// Security Assessment

04.30.2025 - 05.02.2025

Azos Core

Azos Finance

HALBORN

Azos Core - Azos Finance

Prepared by: HALBORN

Last Updated 05/20/2025

Date of Engagement: April 30th, 2025 - May 2nd, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
3	0	0	1	0	2

TABLE OF CONTENTS

1. Introduction

`Azos Finance` engaged `Halborn` to conduct a security assessment on their smart contracts beginning on April 30th, 2025 and ending on May 2nd, 2025. The security assessment was scoped to the smart contracts provided to Halborn. Commit hashes and further details can be found in the Scope section of this report.

The `Azos Finance` codebase in scope consists of a a smart contract responsible for fetching and properly handling prices from a DIA oracle, as well as as the interface based on the DIA oracle and a deployment script.

2. Assessment Summary

`Halborn` was provided 3 days for the engagement and assigned a full-time security engineer to review the security of the smart contracts in scope.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by the `Azos Finance team`. The main one was the following:

- `Change the variable to not be immutable and implement a setter function for it where the event is emitted.`

3. Test Approach And Methodology

`Halborn` performed a manual review of the code. Manual testing is great to uncover flaws in logic, process, and implementation.

The following phases and associated tools were used throughout the term of the assessment:

- Research into architecture, purpose and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Solidity variables and functions in scope that could led to arithmetic related vulnerabilities.

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: azos-core

(b) Assessed Commit ID: b45dc48

(c) Items in scope:

- DIARelayerV2.sol
- IDIAOraclev2.sol
- Deploy.s.sol

Out-of-Scope: Third party dependencies and economic attacks.

REMEDIATION COMMIT ID:

- 95e76b1

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	0	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INCONSISTENCY BETWEEN THE DESIRED IMPLEMENTATION AND THE ACTUAL IMPLEMENTATION	MEDIUM	SOLVED - 05/16/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
FLOATING PRAGMA	INFORMATIONAL	ACKNOWLEDGED - 05/16/2025
MULTIPLE UNUSED MAPPINGS ACROSS THE CODE	INFORMATIONAL	SOLVED - 05/16/2025

7. FINDINGS & TECH DETAILS

7.1 INCONSISTENCY BETWEEN THE DESIRED IMPLEMENTATION AND THE ACTUAL IMPLEMENTATION

// MEDIUM

Description

`DIARelayerV2` has the following event and comment above it:

```
/// @notice Emitted when the stale threshold is updated
/// @param _threshold New threshold value in seconds
event NewStaleThreshold(uint256 _threshold);
```

Whenever the staleness threshold is updated, the event must be emitted. However, there are a few issues with that. Firstly, the event is completely unused, as there is no implemented function to change the threshold at all. Secondly, the threshold is actually immutable:

```
uint256 public immutable STALE_THRESHOLD;
```

This means that the staleness threshold cannot be changed even if we tried implementing a setter function for it.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:M/D:N/Y:N (5.0)

Recommendation

Consider changing the variable to not be immutable and implementing a setter function for it where the event is emitted.

Remediation Comment

SOLVED: The **Azos Finance team** solved the issue by implementing a function to change the stale threshold value and making the stale threshold value mutable.

Remediation Hash

https://github.com/AzosFinance/azos-core/commit/95e76b161a79717353b666a3e627cf725385236e

7.2 FLOATING PRAGMA

// INFORMATIONAL

Description

The 3 in-scope files (`DIARelayerV2` , `IDIAOraclev2` , `Deploy.s.sol`) currently use floating pragma versions `^0.8.20` which means that the code can be compiled by any compiler version that is greater than or equal to `0.8.20` , and less than `0.9.0` .

However, it is recommended that contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma. For example, an outdated pragma version might introduce bugs that affect the contract system negatively.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

Consider locking the pragma version.

Remediation Comment

ACKNOWLEDGED: The contracts wouldn't compile while working on phase2 parts of development.

7.3 MULTIPLE UNUSED MAPPINGS ACROSS THE CODE

// INFORMATIONAL

Description

In the `DIARelayerV2` contract, there are a few mappings which are never accessed and used, namely the `values` and `timestamps` mappings:

```
    /// @notice Mapping of price feed keys to their current values
    /// @dev Values are stored with 18 decimal precision
    mapping(bytes32 => uint256) public values;

    /// @notice Mapping of price feed keys to their last update timestamps
    mapping(bytes32 => uint256) public timestamps;
```

If these values are meant to be unused, then consider removing them from the code.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

Consider removing the mappings if they will not be used or alternatively change the code so they are utilized.

Remediation Comment

SOLVED: The **Azos Finance team** solved the issue by removing the unused mappings.

Remediation Hash

https://github.com/AzosFinance/azos-core/commit/95e76b161a79717353b666a3e627cf725385236e

---