SHERLOCK

Security Review For Football.Fun

Collaborative Audit Prepared For: **Football.Fun**
Lead Security Expert(s): **dobrevaleri**
ParthMandale
Date Audited: **January 7 - January 8, 2026**

Introduction

Sport.fun is a fantasy sports game that uses smart contracts for a user driven marketplace, built on top of uniswap logic. This security review has been carried out to verify the integrity of an extension to the on-chain protocol to introduce a secure system that allows users to claim on chain rewards.

Scope

Repository: footballdotfun/sdf-contracts

Audited Commit: bd0188d139f7863bc6c2725ed5b2481ecff9faac

Final Commit: bd0188d139f7863bc6c2725ed5b2481ecff9faac

Files:

- src/sdf/contracts/interfaces/IRewardsManager.sol
- src/sdf/contracts/RewardsManager/RewardsManager.sol

Final Commit Hash

bd0188d139f7863bc6c2725ed5b2481ecff9faac

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
0	0	5

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue L-1: Reward Limits can not be Disabled in emergencies. [ACKNOWLEDGED]

Source:
https://github.com/sherlock-audit/2026-01-football-fun-update-jan-7th/issues/3

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

`rewardLimit` == 0 is treated as “no limit”, which can be counter intuitive if the governor expects zero to disable rewards in future emergencies.

Vulnerability Detail

`_processReward` only enforces the limit when `rewardLimit > 0`, so a limit of 0 permits any amount. This behavior is not inherently unsafe but can be confusing for operators and lead to misconfiguration in case the reward limits are genuenly needed to be disabled in the future for any emergency reasons.

Impact

Governor won't be able to ever disable rewards limit by setting it to 0 limit, if needed in any future emergencies

Code Snippet

```
function _processReward(uint256 _rewardId, RewardType _rewardType, address
↪  _walletAddress, uint256 _amount, uint256 _expirationTimestamp, bytes calldata
↪  _signature) internal {
// ...existing code... //
        uint256 rewardLimit = rewardLimits[_rewardType];
        if (rewardLimit > 0 && _amount > rewardLimit) revert
        ↪  RewardAmountExceedsLimit();
// ...existing code... //
```

Tool Used

Manual Review

Recommendation

Use `type(uint256).max` to represent “no limit” and reserve 0 to mean “no rewards allowed.” Update the limit check accordingly and document the semantics.

Code Changes in function `_processReward`:

```
- if (rewardLimit > 0 && _amount > rewardLimit) revert RewardAmountExceedsLimit();
+ if (_amount > rewardLimit) revert RewardAmountExceedsLimit();
```

Issue L-2: Centralization Risk: Governor Can Sweep Reward Tokens Without Claim Checks [ACKNOWL-EDGED]

Source: https://github.com/sherlock-audit/2026-01-football-fun-update-jan-7th/issues/4

This issue has been acknowledged by the team but won't be fixed at this time.

Vulnerability Detail

Function `withdrawERC20` allows `GOVERNOR_ROLE` to transfer any `ERC20` held by the contract (including FUN/GOLD) without reward limit checks or signature/expiration verification. This is typical admin sweep functionality, but it represents strong centralization: as the governor can drain the rewards pool at any time, blocking future claims.

Whether this is acceptable depends on the protocol's trust model and stated design in `RewardsManager.sol`

Recommendation

If a treasury sweep is required, document it explicitly and consider adding a timelock or multi sig control to reduce centralization risk.

Issue L-3: Inefficient signer verification loop can be optimized with early return [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-01-football-fun-update-jan-7th/issues/5

This issue has been acknowledged by the team but won't be fixed at this time.

Vulnerability Details

The `_verifySignature()` function in the RewardsManager contract uses an inefficient pattern for validating transaction signers. Currently, the function declares a `validSigner` boolean variable, iterates through all signers to check if the recovered signer matches, sets the flag to `true` when found, and then performs an additional check after the loop to revert if the signer is invalid.

```
## RewardsManager.sol

function _verifySignature(bytes32 _structHash, bytes calldata _signature) internal
↪  view {
    bytes32 digest = keccak256(
        abi.encodePacked("\x19\x01", _domainSeparatorV4(), _structHash)
    );
    address recoveredSigner = ECDSA.recover(digest, _signature);
    bool validSigner = false;
    uint256 signersLength = txSigners.length;
    for (uint256 i = 0; i < signersLength;) {
        if (recoveredSigner == txSigners[i]) {
            validSigner = true;
            break;
        }
        unchecked { i++; }
    }

    if (!validSigner) revert InvalidSignature();
}
```

This approach introduces unnecessary gas costs and code complexity. The contract already demonstrates a more efficient pattern in the `removeTxSigner()` function, which returns directly from the loop when the condition is met. By adopting this pattern, the `validSigner` variable becomes redundant, and the validation check is no longer needed.

Recommendation

Refactor the loop to return immediately when a matching signer is found, eliminating the need for the `validSigner` boolean variable and the subsequent conditional check, but

leaving the revert at the end of the function. This mirrors the implementation pattern already used in `removeTxSigner()`.

Issue L-4: Inefficient signer removal when target is last element [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-01-football-fun-update-jan-7th/issues/6

This issue has been acknowledged by the team but won't be fixed at this time.

Vulnerability Details

The `RewardsManager::removeTxSigner()` function contains an inefficiency when removing the last signer from the `txSigners` array. The current implementation always iterates through the entire array to find the signer to remove, even when that signer is the last element.

When the signer to be removed is the last element in the array, the function:

1. Loops through all signers until reaching the last one
2. Performs a self-assignment (`txSigners[i] = txSigners[signersLength - 1]`)
3. Pops the last element

This results in unnecessary gas consumption from the full loop iteration and redundant storage write operation when the signer is already in the last position.

Recommendation

Check if the target signer is the last element before entering the loop. If it is, skip the swap operation and directly pop the element, avoiding both the loop and the self-assignment.

Issue L-5: Redundant reward type validation in `_transferReward()` [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-01-football-fun-update-jan-7th/issues/7

This issue has been acknowledged by the team but won't be fixed at this time.

Vulnerability Details

The `RewardsManager::_transferReward()` function contains a redundant check that validates whether `_rewardType` is either `FUN_TOKEN` or `GOLD_TOKEN` before executing the token transfer logic:

```
function _transferReward(
    RewardType _rewardType,
    address _rewardAddress,
    address _recipient,
    uint256 _amount
) internal {
    if (_rewardType == RewardType.FUN_TOKEN || _rewardType ==
    ↪  RewardType.GOLD_TOKEN) {
        IERC20 token = IERC20(_rewardAddress);
        if (token.balanceOf(address(this)) < _amount) revert InsufficientBalance();
        token.safeTransfer(_recipient, _amount);
    }
}
```

This validation is unnecessary because `_transferReward()` is exclusively called from `RewardsManager::_processReward()`, which already validates that `_rewardType` is not `UNASSIGNED`:

```
function _processReward(
    uint256 _rewardId,
    RewardType _rewardType,
    address _walletAddress,
    uint256 _amount,
    uint256 _expirationTimestamp,
    bytes calldata _signature
) internal {
    if (_rewardType == RewardType.UNASSIGNED) revert InvalidRewardType();
    // ... additional validation logic
    _transferReward(_rewardType, rewardAddress, _walletAddress, _amount);
}
```

The check in `_transferReward()` adds no security value since the caller already ensures valid reward types. This results in wasted gas on every reward distribution.

Recommendation

Remove the redundant `if` statement from `_transferReward()` and execute the token transfer logic unconditionally, since the function is internal and its caller already validates the reward type.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.