Three Sigma

# Code Audit

Holofair Token launchpad

# Disclaimer

Code Audit

**Holofair** Token launchpad

# Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

# Table of Contents

Code Audit

**Holofair** Token launchpad

## Table of Contents

# Summary

Code Audit

**Holofair** Token launchpad

# Summary

Three Sigma audited Holofair in a 1.2 person week engagement. The audit was conducted from 19/03/2025 to 21/03/2025.

## Protocol Description

Holofair is a decentralized launchpad for deploying ERC20 tokens with automated presale mechanisms and liquidity provisioning. It integrates with Uniswap V2 to ensure seamless liquidity management and enables fair token distributions.

# Scope

Code Audit

**Holofair** Token launchpad

# Scope

| Filepath | nSLOC |
|---|---|
| src/HoloFair.sol | 217 |
| src/HolofairToken.sol | 196 |
| **Total** | **413** |

## Assumptions

OpenZeppelin library is considered secure.

# Methodology

Code Audit

**Holofair** Token launchpad

# Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

## Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

| Severity / Likelihood | LOW | MEDIUM | HIGH |
|---|---|---|---|
| NONE | None | | |
| LOW | Low | | |
| MEDIUM | Low | Medium | Medium |
| HIGH | Medium | High | High |
| CRITICAL | High | Critical | Critical |

# Project Dashboard

Code Audit

**Holofair** Token launchpad

# Project Dashboard

## Application Summary

| | |
|---|---|
| Name | Holofair |
| Commit | 79dddf4 |
| Language | Solidity |
| Platform | Abstract |

## Engagement Summary

| | |
|---|---|
| Timeline | 19/03/2025 to 21/03/2025 |
| № of Auditors | 2 |
| Review Time | 1.2 person weeks |

## Vulnerability Summary

| Issue Classification | Found | Addressed | Acknowledged |
|---|---|---|---|
| Critical | 0 | 0 | 0 |
| High | 4 | 4 | 0 |
| Medium | 4 | 4 | 0 |
| Low | 0 | 0 | 0 |

| | | | |
|---|---|---|---|
| None | 5 | 5 | 0 |

## Category Breakdown

| | |
|---|---|
| Suggestion | 4 |
| Documentation | 0 |
| Bug | 9 |
| Optimization | 0 |
| Good Code Practices | 0 |

# Findings

Code Audit

**Holofair** Token launchpad

# Findings

## 3S-Holofair-H01

Incorrect vesting duration used in **`claimTeamTokens`** function

| | |
|---|---|
| Id | 3S-Holofair-H01 |
| Classification | High |
| Severity | High |
| Likelihood | High |
| Category | Bug |
| Status | Addressed in #8036def. |

---

### Description

The **`claimTeamTokens`** function is designed to allow the team to claim their allocated tokens after the presale has ended successfully. The function calculates the amount of tokens that can be claimed based on the elapsed time since the end of the fundraising period and the total team allocation. However, there is an issue in the calculation of the **`vestedAmount`**. The function incorrectly caps the **`timeElapsed`** variable using **`presaleVestingDuration`** instead of **`teamVestingDuration`**. This discrepancy can lead to two potential problems: if **`presaleVestingDuration`** is greater than **`teamVestingDuration`**, the team may withdraw more tokens than intended, potentially taking tokens that belong to users. Conversely, if **`presaleVestingDuration`** is less than **`teamVestingDuration`**, some of the tokens allocated to the team may remain locked in the contract indefinitely. The relevant code snippet is as follows:

Code snippet:

```
  function claimTeamTokens() external nonReentrant {
        require(msg.sender == teamWallet, "Only team can claim team
tokens");
        require(!presaleActive, "Presale ongoing");
        require(presaleSuccess, "Presale not successful");
        uint256 totalClaimable = teamAllocation;
        uint256 timeElapsed = block.timestamp - fundraisingEndTime;
        if (timeElapsed > presaleVestingDuration) {
```

```
            timeElapsed = presaleVestingDuration;
        }
        uint256 vestedAmount = (totalClaimable * timeElapsed) /
            teamVestingDuration;
        require(vestedAmount > 0, "No vested tokens available");
        _transfer(address(this), teamWallet, vestedAmount);
        emit TokensWithdrawn(teamWallet, vestedAmount);
    }
```

---

### Recommendation

To resolve this issue, replace **`presaleVestingDuration`** with **`teamVestingDuration`** when capping the **`timeElapsed`** variable.

# 3S-Holofair-H02

## `claimTeamTokens` function allows infinite claims by the team

| | |
|---|---|
| Id | 3S-Holofair-H02 |
| Classification | High |
| Severity | High |
| Likelihood | High |
| Category | Bug |
| Status | Addressed in #8036def. |

---

### Description

The **`claimTeamTokens`** function in the **`HolofairToken`** contract is designed to allow the team to claim their allocated tokens after the presale has ended successfully. The function calculates the amount of tokens that can be claimed based on the elapsed time since the end of the fundraising period and the total team allocation. However, the function does not track the amount of tokens that have already been claimed by the team. This oversight allows the team to repeatedly claim the full vested amount, potentially depleting the token supply allocated to users. The function should instead maintain a record of the claimed tokens and subtract this from the **`vestedAmount`** to determine the correct amount of tokens the team should receive in each claim.

Code snippet:

```
  function claimTeamTokens() external nonReentrant {
        require(msg.sender == teamWallet, "Only team can claim team
tokens");
        require(!presaleActive, "Presale ongoing");
        require(presaleSuccess, "Presale not successful");
        uint256 totalClaimable = teamAllocation;
        uint256 timeElapsed = block.timestamp - fundraisingEndTime;
        if (timeElapsed > presaleVestingDuration) {
            timeElapsed = presaleVestingDuration;
        }
        uint256 vestedAmount = (totalClaimable * timeElapsed) /
            teamVestingDuration;
        require(vestedAmount > 0, "No vested tokens available");
        _transfer(address(this), teamWallet, vestedAmount);
```

```
        emit TokensWithdrawn(teamWallet, vestedAmount);
    }
```

---

### Recommendation

Modify the **`claimTeamTokens`** function to include a mechanism for tracking the total amount of tokens claimed by the team. This can be achieved by introducing a new state variable, such as **`claimedTeamTokens`**, which is updated each time the team claims tokens. The **`vestedAmount`** should then be calculated by subtracting **`claimedTeamTokens`** from the total claimable amount. This ensures that the team can only claim the tokens they are entitled to, preventing any potential abuse of the function.

# 3S-Holofair-H03

User deposit balance incorrectly zeroed after partial token claim

| | |
|---|---|
| Id | 3S-Holofair-H03 |
| Classification | High |
| Severity | High |
| Likelihood | High |
| Category | Bug |
| Status | Addressed in #4c21ef9. |

---

**Description:**

The **`claimTokens`** function is designed to allow users to claim their vested tokens after a successful presale. The function calculates the amount of tokens a user is entitled to based on their deposit and the elapsed time since the end of the fundraising period. However, there is a critical issue in the implementation: the user's deposit balance is set to zero after a claim, even if the claim is only partial. This means that users lose their remaining deposit balance and are unable to claim any further tokens they are entitled to as they vest over time.

Code snippet:

```
function claimTokens() external nonReentrant {
    require(!presaleActive, "Presale ongoing");
    require(presaleSuccess, "Presale not successful");
    uint256 userDeposit = deposits[msg.sender];
    require(userDeposit > 0, "No tokens to claim");
    uint256 totalClaimable = (userDeposit * presaleAllocation) /
        totalRaised;
    uint256 timeElapsed = block.timestamp - fundraisingEndTime;
    if (timeElapsed > presaleVestingDuration) {
        timeElapsed = presaleVestingDuration;
    }
    uint256 vestedAmount = (totalClaimable * timeElapsed) /
        presaleVestingDuration;
    require(vestedAmount > 0, "No vested tokens available");
    deposits[msg.sender] = 0;
    _transfer(address(this), msg.sender, vestedAmount);
```

```
        emit TokensWithdrawn(msg.sender, vestedAmount);
    }
```

---

### Recommendation:

Modify the **`claimTokens`** function to track the amount of tokens already withdrawn by the user. Instead of zeroing out the user's deposit balance, maintain a record of the total tokens claimed so far. Subtract this amount from the **`vestedAmount`** to determine the number of tokens the user should receive in the current claim. This ensures that users can continue to claim their vested tokens over time without losing their remaining balance.

# 3S-Holofair-H04

Uniswap V2 pair poisoning DoS vulnerability in liquidity migration

| | |
|---|---|
| Id | 3S-Holofair-H04 |
| Classification | High |
| Severity | High |
| Likelihood | Medium |
| Category | Bug |
| Status | Addressed in #78c5bae. |

---

## Description

The **`HolofairToken`** contract contains a vulnerability in its liquidity migration process that allows denial-of-service (DoS) through require statements in Uniswap V2's liquidity calculations.

1. Pair Creation with Zero Reserves:

```
address liquidityPool = IUniswapV2Factory(
    IUniswapV2Router02(dexRouter).factory()
).createPair(tokenAddr, IUniswapV2Router02(dexRouter).WETH());
```

This creates a pair with initial reserves of (0, 0) for (token, WETH).

2. Attacker Poisoning:

- Attacker sends 1 wei of WETH to the pair

- Attacker calls `sync()` updating reserves to (0, 1 wei)

3. Quoting issue:

When adding liquidity, the router calculates optimal amounts using `quote()`:

```
function quote(uint amountA, uint reserveA, uint reserveB) internal pure
returns (uint amountB) {
    require(amountA > 0, 'UniswapV2Library: INSUFFICIENT_AMOUNT');
    require(reserveA > 0 && reserveB > 0, 'UniswapV2Library:
INSUFFICIENT_LIQUIDITY');
```

```
    amountB = amountA.mul(reserveB) / reserveA;
}
```

With reserves (0, 1 wei), any liquidity addition attempt will revert at this line **`require(reserveA > 0 && reserveB > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');`** .

This attack doesn't provide direct profit to the attacker but creates a griefing vector that permanently blocks the presale's progression to the trading phase. Once the pair is poisoned, the presale cannot be finished successfully, forcing users to withdraw their funds. Since no tokens are available before the liquidity migration, it is impossible to fix the poisoned pair after an attack occurs.

---

## Recommendation

The correct solution for this vulnerability is to bypass the Uniswap Router and interact directly with the Pair contract:

1. **Transfer tokens and ETH directly to the pair**: Instead of using the Router's functions which would fail, directly transfer both assets to the pair contract.

2. **Call the pair's [`mint()`](https://github.com/Uniswap/v2-core/blob/master/contracts/UniswapV2Pair.sol#L110C2-L131C6) function**: After transferring both assets, call the pair's **`mint()`** function to create LP tokens correctly.

# 3S-Holofair-M01

Lack of validation in **`deployTokenWithCustomParams`** function

| | |
|---|---|
| Id | 3S-Holofair-M01 |
| Classification | Medium |
| Severity | Medium |
| Likelihood | High |
| Category | Bug |
| Status | Addressed in #d70464f. |

---

**Description:**

The **`deployTokenWithCustomParams`** function in the **`Holofair`** contract allows creators to deploy tokens with custom presale parameters. However, it lacks critical validations that are present in the **`addPresaleParams`** function. Specifically, the function does not verify that the sum of **`dexAllocation`**, **`teamAllocation`**, and **`presaleAllocation`** equals **`totalSupply`**. This oversight can result in insufficient tokens for team and user claims, potentially leading to a first-come-first-serve scenario and a denial of service if **`totalSupply`** is less than **`dexAllocation`**.

Additionally, the function does not check that the sum of **`ethToTeam`** and **`ethToDex`** equals **`fundraisingTarget`**, which can lead to a denial of service if **`ethToDex`** is less than **`fundraisingTarget`**. If the sum of **`ethToTeam`** and **`ethToDex`** is lower than **`fundraisingTarget`**, some funds will remain in the contract after the liquidity migration; at this point the creator can partially siphon them out by restarting the presale and withdrawing his deposits.

Furthermore, the parameters **`ethToTeam`** and **`teamAllocation`** are not percentage-based and fixed within the contract, allowing creators to set them to zero. This allows creators to bypass the team revenue stream. The **`teamVestingDuration`** is also not fixed, which could be abused by creators to lock team's tokens being vested by either setting **`teamVestingDuration`** to an unreasonably large value, or to zero which will cause a Denial of Service in the **`claimTeamTokens`** function due to a division by zero.

---

**Recommendation:**

To address these issues, implement the same validation checks in the **`deployTokenWithCustomParams`** function as those in the **`addPresaleParams`** function. Ensure that the sum of **`dexAllocation`**, **`teamAllocation`**, and **`presaleAllocation`** equals **`totalSupply`**, and that **`ethToTeam`** plus **`ethToDex`** equals **`fundraisingTarget`**. Additionally, consider making **`ethToTeam`** and **`teamAllocation`** percentage-based and fixed within the contract to prevent creators from setting them to zero. Bound the **`teamVestingDuration`** to a reasonable value to prevent abuse.

# 3S-Holofair-M02

**withdrawDeposit** function allows indefinite locking of user funds

| | |
|---|---|
| Id | 3S-Holofair-M02 |
| Classification | Medium |
| Severity | High |
| Likelihood | Low |
| Category | Bug |
| Status | Addressed in #223b26c. |

## Description

The **withdrawDeposit** function in the **HolofairToken** contract is designed to allow users to withdraw their deposited funds during the presale. However, the current implementation restricts withdrawals to only when the presale is active. This means that if the presale is paused, users are unable to withdraw their funds. Since the presale can be paused indefinitely, this creates a risk where users' deposited funds could be locked forever.

Code snippet:

```
    function withdrawDeposit(uint256 amount) external nonReentrant {
        require(presaleActive, "Presale not active");
        uint256 userDeposit = deposits[msg.sender];
        require(userDeposit >= amount, "Insufficient deposit");
        // Reset the user's deposit before transferring to prevent
reentrancy
        deposits[msg.sender] -= amount;
        totalRaised -= amount;
        (bool success, ) = msg.sender.call{value: amount}("");
        require(success, "Withdrawal failed");
        emit WithdrawDeposit(msg.sender, 0, amount);
    }
```

### Proof of Concept (PoC)

See the PoC here.

### Recommendation

Modify the **`withdrawDeposit`** function to allow users to withdraw their funds even when the presale is paused. This can be achieved by removing the **`require(presaleActive, "Presale not active");`** line from the function.

# 3S-Holofair-M03

Lack of **`resumePresale`** function leads to incorrect **`fundraisingEndTime`** adjustment

| | |
|---|---|
| Id | 3S-Holofair-M03 |
| Classification | Medium |
| Severity | Medium |
| Likelihood | High |
| Category | Bug |
| Status | Addressed in #9370598. |

---

## Description

The **`HolofairToken`** contract includes a **`pausePresale`** function that allows the creator to pause the presale. However, the only way to resume the presale after it has been paused is by calling the **`startPresale`** function. This approach inadvertently resets the **`fundraisingEndTime`** by adding the **`fundraisingPeriod`** to the current block timestamp, effectively extending the presale duration. This extension can disrupt the vesting schedule for users who have already deposited funds, as their vesting unlock time is tied to the original **`fundraisingEndTime`**.

Code snippet:

```
function startPresale() external {
    require(msg.sender == creator, "Only creator can start presale");
    require(!presaleActive, "Presale already started");
    fundraisingEndTime = block.timestamp + fundraisingPeriod;
    presaleActive = true;
}
function pausePresale() external {
    require(msg.sender == creator, "Only creator can pause presale");
    require(presaleActive, "Presale already paused");
    presaleActive = false;
}
```

### Proof of Concept (PoC)

See the PoC here.

---

### Recommendation

Implement a **`resumePresale`** function that allows the presale to be resumed without modifying the **`fundraisingEndTime`**.

# 3S-Holofair-M04

Vulnerability in **`startPresale`** function allows indefinite token lock

| | |
|---|---|
| Id | 3S-Holofair-M04 |
| Classification | Medium |
| Severity | High |
| Likelihood | Low |
| Category | Bug |
| Status | Addressed in #9370598. |

## Description

The **`startPresale`** function in the **`HolofairToken`** contract is designed to initiate the presale process, setting the **`presaleActive`** state to **`true`** and establishing the **`fundraisingEndTime`** based on the current block timestamp and the predefined **`fundraisingPeriod`**. However, the function lacks a critical check to ensure that the presale has not already been marked as successful (**`presaleSuccess`** is **`false`**). This oversight allows a malicious creator to restart the presale even after it has concluded successfully and liquidity has been migrated to Uniswap V2. By doing so, the creator can lock the tokens indefinitely, as both the **`claimTokens`** and **`claimTeamTokens`** functions require **`presaleActive`** to be **`false`** for token claiming to proceed. Furthermore, by calling **`pausePresale`** followed by **`startPresale`** the creator is able to extend the presale duration, allowing the creator to lock the tokens indefinitely.

### Proof of Concept (PoC)

See the PoC here.

## Recommendation

To prevent the indefinite locking of tokens, introduce a check in the **`startPresale`** function to ensure that **`presaleSuccess`** is **`false`** before allowing the presale to be restarted.

# 3S-Holofair-N01

Missing zero address check in **`setFeeRecipient`** function

| | |
|---|---|
| Id | 3S-Holofair-N01 |
| Classification | None |
| Category | Suggestion |
| Status | Addressed in 4222984. |

## Description

The **`setFeeRecipient`** function in the **`Holofair`** contract is designed to allow the contract owner to update the address that receives fees collected by the contract. However, the current implementation lacks a check to prevent the **`feeRecipient`** from being set to the zero address (**`address(0)`**). If the **`feeRecipient`** is set to the zero address, any fees sent to this address will be irretrievably lost, as the zero address is not a valid recipient for Ether or tokens. This could result in a loss of funds that are meant to be collected as fees.

## Recommendation

Implement a check within the **`setFeeRecipient`** function to ensure that the **`_feeRecipient`** is not set to the zero address.

# 3S-Holofair-N02

CEI violation in **`deployToken`** and **`deployTokenWithCustomParams`** functions

| | |
|---|---|
| Id | 3S-Holofair-N02 |
| Classification | None |
| Category | Bug |
| Status | Addressed in b879504. |

## Description

The **`deployToken`** and **`deployTokenWithCustomParams`** functions in the **`Holofair`** contract are responsible for deploying new tokens and setting up their initial parameters, including creating a liquidity pool on a decentralized exchange. These functions currently suffer from a Checks-Effects-Interactions (CEI) pattern violation. The CEI pattern is a best practice in smart contract development that suggests performing all checks, then updating state variables, and finally making external calls. This pattern helps prevent reentrancy attacks and other unexpected behaviors.

In both functions, an external call to transfer the launch fee to the **`feeRecipient`** is made before updating critical state variables such as **`tokenRecords`**, **`creatorTokenIds`**, and **`tokenLiquidityPool`**. This external call could potentially allow for reentrancy or other unintended side effects before the contract's state is fully updated.

## Recommendation

To adhere to the CEI pattern and enhance the security of the contract, it is recommended to move the external call to transfer the launch fee to the **`feeRecipient`** to the end of the **`deployToken`** and **`deployTokenWithCustomParams`** functions.

# 3S-Holofair-N03

Unnecessary **`receive`** function in Holofair contract

| | |
|---|---|
| Id | 3S-Holofair-N03 |
| Classification | None |
| Category | Suggestion |
| Status | Addressed in c157212. |

## Description

The **`receive`** function in the **`Holofair`** contract is designed to allow the contract to accept Ether transfers directly. This function is automatically invoked when the contract receives Ether without any accompanying data. However, in the context of the **`Holofair`** contract, there is no apparent need for the contract to accept direct Ether transfers. The presence of this function could lead to users mistakenly sending Ether to the contract, resulting in unintended loss of funds.

## Recommendation

Remove the **`receive`** function from the **`Holofair`** contract to prevent accidental Ether transfers.

# 3S-Holofair-N04

Inconsistency in **`WithdrawDeposit`** event parameters

| | |
|---|---|
| Id | 3S-Holofair-N04 |
| Classification | None |
| Category | Suggestion |
| Status | Addressed in 1c57c4c. |

## Description

The **`withdrawDeposit`** function in the **`HolofairToken`** contract allows users to withdraw their deposits if the presale is active. The function emits a **`WithdrawDeposit`** event, which is defined as **`event WithdrawDeposit(address indexed user, uint256 amount, uint256 refund);`**. However, the event is emitted with the parameters **`emit WithdrawDeposit(msg.sender, 0, withdrawnAmount);`**, where **`withdrawnAmount`** is the amount being withdrawn by the user. This creates ambiguity regarding the purpose of the **`amount`** field in the event, as it is always set to zero. The **`refund`** field is used to indicate the withdrawn amount, but it is unclear what the **`amount`** field should represent. This inconsistency can lead to confusion and misinterpretation of the event data. The client should either remove the redundant **`amount`** field from the event or clarify its intended use, potentially as the remaining user balance after withdrawal.

## Recommendation

To improve clarity and code quality, consider removing the **`amount`** field from the **`WithdrawDeposit`** event if it is not needed. Alternatively, if the **`amount`** field is intended to represent the remaining user balance after withdrawal, update the event emission to reflect this. Ensure that the event parameters accurately convey the intended information to avoid confusion.

# 3S-Holofair-N05

Misleading Documentation for **`ethToDex`** and **`ethToTeam`** Variables

| | |
|---|---|
| Id | 3S-Holofair-N05 |
| Classification | None |
| Category | Suggestion |
| Status | Addressed in 7e087f7. |

## Description

In the **`HolofairToken`** contract, the variables **`ethToDex`** and **`ethToTeam`** are intended to represent the absolute values of ETH allocated to the decentralized exchange (DEX) and the team wallet, respectively. However, the comments in the code incorrectly describe these variables as percentages. This discrepancy between the code comments and the actual functionality of the variables can lead to misunderstandings for developers and auditors reviewing the contract. The relevant code snippet is as follows:

```
uint256 public ethToDex; // Percentage of ETH allocated to DEX
uint256 public ethToTeam; // Percentage of ETH allocated to team wallet
```

## Recommendation

Update the comments for the **`ethToDex`** and **`ethToTeam`** variables to accurately reflect that they represent absolute values rather than percentages.