transfer and return the error message. The same situation also exists in some other transfers:

```
pub fn claim_rewards(ctx: Context<ClaimRewards>) -> Result<()> {
    ...
    let total = status.claimable.checked_add(earned).unwrap();
    require!(total > 0, StakingError::NoClaimable);
+   let reward_vault_balance = ctx.accounts.reward_vault.amount;
+   require!(
+       reward_vault_balance >= total,
+       StakingError::InsufficientVaultBalance
+   );

    ...

    Ok(())
}
```

3.23 Wrong amount leads to waste of resources

SEVERITY: INFO **STATUS:** Fixed

PATH:

`solana/staking/lib.rs`

DESCRIPTION:

The amount of stake is allowed to be 0 when executing the stake function, but this is not an expected result and may lead to waste of resources.

IMPACT:

If the pledge amount is recorded as 0, it is meaningless and will lead to a waste of resources.

RECOMMENDATIONS:

Force check amount greater than 0 and add new error message:

```
pub fn stake(
    ctx: Context<Stake>,
    amount: u64,
    duration: i64,
) -> Result<()> {
+   require!(amount > 0, StakingError::AmountMustBePositive);
    ...
}
```

4. CONCLUSION

In this audit, we thoroughly analyzed **TNTX solana** smart contract implementation. The problems found are described and explained in detail in Section 3. The problems found in the audit have been communicated to the project leader. We therefore consider the audit result to be **PASSED**.

To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

5. APPENDIX

5.1 Basic Coding Assessment

5.1.1 Apply Verification Control

Description	The security of apply verification
Result	Not found
Severity	CRITICAL

5.1.2 Authorization Access Control

Description	Permission checks for external integral functions
Result	Not found
Severity	CRITICAL

5.1.3 Forged Transfer Vulnerability

Description	Assess whether there is a forged transfer notification vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.4 Transaction Rollback Attack

Description	Assess whether there is transaction rollback attack vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.5 Transaction Block Stuffing Attack

Description	Assess whether there is transaction blocking attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.6 Soft Fail Attack Assessment

Description	Assess whether there is soft fail attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.7 Hard Fail Attack Assessment

Description	Examine for hard fail attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.8 Abnormal Memo Assessment

Description	Assess whether there is abnormal memo vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.9 Abnormal Resource Consumption

Description	Examine whether abnormal resource consumption in contract processing
Result	Not found
Severity	CRITICAL

5.1.10 Random Number Security

Description	Examine whether the code uses insecure random number
Result	Not found
Severity	CRITICAL

5.2 Advanced Code Scrutiny

5.2.1 Cryptography Security

Description	Examine for weakness in cryptograph implementation
Result	Not found
Severity	HIGH

5.2.2 Account Permission Control

Description	Examine permission control issue in the contract
Result	Not found
Severity	MEDIUM

5.2.3 Malicious Code Behavior

Description	Examine whether sensitive behavior present in the code
Result	Not found
Severity	MEDIUM

5.2.4 Sensitive Information Disclosure

Description	Examine whether sensitive information disclosure issue present in the code
Result	Not found
Severity	MEDIUM

5.2.5 System API

Description	Examine whether system API application issue present in the code
Result	Not found
Severity	LOW

6. DISCLAIMER

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 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 without ExVul's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts ExVul 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. ExVul's position is that each company and individual are responsible for their own due diligence and continuous security. ExVul'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.

7. REFERENCES

[1] MITRE. CWE-191: Integer Underflow (Wrap or Wraparound). https://cwe.mitre.org/data/definitions/191.html.

[2] MITRE. CWE-197: Numeric Truncation Error. https://cwe.mitre.org/data/definitions/197.html.

[3] MITRE. CWE-400: Uncontrolled Resource Consumption. https://cwe.mitre.org/data/definitions/400.html.

[4] MITRE. CWE-440: Expected Behavior Violation. https://cwe.mitre.org/data/definitions/440.html.

[5] MITRE. CWE-684: Protection Mechanism Failure. https://cwe.mitre.org/data/definitions/693.html.

[6] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[7] MITRE. CWE CATEGORY: Behavioral Problems. https://cwe.mitre.org/data/definitions/438.html.

[8] MITRE. CWE CATEGORY: Numeric Errors. https://cwe.mitre.org/data/definitions/189.html.

[9] MITRE. CWE CATEGORY: Resource Management Errors. https://cwe.mitre.org/data/definitions/399.html.

[10] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

8. About Exvul Security

Premier Security for the Web3 Ecosystem

ExVul is a premier Web3 security firm committed to forging a secure and trustworthy decentralized ecosystem. Our elite team consists of security veterans from world-leading technology and blockchain security firms, including Huawei, YBB Captical, Qihoo 360, Amber, ByteDance, MoveBit, and PeckShield. Team member Nolan is ranked as a top-40 whitehat on Immunefi and is the platform's sole All-Star in the APAC region.

Our expertise covers the full spectrum of Web3 security. We conduct **meticulous smart contract audits**, having fortified thousands of projects on chains like Evm, Solana, Aptos, Sui etc. Our **Blockchain Protocol Audits** secure the core infrastructure of L1/L2 by uncovering deep-seated vulnerabilities. We also offer **comprehensive wallet audits** to protect user assets and provide **proactive web3 pentest**, enabling partners to neutralize threats before they strike.

Trusted by industry leaders, ExVul is the security partner for **OKX, Bitget, Cobo, Infini, Stacks, Aptos, Sui, CoreDAO, Sei** etc.

Contact

Website
www.exvul.com

Email
contact@exvul.com

Twitter
@EXVULSEC

Github
github.com/EXVUL-Sec