Smart Contract Audit Report

1. Overview

- **Project Name:** BonFire
- **Contracts Audited:**
    - File: BFX.sol
    - File: BonFireBuyAndBurn.sol
    - File: BonFireLpAdder.sol
    - File: BonFireUIHelper.sol
    - File: BonAndBurnConst.sol
- **Code base:** Provided by File

2. Executive Summary

This audit aims to assess the security of the smart contracts for the ICO protocol. We identified issues:

- High Risk
- Medium Risk
- Low Risk
- Informational / Gas Optimization Suggestions

All high and medium severity issues have been addressed / partially addressed / not addressed by the development team at the time of publishing this report.

3. Methodology

Our audit involved the following steps:

1. **Manual Code Review** – Analyzed logic for correctness, authorization, error handling, etc.
2. **Automated Testing** – Used static analysis tools such as Slither, MythX, or Hardhat plugins.
3. **Gas Usage Review** – Identified areas for optimization.
4. **Dependency Analysis** – Assessed third-party libraries (e.g., OpenZeppelin).

4. Risk Classification

Severity	Description
High	Critical vulnerabilities that can lead to loss.
Medium	Significant issues but not critical.
Low	Minor issues, often best practices.
Informational	No direct risk but useful suggestions.

5. Inheritance Graph

NA

6. Ownership Privileges

BFX.sol

- The buy and burn contract address can be deposited into the recycle pool.
- The buy-and-burn contract address can start a new cycle.
- The genesis address and dev address can update the distribution percentage between 1 to 10%.
- The genesis and dev address can update the buy and burn contract address.
- **BonFireBuyAndBurn.sol**
- The genesis or dev address can set the LP add threshold value.
- The genesis address or dev address can set the burn settings in the contract.
- The genesis and dev address can update the slippage in the contract.
- The LP adder can increase liquidity.
- **Note** - This Audit report consists of a security analysis of the **BonFire** smart contract. This analysis did not include functional testing (or unit testing) of the contract's logic. Moreover, we only audited one token contract for the **BonFire** team. Other contracts associated with the project were not audited by our team. We recommend investors do their own research before investing.

7. Findings

7.1 High Severity

[H-01] Missing access control.

- **Description:** The contract contains the functionality in which any user can burn the tokens from the contract. To address the lack of access control, we can restrict who can call the burnBonFire function.
- **Mitigation:** Access control ensures that only authorized entities (e.g., the contract owner or a specific role) can execute this function, preventing unauthorized burns.
- File: BONFIREBUYANDBURN.SOL
- Line: L341-346
- **Status:** Open

7.2 Medium Severity

[M-01] Potential Rounding Errors

- **Description:** In Solidity, integer division truncates fractional results, causing potential rounding errors in reward calculations like (totalVesting * userSBP) / totalSBP. If the numerator isn't an exact multiple of the denominator, the fractional part is discarded, slightly reducing the reward for users. Over multiple cycles, this truncation can lead to cumulative imbalances, particularly for users with smaller shares.
- **Mitigation:** To mitigate this, contracts can use scaling factors (e.g., multiplying by 10**18 to increase precision), track and redistribute remainders, or accept small rounding losses while ensuring transparency and fairness in the distribution process through proper documentation and state adjustments.
- File: BFX.SOL
- Line: L477-485
- **Status:** Open

[M-02] Missing 'isContract' check.

- **Description:** The contract lacks a validation check to ensure that specific parameters are contract addresses. Without this check, there is a risk that non-contract addresses (such as externally owned accounts, or EOAs) could be mistakenly set for parameters intended to reference other contracts. This could lead to failures in executing critical interactions within the contract, as EOAs do not support contract-specific functions

- **Mitigation:** To mitigate this, Implement a validation check to ensure that parameters designated as contract addresses are verified as such. This can be done using Solidity's Address library function isContract, which checks if an address has associated contract code.
- File: BFX.SOL
- Line: L507-512
- **Status:** Open

[M-03] Missing Non-reentrant check.

- **Description:** The createX28BonfirePool function could face reentrancy risks during external calls to the Uniswap V3 manager or malicious token contracts. Reentrancy occurs if a callback is triggered before the function execution completes, allowing attackers to manipulate state. For instance, a malicious _x28 token might reenter the function during createAndInitializePoolIfNecessary, altering token balances or state.
- **Mitigation:** To mitigate this, use the Checks-Effects-Interactions pattern, update state before external calls, implement a nonReentrant modifier, and validate tokens. Additionally, ensure external contracts, like Uniswap, are trusted. Defensive coding reduces vulnerabilities by securing state and limiting reliance on external dependencies in the same transaction.
- File: BFX.SOL
- Line: L558-593
- **Status:** Open

[M-04] Missing Threshold

- **Description:** The contract allows the lpAddThreshold value to be set to any arbitrary number, including zero. This is problematic, as setting the value to zero would prevent liquidity from being added.
- **Mitigation:** To avoid unintended behaviour, the contract should enforce a minimum threshold value, ensuring that the lpAddThreshold cannot be set below this defined limit.
- File: BONFIREBUYANDBURN.SOL
- Line: L276-279
- **Status:** Open

[M-05] Missing Non-reentrant check.

- **Description:** The addLiquidityToBonFireX28Pool function is vulnerable to re-entrancy, where an external contract could re-enter the function before the liquidityAdded flag is set, potentially bypassing access checks or causing inconsistent state.
- **Mitigation:** To mitigate this, the ReentrancyGuard library from OpenZeppelin can be used with the nonReentrant modifier, preventing any nested calls until the initial function call completes. Additionally, state updates like setting liquidityAdded should be done before interacting with external contracts, further minimizing the risk. These measures together protect against re-entrancy attacks and ensure the function's integrity.
- File: BONFIREBUYANDBURN.SOL
- Line: L221-270
- **Status:** Open

[M-06] Fixed Slippage (10%).

- **Description:** The 10% slippage tolerance is hardcoded, which might be too high or too low for certain transactions. Fixed slippage could lead to unnecessary failed transactions or suboptimal trades. This can also cause the lack of configurability as the slippage is hardcoded, there's no flexibility for the contract owner or the caller to adjust this tolerance based on market conditions or specific requirements.
- **Mitigation:** Instead of hardcoding the 10% slippage, allow the slippage to be passed as a parameter to the function or set it as a configurable contract-level variable.
- File: BONFIREBUYANDBURN.sol
- Line: L1035-1063
- **Status:** Open

[M-07] Deadline Mismanagement

- **Description:** The _deadline parameter is passed to the position manager, but there is no check in the function to ensure it is valid (e.g., a timestamp in the future).
- **Mitigation:** To mitigate this, Validate _deadline by comparing it against the current block timestamp (block.timestamp).
- File: BONFIREBUYANDBURN.SOL
- Line: L221-270, L393-438
- **Status:** Open

[M-08] Regain ownership

- **Description:** The owner can regain ownership after transferring it with the following steps: 1. Call the lock function to set _previousOwner to the owner's address. 2. Call the unlock function to get ownership back. 3.Transfer/renounce ownership 4. Call the unlock function to get ownership back. Make sure to set the _previousOwnership back to address zero after using the unlock function.
- File: BONFIREBUYANDBURN.SOL
- Line: L15
- **Status:** Open

7.3 Low Severity

[L-01] Lack of Input Token Validation

- **Description:** The inputToken is validated only against allowedTokens for unsupported tokens, but there's no sanity check to ensure it is non-zero or not an unsupported address (e.g., zero address). This could lead to undefined behaviour or vulnerabilities in downstream logic.
- **Mitigation:** To mitigate this, Validate inputToken != address(0) and add further checks to ensure it adheres to expected standards.
- File: BFX.SOL
- Line: L154-256
- **Status:** Open

[L-02] Missing events arithmetic

- **Description:** It is recommended to emit all the critical parameter changes.
- File: BFX.SOL
- Line: L488-491, L494-497, L500-504, L507-512, L515-523, L533-537, L540-548
- **Status:** Open

[L-03] Missing zero or dead address check.

- **Description:** It is recommended to check that the address not be zero or a dead address.
- File: BFX.SOL
- Line: L558-593
- **Status:** Open

[L-04] Missing Visibility

- **Description:** It is recommended to add 'public,' 'private,' or 'internal' visibility during the declaration or initialization of a state variable or a mapping in the contract.
- File: BONFIREBUYANDBURN.SOL
- Line: L67, L110-14
- **Status:** Open

[L-05] Improper approval Management

- **Description:** The TransferHelper.safeApprove calls overwrite any existing approvals without checking. Use safeApprove only when resetting approvals to 0 before setting new amounts.
- File: BONFIREBUYANDBURN.SOL
- Line: L244, L245, L417, L418
- **Status:** Open

[L-06] No Validation of _deadline

- **Description:**The _deadline parameter is passed to increaseLiquidity, but there's no validation to ensure that it's a future timestamp. A past deadline would cause the transaction to fail, but the user won't receive any helpful feedback or rejection at the time of the call. Add validation for _deadline to ensure it is in the future.
- File: BONFIRELPADDER.SOL
- Line: L50-62
- **Status:** Open

7.4 Informational/Optimization Severity

[I-01] High cardinality value

- **Description:** In Uniswap V3, increasing observation cardinality reserves memory for price observations, which incurs high gas costs proportional to the increase. Setting it directly to 100 allocates all memory upfront, leading to significant expenses. Instead, starting with a smaller cardinality (e.g., 20 or 50) reduces initial costs and meets early usage needs. Cardinality can be incrementally increased later as required, distributing gas costs over multiple transactions. This approach optimizes gas usage while ensuring the pool has sufficient observation capacity for its current operational requirements, making it a cost-effective and practical strategy for managing Uniswap V3 pool configurations..
- File: BFX.SOL
- Line: 558-593
- **Status:** Open

[I-02] Public function that could be declared external (external-function)

- **Description:** Use the `external` attribute for functions never called from the contract.
- File: BONFIRELPADDER.SOL
- Line: 50-62
- **Status:** Open

8. Conclusion

The smart contracts are generally well-written and follow modern best practices. After addressing the findings noted, the contracts should be considered safe for deployment, assuming no changes are made without further auditing.

9. Disclaimer

This audit report is not the final deployed version of the contract. So, any changes made after the audit will be considered out of the scope of the audit, and we will not be responsible for any changes related to that.