ExVul

SMART CONTRACT AUDIT REPORT

TNTX evm Smart Contract

OCTOBER 2025

Contents

1. EXECUTIVE SUMMARY

ExVul Web3 Security was engaged by **TNTX evm** 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 evm	18/09/2025 - 14/10/2025	Solidity

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

Commit Hash
5f56253c93d5926c62d5eae4eb985d8c50910878

2.2 Summary

Severity	Found
CRITICAL	0
HIGH	5
MEDIUM	4
LOW	9
INFO	1

2.3 Key Findings

Severity	Findings Title	Status
HIGH	Finished Stakes Continue to Accrue Rewards	Fixed
HIGH	Stake Extension Bypasses Lock-up	Fixed
HIGH	Restaking Finished Stake Locks Principal	Fixed
HIGH	Unverified Reward Amount Can Lead to Insolvency and DoS	Fixed
HIGH	Player Can Double-Profit by Claiming Rewards and Canceling Participation	Fixed
MEDIUM	Improper Game State Deletion Allows Game ID Reuse	Fixed
MEDIUM	Repeated Calls to playersReal Can Corrupt Airdrop State	Fixed
MEDIUM	Permanently Locked ETH from receive()	Fixed
LOW	Inefficient Struct Packing	Fixed
LOW	State-setting functions do not check for redundant updates	Fixed
LOW	Insufficient Parameter Validation in Airdrop Creation	Fixed
LOW	Insufficient Parameter Validation in Game Creation	Fixed
LOW	Inconsistent Batch Size Validation Between Game and Airdrop Results	Fixed
LOW	Unused authority Field in Structs Misrepresents Authorization Logic	Fixed
LOW	Results Can Be Submitted Before Game Start	Fixed
LOW	Staking with zero amount creates meaningless record	Fixed
LOW	Reward Burn Fails on Non-Standard Tokens	Fixed

Severity	Findings Title	Status
LOW	Inconsistent Pool Initialization in notifyReward	Fixed
INFO	Typo in variable and event names	Fixed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Finished Stakes Continue to Accrue Rewards

SEVERITY: HIGH **STATUS:** Fixed

PATH:

`contracts/staking_manager/ERC20Staking.sol`

DESCRIPTION:

When a stake expires, its status becomes Finished, but the staked amount is not zeroed out. The claimRewards function continues to calculate rewards for these finished stakes, even though they are no longer active.

```
// contracts/staking_manager/ERC20Staking.sol

function expireStakeFor(address user, address token) external
    nonReentrant {
     // ...
     p.totalStaked -= s.amount;
     s.status = Status.Finished;
     //...
}


function claimRewards(address token) external nonReentrant {
     // ...
     // 'earned' is calculated even for 'Finished' stakes.
     uint256 earned = (s.amount * delta) / Q64;
     // ...
}
```

IMPACT:

Users with expired stakes can claim rewards they did not earn. This drains the reward pool at the expense of active stakers and can lead to the contract running out of funds.

RECOMMENDATIONS:

In claimRewards, only calculate new rewards for stakes that are currently active (Status.Current).

```
function claimRewards(address token) external nonReentrant {
    StakingStatus storage s = statuses[msg.sender][token];
    require(s.isInitialized, "no status");
    Pool storage p = pools[token];
    uint256 delta = p.rewardPerShare - s.rewardPerSharePaid;
-   uint256 earned = (s.amount * delta) / Q64;
+   uint256 earned = 0;
+   if (s.status == Status.Current) {
+       earned = (s.amount * delta) / Q64;
+   }
    uint256 total = s.claimable + earned;
    require(total > 0, "no rewards");
    s.claimable = 0;
    s.rewardPerSharePaid = p.rewardPerShare;
    IERC20(token).safeTransfer(msg.sender, total);
    emit Claimed(msg.sender, token, total);
}
```

3.2 Stake Extension Bypasses Lock-up

SEVERITY: HIGH **STATUS:** Fixed

PATH:

`contracts/staking_manager/ERC20Staking.sol`

DESCRIPTION:

When extending an active stake, the stake function does not require the new lock-up period to be longer. An attacker can add a large amount to a nearly-expired stake without extending its duration, effectively bypassing the minimum lock-up requirement for the new capital.

```
// contracts/staking_manager/ERC20Staking.sol
// ...
} else if (s.status == Status.Current) {
    // ...
    s.amount += amount;

    // If `newEnd` is not greater than the original `s.endTimestamp`,
    // the timestamp is not updated.
    if (newEnd > s.endTimestamp) s.endTimestamp = newEnd;

    s.rewardPerSharePaid = p.rewardPerShare;
}
```

IMPACT:

This flaw enables attacks, where an attacker can stake a large amount for an extremely short period to capture rewards meant for long-term stakers. It breaks the core economic model of the protocol.

RECOMMENDATIONS:

When adding to a stake, require that the new endTimestamp must be later than the existing one.

```
// contracts/staking_manager/ERC20Staking.sol
// ...
```

```
} else if (s.status == Status.Current) {
    // ...
    uint256 delta = p.rewardPerShare - s.rewardPerSharePaid;
    uint256 earned = (s.amount * delta) / Q64;
    s.claimable += earned;
    s.amount += amount;
+   require(newEnd > s.endTimestamp, "duration must extend lock-up");
    s.endTimestamp = newEnd;
    s.rewardPerSharePaid = p.rewardPerShare;
} else {
// ...
```

3.3 Restaking Finished Stake Locks Principal

SEVERITY: HIGH **STATUS:** Fixed

PATH:

`contracts/staking_manager/ERC20Staking.sol`

DESCRIPTION:

When stake is called for a Status.Finished position, the code overwrites s.amount with the new deposit before the old principal is withdrawn, making the original funds unreachable.

```
// contracts/staking_manager/ERC20Staking.sol
function stake(address token, uint256 amount, int64 durationSec)
        external
        payable
        nonReentrant
    {


        if (!s.isInitialized) {
        // ...
        } else {
            // === Nouveau cycle quand Finished ===
            // ne touche pas a s.claimable (ca reste payable plus tard)
            s.amount = amount;
            s.startTimestamp = nowTs;
            s.endTimestamp = newEnd;
            s.status = Status.Current;
            s.rewardPerSharePaid = p.rewardPerShare;
            // s.claimable est conserve tel quel (pas d'earnings pendant
   le gap car amount=0)
        }

        p.totalStaked += amount;
        emit Staked(msg.sender, token, amount, s.endTimestamp);
    }
```

IMPACT:

Users who restake before withdrawing a finished position lose the previous principal; unstake only returns the new s.amount, while the older tokens stay locked in the contract.

RECOMMENDATIONS:

Require users to withdraw finished stakes (or preserve the previous principal) before allowing a new cycle so s.amount cannot be overwritten while non-zero.

```
-         } else {
+         } else {
+             require(s.amount == 0, "must unstake finished stake first");
              // === Nouveau cycle quand Finished ===
              s.amount = amount;
              s.startTimestamp = nowTs;
              s.endTimestamp = newEnd;
```

3.4 Unverified Reward Amount Can Lead to Insolvency and DoS

SEVERITY: HIGH **STATUS:** Fixed

PATH:

`contracts/staking_manager/ERC20Staking.sol`

DESCRIPTION:

The notifyReward function trusts the amount parameter provided by the notifier without verifying that the corresponding tokens have actually been received by the contract.

```
// contracts/staking_manager/ERC20Staking.sol
function notifyReward(address token, uint256 amount)external
    nonReentrant
    onlyOwnerOrNotifier
{
    Pool storage p = pools[token];// ...

    uint256 delta = (amount * Q64) / p.totalStaked;
    p.rewardPerShare += delta;
    // ...
    }
```

IMPACT:

If an incorrect amount is passed to notifyReward, the contract's reward accounting will be corrupted. This leads to a state of insolvency where liabilities exceed assets. Consequently, claimRewards calls will begin to fail due to insufficient balance, resulting in a Denial of Service (DoS) that locks all remaining rewards in the contract.

RECOMMENDATIONS:

Check the validity of the amount parameter before the operation by verifying the actual token balance received.

```
function notifyReward(address token, uint256 amount)external
```

```
    nonReentrant
    onlyOwnerOrNotifier
{
+   uint256 balanceBefore = IERC20(token).balanceOf(address(this));
+   IERC20(token).safeTransferFrom(msg.sender, address(this), amount);
+   uint256 balanceAfter = IERC20(token).balanceOf(address(this));
+   uint256 actualAmount = balanceAfter - balanceBefore;
+   require(actualAmount > 0, "No tokens received");

    Pool storage p = pools[token];
    // Use actualAmount instead of amount for calculations
-   uint256 delta = (amount * Q64) / p.totalStaked;
+   uint256 delta = (actualAmount * Q64) / p.totalStaked;
    p.rewardPerShare += delta;
    // ...
}
```

3.5 Player Can Double-Profit by Claiming Rewards and Canceling Participation

SEVERITY: HIGH **STATUS:** Fixed

PATH:

`contracts/game_manager/game_manager.sol`

DESCRIPTION:

A critical race condition allows players to claim rewards in early batches and then cancel their participation to withdraw their entry fee. This is possible because the cancellation logic does not properly check if the game has already started, only if it is full.

```
// contracts/game_manager/game_manager.sol
function cancelGame(string calldata gameId) external nonReentrant {
    // ...
    // The time check is inside an if-statement, allowing players in
   non-full
    // games to cancel even after the game has started.
    if (players[key].length >= g.maxPlayers) {
        require(!g.startNow && block.timestamp < g.startDate - 5 minutes,
                "Cancel not allowed");
    }
    // ...
}
```

IMPACT:

The attack occurs when submitResults is partially executed, paying the attacker, but the game is not yet marked as finished. The attacker can then call cancelGame to reclaim their entry fee, achieving a double profit.

RECOMMENDATIONS:

Prevent cancellation once result submission begins to protect game integrity.

```
// contracts/game_manager/game_manager.sol
```

```
function cancelGame(string calldata gameId) external nonReentrant {
    // [...]
    require(!g.finished, "Game closed");
    require(_isInGame(key, msg.sender), "Not in game");

   if (players[key].length >= g.maxPlayers) {
       require(!g.startNow && block.timestamp < g.startDate - 5 minutes,
               "Cancel not allowed");
   }
+   // Unconditionally prevent cancellation after the game's start time.
+   require(block.timestamp < g.startDate, "Game has started");

    IERC20(g.token).safeTransfer(msg.sender, g.entryFee);
    // [...]
}
```

3.6 Improper Game State Deletion Allows Game ID Reuse

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`contracts/game_manager/game_manager.sol`

DESCRIPTION:

Improper game state deletion logic error allows reuse of game identifiers which may compromise off-chain system integrity. cancelGameServer function uses delete games[key] to remove game record from storage, allowing the same gameId to be reused for creating new games.

```
// contracts/game_manager/game_manager.sol
function cancelGameServer(string calldata gameId, string calldata reason)
        external onlyAuthorityServer {
    // [...]
    delete players[key];
    delete games[key]; // Complete deletion allows ID reuse
}
```

IMPACT:

Can cause significant state confusion for off-chain services, frontends, or users who may rely on game IDs being unique throughout the system's lifetime.

RECOMMENDATIONS:

Mark games as cancelled instead of deleting them to prevent ID reuse.

```
+ enum GameState { Active, Finished, Cancelled }

struct Game {
    string  idText;
    uint256 entryFee;
    bool    startNow;
    uint256 startDate;
```

```
    uint8   maxPlayers;
    address token;
    address creator;
    address authority;
    bool    finished;
+   GameState state;
    // ... other fields
}

function cancelGameServer(string calldata gameId, string calldata reason)
        external onlyAuthorityServer {
    bytes32 key = keccak256(bytes(gameId));
    Game storage g = games[key];
    require(bytes(g.idText).length != 0, "game not found");
    require(!g.finished, "already finished");
    require(players[key].length == 0, "players already joined");
    delete players[key];
-   delete games[key];
+   g.state = GameState.Cancelled;
+   g.finished = true;
}
```

3.7 Repeated Calls to playersReal Can Corrupt Airdrop State

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`contracts/game_manager/game_manager.sol`

DESCRIPTION:

The playersReal function can be called multiple times, allowing actualPlayers to be set to a value less than the already processed recipient count.

```
// contracts/game_manager/game_manager.sol
function playersReal(string calldata airdropId, uint16 actualPlayers_)
    external
    onlyAuthorityServer
{
    //[...]
    ad.actualPlayers = actualPlayers_;
    emit AirdropPlayersRealSet(key, actualPlayers_);
}

function submitAirdropResults(
        string calldata airdropId,
        address[] calldata recipients,
        uint256[] calldata amounts
    ) external nonReentrant onlyAuthorityServer {
    //[...]
    require(ad.processedCount + recipients.length <= ad.actualPlayers,
            "too many in batch");
    //[...]
}
```

IMPACT:

This corrupts the airdrop's state by creating an invalid condition where processedCount > actualPlayers, making further reward distribution impossible and permanently locking all remaining funds.

RECOMMENDATIONS:

To allow for corrections while maintaining state integrity, ensure the new actualPlayers value cannot be set below the current processedCount.

```
        require(!ad.finished, "finished");
        require(actualPlayers_ > 0, "actual=0");
+       require(actualPlayers_ >= ad.processedCount,
+               "new value < processed count");
        require(uint32(actualPlayers_) <= uint32(ad.maxPlayers),
                "actual>max");

        ad.actualPlayers = actualPlayers_;
```

3.8 Permanently Locked ETH from receive()

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`contracts/staking_manager/ERC20Staking.sol`

DESCRIPTION:

The contract implements a receive() external payable {} function. This allows the contract to accept direct ETH transfers without invoking any other function.

```
// contracts/staking_manager/ERC20Staking.sol
receive() external payable {}
```

IMPACT:

Any user who mistakenly sends ETH directly to the contract address will have their funds permanently locked in the contract. There is no function available for non-owner users to recover this ETH.

RECOMMENDATIONS:

Add an owner-controlled function to withdraw any ETH that may have been accidentally sent to the contract. This provides a recovery mechanism for otherwise lost funds.

```
+   function rescueETH(address to, uint256 amount) external onlyOwner {
+       require(to != address(0), "to=0");
+       require(address(this).balance >= amount, "not enough ETH");
+       (bool ok, ) = payable(to).call{value: amount}("");
+       require(ok, "eth transfer failed");
+   }
+
    receive() external payable {}
}
```

3.9 Inefficient Struct Packing

SEVERITY: LOW **STATUS:** Fixed

PATH:

`contracts/game_manager/game_manager.sol`

DESCRIPTION:

The Solidity compiler can pack multiple, smaller-sized variables into a single 32-byte storage slot to save gas. This packing mechanism is defeated when smaller-sized variables are separated by larger ones. In the GameManager contract, the Game and Airdrop structs have their member variables declared in an order that prevents optimal packing, leading to wasted storage space.

IMPACT:

The sub-optimal struct layout causes every write operation to the Game or Airdrop structs to consume more gas than necessary. This increases the on-chain operational costs for both users and the protocol.

RECOMMENDATIONS:

Reorder the struct members by size (smallest to largest) to enable optimal storage packing and reduce gas costs.

```
struct Game {
-   string  idText;
-   uint256 entryFee;
-   bool    startNow;
-   uint256 startDate;
-   uint8   maxPlayers;
-   address token;
-   address creator;
-   address authority;
-   bool    finished;
-   uint256 creationFeePaidWei;
-   uint256 processedCount;
```

```
-   uint256 processedRewards;
-   uint256 totalStaked;
+   // Pack small types together
+   uint8   maxPlayers;      // 1 byte
+   bool    startNow;        // 1 byte
+   bool    finished;        // 1 byte
+   // 29 bytes left in this slot
+
+   // Addresses (20 bytes each)
+   address token;
+   address creator;
+   address authority;
+
+   // uint256 values (32 bytes each)
+   uint256 entryFee;
+   uint256 startDate;
+   uint256 creationFeePaidWei;
+   uint256 processedCount;
+   uint256 processedRewards;
+   uint256 totalStaked;
+
+   // Dynamic types last
+   string  idText;
}
```

3.10 State-setting functions do not check for redundant updates

SEVERITY: LOW **STATUS:** Fixed

PATH:

`contracts/game_manager/game_manager.sol`

DESCRIPTION:

Several state-setting functions (setAuthorityServer, setEthVault, setFeesPermille, setStaking, updateJoinFee) do not check if the new value is the same as the current value. This leads to unnecessary storage writes (SSTORE) and event emissions, wasting gas.

IMPACT:

Calling these functions with existing values leads to wasted gas, increasing the cost of admin operations and adding noise to the on-chain event history.

RECOMMENDATIONS:

Add a check in each state-setting function to ensure the new value is different from the current value before applying the change.

```
function setAuthorityServer(address newAuthority) external onlyOwner {
+    require(newAuthority != authorityServer, "Value unchanged");
     authorityServer = newAuthority;
     emit AuthorityServerUpdated(newAuthority);
}

function setEthVault(address newVault) external onlyOwner {
+    require(newVault != ethVault, "Value unchanged");
     ethVault = newVault;
     emit EthVaultUpdated(newVault);
}

function setFeesPermille(uint256 newFees) external onlyOwner {
+    require(newFees != feesPermille, "Value unchanged");
     feesPermille = newFees;
```

```
    emit FeesPermilleUpdated(newFees);
}
```

3.11 Insufficient Parameter Validation in Airdrop Creation

SEVERITY: LOW **STATUS:** Fixed

PATH:

`contracts/game_manager/game_manager.sol`

DESCRIPTION:

The createAirdrop function lacks validation for critical parameters allowing creation of invalid airdrops. Missing checks for airdropId and startDate parameters can lead to unusable airdrops with empty IDs or past start dates.

```
// contracts/game_manager/game_manager.sol
function createAirdrop(
    // [...]
) external payable nonReentrant {
    require(token != address(0), ”token=0”);
    // Missing validation for airdropId and startDate
    // [...]
}
```

IMPACT:

Allows creation of airdrops with empty IDs or past start dates causing poor user experience and potential system confusion for off-chain services.

RECOMMENDATIONS:

Add comprehensive parameter validation to prevent invalid airdrop creation.

```
function createAirdrop(
    string calldata airdropId,
    uint256 startDate,
    uint16  maxPlayers,
    address token,
    uint256 totalAirdrop
```

```
) external payable nonReentrant {
    require(token != address(0), "token=0");
+   require(bytes(airdropId).length > 0, "Airdrop ID cannot be empty");
+   require(startDate > block.timestamp, "Start date must be in future");
    require(maxPlayers > 0, "maxPlayers=0");
    require(totalAirdrop > 0, "totalAirdrop=0");
    bytes32 key = keccak256(bytes(airdropId));
    // ...
}
```

3.12 Insufficient Parameter Validation in Game Creation

SEVERITY: LOW **STATUS:** Fixed

PATH:

`contracts/game_manager/game_manager.sol`

DESCRIPTION:

The createGame function lacks validation for critical parameters allowing creation of invalid or illogical games. Missing checks for maxPlayers, startDate, and gameId parameters can lead to unusable games.

```
// contracts/game_manager/game_manager.sol
function createGame(
    // [...]
) external payable nonReentrant {
    require(token != address(0), "token=0");
    // Missing validations for maxPlayers, startDate, gameId
    // [...]
}
```

IMPACT:

Allows creation of games with zero players, past start dates, or empty game IDs causing poor user experience and potential system confusion.

RECOMMENDATIONS:

Add comprehensive parameter validation to prevent invalid game creation.

```
function createGame(
    string calldata gameId,
    uint256 entryFee,
    bool startNow,
    uint256 startDate,
    uint8 maxPlayers,
```

```
    address token,
    uint256 feeInETH
) external payable nonReentrant {
    require(token != address(0), "token=0");
+   require(bytes(gameId).length > 0, "Game ID cannot be empty");
+   require(maxPlayers > 0, "Must have at least 1 player");
+   if (startNow) {
+       require(startDate == block.timestamp,
+               "Start date must be current timestamp if startNow is
   true");
+   } else {
+       require(startDate > block.timestamp,
+               "Start date must be in the future");
+   }
    bytes32 key = keccak256(bytes(gameId));
    require(bytes(games[key].idText).length == 0, "game exists");
    // ... rest of function
}
```

3.13 Inconsistent Batch Size Validation Between Game and Airdrop Results

SEVERITY: LOW **STATUS:** Fixed

PATH:

`contracts/game_manager/game_manager.sol`

DESCRIPTION:

The submitResults function lacks explicit recipient count validation that exists in submitAirdropResults. While the final require(g.processedRewards == expectedRewards) check provides indirect protection, explicit validation would improve code consistency and clarity.

```
// contracts/game_manager/game_manager.sol
function submitAirdropResults(...) external onlyAuthorityServer {
    // ...
    require(ad.processedCount + recipients.length <= ad.actualPlayers,
            "too many in batch");
    // ...
}

function submitResults(...) external onlyAuthorityServer {
    // ...
    uint256 totalPlayers = players[key].length;
    g.processedCount += recipients.length;
    // ...
}
```

IMPACT:

Code inconsistency reduces maintainability.

RECOMMENDATIONS:

Add consistent batch size validation to maintain code consistency and provide defensive protection.

```
function submitResults(
```

```
    string calldata gameId,
    address[] calldata recipients,
    uint256[] calldata amounts
) external nonReentrant onlyAuthorityServer {
    // ...
    uint256 totalPlayers = players[key].length;
+   require(g.processedCount + recipients.length <= totalPlayers,
+           "Too many recipients in batch");
    IERC20 token = IERC20(g.token);
    // ...
}
```

3.14 Unused authority Field in Structs Misrepresents Authorization Logic

SEVERITY: LOW **STATUS:** Fixed

PATH:

`contracts/game_manager/game_manager.sol`

DESCRIPTION:

The Game and Airdrop structs contain a misleading authority field that is never used for on-chain authorization; all access control is handled globally.

```
struct Game {
    string  idText;
    uint256 entryFee;
    bool    startNow;
    uint256 startDate;
    uint8   maxPlayers;
    address token;
    address creator;
    address authority;  // Never used in actual access control
    bool    finished;
    // ...
}
```

IMPACT:

This misrepresents the contract's true centralized authorization model, potentially causing future developers to make incorrect security assumptions.

RECOMMENDATIONS:

Remove the unused authority field from the Game and Airdrop structs for clarity.

```
struct Game {
    string  idText;
    uint256 entryFee;
```

```
    bool    startNow;
    uint256 startDate;
    uint8   maxPlayers;
    address token;
    address creator;
-   address authority;
    bool    finished;
    // ... other fields
}

struct Airdrop {
    string  idText;
    uint256 startDate;
    uint16  maxPlayers;
    uint16  actualPlayers;
    address token;
    address creator;
-   address authority;
    uint256 totalAirdrop;
    bool    finished;
    // ... other fields
}
```

3.15 Results Can Be Submitted Before Game Start

SEVERITY: LOW **STATUS:** Fixed

PATH:

`contracts/game_manager/game_manager.sol`

DESCRIPTION:

The submitResults function lacks a check to ensure it is only called after the game's startDate, allowing the authorityServer to end a game prematurely.

```
// contracts/game_manager/game_manager.sol
function submitResults(
    // [...]
) external nonReentrant onlyAuthorityServer {
    // [...]
    require(!g.finished, "Already finished");
    // Missing check for g.startDate
    // [...]
}
```

IMPACT:

A malicious or faulty authorityServer can terminate a game before its scheduled start, violating player trust and preventing others from joining.

RECOMMENDATIONS:

Add a time check to ensure results are submitted only after the game has started.

```
// contracts/game_manager/game_manager.sol
function submitResults(
    // [...]
) external nonReentrant onlyAuthorityServer {
    // [...]
    require(!g.finished, "Already finished");
```

```
+   require(block.timestamp >= g.startDate, "Game has not started yet");
    require(recipients.length == amounts.length, "Mismatched inputs");
    // [...]
}
```

3.16 Staking with zero amount creates meaningless record

SEVERITY: LOW **STATUS:** Fixed

PATH:

`contracts/staking_manager/ERC20Staking.sol`

DESCRIPTION:

The stake function does not validate that the amount parameter is greater than zero. This allows a user to call the function with amount = 0, creating a staking record without any principal being staked.

```
function stake(address token, uint256 amount, int64 durationSec)
    external
    payable
    nonReentrant
{
    require(durationSec >= 24 * 3600, "duration < 24h");
    // No check for amount > 0

    // ...
    IERC20(token).safeTransferFrom(msg.sender, address(this), amount);
    // amount can be 0
    // ...
}
```

IMPACT:

This allows for the creation of "zombie" staking records with a zero principal amount, which adds to on-chain state bloat. It creates a confusing scenario where a stake can exist without any value, potentially leading to wasted gas for the user.

RECOMMENDATIONS:

Add a requirement to ensure that the staking amount is non-zero.

```
// contracts/staking_manager/ERC20Staking.sol
```

```
function stake(address token, uint256 amount, int64 durationSec)
    external
    payable
    nonReentrant
{
+   require(amount > 0, "amount is zero");
    require(durationSec >= 24 * 3600, "duration < 24h");
    if (feeStakeWei > 0) {
    // ...
}
```

3.17 Reward Burn Fails on Non-Standard Tokens

SEVERITY: LOW **STATUS:** Fixed

PATH:

`contracts/staking_manager/ERC20Staking.sol`

DESCRIPTION:

When totalStaked == 0, rewards are burned with a raw transfer. Tokens that deduct fees or omit return values can make this call revert or leave dust behind.

```
function notifyReward(address token, uint256 amount)
        external
        nonReentrant
        onlyOwnerOrNotifier
    {
       // ...
        if (p.totalStaked > 0) {
            // Distribution proportionnelle
            uint256 delta = (amount * Q64) / p.totalStaked;
            p.rewardPerShare += delta;
        } else {
            // Bruler la reward si aucun stake
            IERC20(token).transfer(
                address(0x000000000000000000000000000000000000dEaD),
                amount);
        }
       // ...
    }
```

IMPACT:

Burning path fails or misbehaves for fee-on-transfer or non-standard tokens, breaking reward distribution.

RECOMMENDATIONS:

Replace the call with safeTransfer so burn works across token variants.

```
            // Bruler la reward si aucun stake
-           IERC20(token).transfer(
-               address(0x000000000000000000000000000000000000dEaD),
-               amount);
+           IERC20(token).safeTransfer(
+               address(0x000000000000000000000000000000000000dEaD),
+               amount);

        emit NotifiedReward(token, amount, p.rewardPerShare);
```

3.18 Inconsistent Pool Initialization in notifyReward

SEVERITY: LOW **STATUS:** Fixed

PATH:

`contracts/staking_manager/ERC20Staking.sol`

DESCRIPTION:

When notifyReward creates a new pool, it sets isInitialized to true but fails to populate the p.token field, leaving it as the zero address.

```
// contracts/staking_manager/ERC20Staking.sol
if (!p.isInitialized) {
        p.isInitialized = true;
        p.totalStaked = 0;
        p.rewardPerShare = 0;
        }
```

IMPACT:

It creates a latent bug and an inconsistent state.

RECOMMENDATIONS:

Set the p.token field during pool initialization.

```
// contracts/staking_manager/ERC20Staking.sol
if (!p.isInitialized) {
        p.isInitialized = true;
+       p.token = token;
        p.totalStaked = 0;
        p.rewardPerShare = 0;
        }
```

3.19 Typo in variable and event names

SEVERITY: INFO **STATUS:** Fixed

PATH:

`contracts/game_manager/game_manager.sol`

DESCRIPTION:

The word “staking” is consistently misspelled as “stacking” throughout the contract in relation to airdrop staking fees. This affects a struct field, an event definition, and logic within a function.

```
struct Airdrop {
    string  idText;
    uint256 startDate;
    uint16  maxPlayers;
    uint16  actualPlayers;
    address token;
    address creator;
    address authority;
    uint256 totalAirdrop;
    bool    finished;
-   bool    stackingFeePaid;  // Typo: should be ”stakingFeePaid”
    uint256 processedCount;
    uint256 processedRewards;
}
```

IMPACT:

While this typo does not cause a functional bug, it reduces code quality and clarity. It can lead to confusion for future developers or auditors.

RECOMMENDATIONS:

Correct the spelling from “stacking” to “staking” in all relevant places.

```
struct Airdrop {
```

```
    // ... other fields
-   bool    stackingFeePaid;
+   bool    stakingFeePaid;
    // ... other fields
}

// Update all references throughout the contract
-   if (!ad.stackingFeePaid) {
+   if (!ad.stakingFeePaid) {
        // ...
-       ad.stackingFeePaid = true;
+       ad.stakingFeePaid = true;
    }
```

4. CONCLUSION

In this audit, we thoroughly analyzed **TNTX evm** 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