SHERLOCK

Security Review For Football.Fun

Collaborative Audit Prepared For: **Football.Fun**
Lead Security Expert(s): **0x3b**
AlexMurphy
Date Audited: **October 29 - November 3, 2025**

Introduction

Sport.Fun is a fantasy sports platform 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 market places and other smart contract based mechanics

Scope

Repository: footballdotfun/sdf-contracts

Audited Commit: acb57be97abd871a619f6a684feb66010c20985c

Final Commit: 16025f777b67630e0be240950199152837cc2b84

Files:

- src/fdf/contracts/core/upgrades/DevelopmentPlayersV2.sol
- src/fdf/contracts/core/upgrades/PlayerContractsV2.sol
- src/fdf/contracts/core/upgrades/PlayerPackV2.sol
- src/fdf/contracts/core/upgrades/PlayerV2.sol
- src/fdf/contracts/exchange/upgrades/FDFPairV2.sol
- src/fdf/contracts/exchange/upgrades/FeeManagerV3.sol
- src/fdf/script/upgrade/core/DevelopmentPlayers/V2/README.md
- src/fdf/script/upgrade/core/PlayerContracts/V2/README.md
- src/fdf/script/upgrade/core/PlayerPack/V2/README.md
- src/fdf/script/upgrade/core/Player/V2/README.md
- src/fdf/script/upgrade/exchange/FDFPair/V2/README.md
- src/fdf/script/upgrade/exchange/FeeManager/V3/README.md

Final Commit Hash

16025f777b67630e0be240950199152837cc2b84

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	1	3

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue M-1: Flash sale fee bypass through token swap recipient manipulation [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-football-fun-oct-29th/issues/8

Vulnerability Detail

The `FDFPairV2.sol` contract implements a flash sale fee mechanism to discourage rapid selling of player tokens. This fee is tracked per tuple of `(playerId, seller)` and enforced through a cooldown period stored in `flashSaleCooldownEnd[_playerId][_seller]` within the fee manager contract. However, the token swap functionality (`_swapTokens()`) incorrectly uses the user-controlled `params.recipient` address when calculating sell fees, rather than the actual seller's address.

When a user initiates a token swap by transferring ERC1155 tokens to the contract, the `onERC1155BatchReceived()` function is invoked with `_from` representing the actual token sender (seller). For regular token sales via `_handleTokenSale()`, this `_from` address is correctly passed to `_swapPlayerTokensForCurrency()` as the `_seller` parameter.

However, in the `_swapTokens()` function, the `params.recipient` field (which is fully controlled by the user as part of the `SwapTokensParams` struct) is incorrectly used as the seller identifier:

```
function _swapTokens(SwapTokensParams memory params) internal returns (bytes4) {

// ...

(currencyReceivedArr, sellFeeAmounts) = _swapPlayerTokensForCurrency(
                params.playerTokenIdsIn,
                params.playerTokenAmountsIn,
                0,
                params.deadline,
                address(this),
                params.recipient // INCORRECT: Using user-controlled recipient
                ↪  instead of actual seller
);

// ...

}
```

The `_seller` parameter is then used in `_swapPlayerTokensForCurrency()` to calculate the sell fee.

Impact

Consider the following scenario:

1. Alice sells a large amount of player token ID 123, triggering a flash sale and activating the flash sale cooldown for the tuple `(123, Alice)`.
2. Alice is now subject to the elevated flash sale fee for subsequent sales of player ID 123 until the cooldown expires.
3. Instead of waiting for the cooldown, Alice initiates a token swap using `_swapTokens()` and sets `params.recipient` to Bob's address (or any other address she controls).
4. The fee calculation incorrectly checks `flashSaleCooldownEnd[123][Bob]` instead of `flashSaleCooldownEnd[123][Alice]`.
5. Since Bob has no active flash sale cooldown for player ID 123, Alice pays only the normal fee instead of the flash sale fee.

Recommendation

The `_swapTokens()` function should pass the actual seller's address rather than the user-controlled recipient address when calling `_swapPlayerTokensForCurrency()`.

Discussion

fedyashevvic

Thanks for finding this edge case, it's indeed valid

Acknowledged and fixed in here - https://github.com/footballdotfun/sdf-contracts/pull/3/commits/ec03300950e00800260a7fa8e3d386be91deae8a

Issue L-1: `currentPlayerTokenReserve` can be calculated in 1 step, instead of 2 [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-football-fun-oct-29th/issues/7

Vulnerability Detail

Inside `FDFPairV2` we have a 2 step process of taking `currentPlayerTokenReserve` and reducing it by the players that were sent to the contract. However the code can be reduced to a much simpler expression (shown in Recommendation).

```
sellVars.currentPlayerTokenReserve = playerTokenReserves[i];

// ...
sellVars.currentPlayerTokenReserve = sellVars.currentPlayerTokenReserve -
↪  sellVars.saleAmount;
```

Impact

Reduced gas cost and cleaner code.

Tool Used

Manual Review

Recommendation

Consider changing the code to:

```
            sellVars.playerId = _playerTokenIds[i];

-           sellVars.currentPlayerTokenReserve = playerTokenReserves[i];
+           sellVars.currentPlayerTokenReserve = playerTokenReserves[i] -
↪  sellVars.saleAmount;

            // Load currency reserves for the playerId.
            sellVars.currentCurrencyReserve =
            ↪  currencyReservesByPlayerId[sellVars.playerId];

            // Tokens are already in the contract, so we need to calculate the
            ↪  original price.
-           sellVars.currentPlayerTokenReserve = sellVars.currentPlayerTokenReserve
↪  - sellVars.saleAmount;
```

Discussion

fedyashevvic

Thanks for finding this optimisation

Acknowledged and fixed in here - https://github.com/footballdotfun/sdf-contracts/pull/3/commits/529a2ad240d85c95f91019c2f93574868707ed33

Issue L-2: Refunds sent to recipient instead of original payer [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-football-fun-oct-29th/issues/9

Summary

The `FDFPairV2` contract contains a refund mechanism in the `_swapCurrencyForPlayerTokens()` execution that sends excess currency to the recipient of the player tokens instead of the original payer. This behavior could lead to confusion if the recipient is different from the payer.

Vulnerability Detail

In the `buyTokens()` function, users can purchase player tokens by specifying a maximum amount of currency they are willing to spend. The `_swapCurrencyForPlayerTokens()` function is responsible for executing the purchase and handling any excess currency. However, the refund of excess currency is sent to the `_recipient` address, which may not be the same as the original payer (`msg.sender`) which is the standard behavior in other protocols.

Impact

The impact of this issue is for the users of the protocol in case they expect to receive any refund in the original payer instead of the recipient of the player tokens which might cause the original buyer to lose funds.

Code Snippet

https://github.com/sherlock-audit/2025-10-football-fun-oct-29th/blob/7486b31cb00bfe5ec3eca49ebd931d7a426b47f8/sdf-contracts/src/fdf/contracts/exchange/upgrades/FDFPairV2.sol#L613

Recommendation

It is recommended to direct the refund to the original payer rather than the recipient. If the current behavior is intentional, it is advisable to clearly document this to ensure users are informed.

Discussion

fedyashevvic

Acknowledged and explained here - https://github.com/footballdotfun/sdf-contracts/pull/3/commits/af241588052fa031c698a59e717300c4e8b8ea57

Issue L-3: Incorrect interface id returned by `supportsInterface()` in `FDFPairV2` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-football-fun-oct-29th/issues/10

Vulnerability Detail

`FDFPairV2` currently returns `type(IFDFPair).interfaceId` from `supportsInterface()`. Because the contract implements the V2 surface, integrations that probe for `IFDFPairV2` via ERC-165 will conclude the pool lacks the V2 API and may refuse to interact or gate upgraded functionality. This breaks interface detection for external integrations and tooling that rely on the advertised interface id.

Recommendation

Update `supportsInterface()` to compare against `type(IFDFPairV2).interfaceId` so the contract advertises the correct ERC-165 identifier for its implemented interface.

Discussion

fedyashevvic

Acknowledged and fixed in here - https://github.com/footballdotfun/sdf-contracts/pull/3/commits/61d6d7f7d678c068f2208502098215824932 9f7f

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.