SHERLOCK

Security Review For Football.Fun

Collaborative Audit Prepared For: **Football.Fun**
Lead Security Expert(s): **0x3b**
AlexMurphy
samuraii77
Date Audited: **August 26 - September 2, 2025**
Final Commit: **095a0b7**

Introduction

Football.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 the game marketplace and other smart contract based mechanics.

Scope

Repository: footballdotfun/fdf-contracts

Audited Commit: 72b507735ce24b7300b358e97d0995e325b91d79

Final Commit: 095a0b753fa4d74931fb9041975b66d7bdec1a80

Files:

- src/contracts/DevelopmentPlayers.sol
- src/contracts/FakeUSDC.sol
- src/contracts/Fun.sol
- src/contracts/PackSaleReveal.sol
- src/contracts/PackSale.sol
- src/contracts/PlayerContracts.sol
- src/contracts/PlayerPack.sol
- src/contracts/Player.sol
- src/exchange/FDFFactory.sol
- src/exchange/FDFPair.sol
- src/exchange/FeeManager.sol
- src/interfaces/IDevelopmentPlayers.sol
- src/interfaces/IERC2981.sol
- src/interfaces/IFDFFactory.sol
- src/interfaces/IFDFPair.sol
- src/interfaces/IFeeManager.sol
- src/interfaces/IFun.sol
- src/interfaces/IOwnable.sol
- src/interfaces/IPackSaleReveal.sol
- src/interfaces/IPackSale.sol
- src/interfaces/IPlayerContracts.sol
- src/interfaces/IPlayerPack.sol

- src/interfaces/IPlayer.sol
- src/libraries/PricingLibrary.sol

Final Commit Hash

095a0b753fa4d74931fb9041975b66d7bdec1a80

Findings

Each issue has an assigned severity:

- 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.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- 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	3

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue L-1: Approval Requirement Exposes Users to Unnecesary Risk if CONTRACT_RENEWAL_ROLE Is Compromised

Source: https://github.com/sherlock-audit/2025-08-football-fun-revision-audit-aug-26th/issues/25

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

Summary

The `renewContract` function from the `PlayerContracts` contract requires users to grant approval for the token transfer, which may result in users providing unlimited approval to the contract. If the account with the `CONTRACT_RENEWAL_ROLE` is compromised, all users who have granted approval are at risk.

Vulnerability Detail

The function in question depends on users granting approval to the contract for token transfers. This design may lead users to grant infinite approval, allowing the contract to transfer tokens on their behalf without further consent. If the CONTRACT_RENEWAL_ROLE account is compromised, an attacker could exploit the existing approvals to transfer tokens from all users who have previously granted approval.

Impact

A compromised CONTRACT_RENEWAL_ROLE account could result in unauthorized transfers of tokens from all users who have granted approval to the contract, potentially leading to significant loss of user funds.

Code Snippet

https://github.com/sherlock-audit/2025-08-football-fun-revision-audit-aug-26th/blob/main/fdf-contracts/src/contracts/PlayerContracts.sol#L66

Tool Used

Manual Review

Recommendation

Implement a check to ensure that the allowance is set to zero after each transfer, requiring users to grant only the necessary allowance for each transaction. This limits the risk associated with compromised roles and prevents the contract from having ongoing access to user tokens.

Issue L-2: Promoting does not remove from the player IDs array in `DevelopmentPlayers`

Source: https://github.com/sherlock-audit/2025-08-football-fun-revision-audit-aug-26th/issues/26

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

Summary

Promoting does not remove from the player IDs array in `DevelopmentPlayers`

Vulnerability Detail

When players are cut in `DevelopmentPlayers`, we remove them from the array if locked balance goes to 0:

```
if (lockedBalances[_user][_idsToCut[i]] == 0) {
    uint256[] storage userIds = userPlayerIds[_user];
    for (uint256 j = 0; j < userIds.length; j++) {
        if (userIds[j] == _idsToCut[i]) {
            userIds[j] = userIds[userIds.length - 1];
            userIds.pop();
            break;
        }
    }
}
```

However, when calling `promotePlayers()`, this is not the case. This causes the state to be stale and potential duplicate entries upon the ERC1155 handler as we assume that 0 locked balance means the ID is not in the array:

```
if (lockedBalances[packBuyerAddress][_ids[i]] == 0) {
    // Check that the share is not empty.
    userPlayerIds[packBuyerAddress].push(_ids[i]);
}
```

Impact

State stale and potential duplicate entries.

Code Snippet

https://github.com/sherlock-audit/2025-08-football-fun-revision-audit-aug-26th/blob/16801bd73ac16007566716bebda377492af1f157/fdf-contracts/src/contracts/DevelopmentPlayers.sol#L125-L145

Tool Used

Manual Review

Recommendation

Consider removing from the array when promoting too.

Issue L-3: Sell price calculations can be slightly incorrect due to rounding differences

Source: https://github.com/sherlock-audit/2025-08-football-fun-revision-audit-aug-26th/issues/27

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

Summary

Sell price calculations can be slightly incorrect due to rounding differences

Vulnerability Detail

The actual sell mechanism has a small difference in its calculations compared to the view getter function for determining the sell price.

The `amountToReceive` and `feeAmount` are values returned from the view function and they are computed as follows:

```
amountToReceive = PricingLibrary.sellNumShares(_amountToSell, playerTokenReserve,
↪  currencyReserve, sellFeeRate, FEE_BASIS_POINTS_DENOMINATOR);

uint256 grossAmount = PricingLibrary.sellNumSharesWithoutFee(_amountToSell,
↪  playerTokenReserve, currencyReserve);
feeAmount = (grossAmount * sellFeeRate) / FEE_BASIS_POINTS_DENOMINATOR;
```

`amountToReceive` is correct and it holds the exact amount of tokens that a user would receive if he sells his players, with fee applied. `grossAmount` holds a gross amount, simply based on a `x * y = k` formula. Then, the fee is computed by applying the sell fee to it. However, this is incorrect as the actual fee is computed by subtracting the actual tokens out from the raw/gross amount. This would be 100% correct in normal Maths, but can be incorrect when we factor in Solidity's rounding behavior.

If we ignore the actual sell function's behavior, we can easily prove the view function is incorrect by seeing that `feeAmount + amountToReceive != grossAmount` in some cases, which means something is wrong:

1. `grossAmount = 199`, `fee = 10`, `denom = 100`.
2. `amountToReceive = 199 * (100 - 10) / 100 = 179` (with rounding).
3. `feeAmount = 199 * 10 / 100 = 19` (with rounding).
4. `179 + 19 != 199`.

Impact

Incorrect fee calculation.

Code Snippet

https://github.com/sherlock-audit/2025-08-football-fun-revision-audit-aug-26th/blob/16801bd73ac16007566716bebda377492af1f157/fdf-contracts/src/exchange/FDFPair.sol#L961-L991

Tool Used

Manual Review

Recommendation

When computing the fee amount in the function, consider subtracting the actual amount out from the gross amount instead of doing the division.

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.