Keep3rV2Oracle and Keep3rV2OracleFactory Smart Contract Audit Report

This report summarizes the findings of an audit conducted on the Keep3rV2Oracle and Keep3rV2OracleF

1. Code Size Exceeding Limits

* **Severity**: Medium
* **Description**: The contract code size exceeds 24576 bytes, a limit introduced in Solidity's Spurious Dr
* **Impact**: The contract might be unable to be deployed on mainnet due to the code size exceeding the
* **Mitigation**: Consider optimizing the code by using libraries, removing unnecessary code, or enabling

2. Uninitialized Local Variables

* **Severity**: Medium
* **Description**: Several local variables within the `Keep3rV2Oracle` contract are declared but never init
* **Impact**: Accessing uninitialized local variables can lead to unpredictable behavior and potential vuln
* **Mitigation**: Ensure that all local variables are properly initialized before they are used.

3. Unused Return Values

* **Severity**: Medium
* **Description**: The `Keep3rV2OracleFactory` contract ignores the return values of the `update` functio
* **Impact**: Ignoring return values can obscure potential errors or issues that might occur during externa
* **Mitigation**: Always handle the return values of external calls to ensure proper error handling and pre

4. Missing Zero-Address Validation

* **Severity**: Medium
* **Description**: The `Keep3rV2Oracle` constructor and the `Keep3rV2OracleFactory` `setGovernance`
* **Impact**: Using the zero address can lead to function failures or other unpredictable behavior.
* **Mitigation**: Validate that all addresses passed as arguments to the contract functions are non-zero.

5. External Calls Within Loops

* **Severity**: Medium
* **Description**: The `Keep3rV2OracleFactory` contract has external calls within loops in the `work`, `wo
* **Impact**: This can lead to increased gas consumption and potentially introduce reentrancy vulnerabili
* **Mitigation**: Consider restructuring the code to minimize external calls within loops and implement ap

6. Potential Integer Overflow and Underflow Attacks

* **Severity**: Low
* **Description**: While the contract does not explicitly use SafeMath or similar libraries, the arithmetic o
* **Impact**: An integer overflow or underflow could lead to incorrect results or potentially allow attackers
* **Mitigation**: Implement proper error handling and use SafeMath or similar libraries for all arithmetic o

7. Potential Improper Access Control

* **Severity**: Low
* **Description**: The `factory` and `keeper` modifiers rely on the `_factory` state variable and the `keepe
* **Impact**: An attacker could gain control of functions that require the `factory` or `keeper` roles, potent

* **Mitigation**: Ensure that the `_factory` state variable and the `keepers` function are appropriately prot

8. Potential DoS Attacks

* **Severity**: Low
* **Description**: The `work` and `workForFree` functions in the `Keep3rV2OracleFactory` contract loop o
* **Impact**: A DoS attack could prevent the contract from being updated, impacting the functionality of th
* **Mitigation**: Consider implementing mechanisms to limit the number of pairs processed within a singl

9. Potential Front-Running Attacks

* **Severity**: Low
* **Description**: The contract lacks mechanisms to prevent front-running attacks, where attackers could
* **Impact**: Front-running attacks could lead to price manipulation or other unwanted outcomes.
* **Mitigation**: Consider implementing mechanisms to prevent front-running attacks, such as using a de

10. Assembly Usage

* **Severity**: Low
* **Description**: The `Keep3rV2OracleFactory` contract uses assembly in the `deploy` function.
* **Impact**: While assembly can be useful for optimization, it can also be harder to audit and maintain, p
* **Mitigation**: Consider replacing assembly with more standard Solidity code whenever possible to imp

11. Solidity Pragma Version

* **Severity**: Low
* **Description**: The contract uses the `pragma solidity ^0.8.2` directive, which allows older versions of
* **Impact**: Using older versions of Solidity can introduce vulnerabilities that have been addressed in ne
* **Mitigation**: Upgrade the contract to use the latest recommended Solidity version and ensure that the

12. Naming Conventions

* **Severity**: Low
* **Description**: Several variable and function names within the contract do not follow recommended So
* **Impact**: Inconsistent naming conventions can make the code harder to read and understand, potent
* **Mitigation**: Ensure that all variables, functions, and constants adhere to standard Solidity naming co

13. State Variables That Could Be Constant

* **Severity**: Low
* **Description**: The `Keep3rV2Oracle` contract's `Q112` and `e10` variables could be declared as cons
* **Impact**: Using state variables when constants are sufficient can potentially increase gas usage and
* **Mitigation**: Declare the `Q112` and `e10` variables as constants if they are not intended to be modifi

Recommendations

1. Address the code size exceeding limits issue to ensure deployability on mainnet.
2. Ensure proper initialization of all local variables before use.
3. Handle return values of external calls appropriately.
4. Validate that all addresses passed as arguments are non-zero.
5. Reorganize the code to minimize external calls within loops and implement reentrancy guards.

6. Implement proper error handling and consider using SafeMath for all arithmetic operations.
7. Review and strengthen access control mechanisms around the `_factory` and `keepers` functions.
8. Implement measures to mitigate potential DoS attacks, such as limiting the number of pairs processed
9. Consider techniques to prevent front-running attacks.
10. Replace assembly with standard Solidity code whenever possible.
11. Upgrade the contract to use the latest recommended Solidity version and update the `pragma` directi
12. Ensure consistent naming conventions throughout the code.
13. Declare variables as constants whenever possible.

This report provides a comprehensive analysis of the potential vulnerabilities in the Keep3rV2Oracle and