Analysed by Slither

Introduction

This document summarizes the findings of the security audit conducted on the smart contract **Foalca.sol**. Each finding is classified based on its potential impact, confidence level, and nature. Explanations are provided for each finding, along with space for the assessment of practical impact.

Resources

The analysis was performed using the **Slither** - Static Analyzer for Solidity and Vyper tool.

Tools

The analysis does not take into account external platforms custom-built to manage or protect contract functions.

Disclaimer

This security analysis is the first basic analysis that only evaluates the condition, optimization, and security of the contract code. It does not promise any future project results and makes no guarantees to anyone.

1. Findings Summary

Issue Category	Number of Findings	Impact
Uninitialized Local Variable	1	Medium
Missing Zero Address Check	1	Low
Variable Scope Issue	1	Low
Timestamp Dependence	2	Low
Use of Assembly	1	Informational
Costly Loop Operation	1	Informational
Dead Code	5	Informational
Solidity Version Usage	9	Informational
Naming Convention Issues	2	Informational
Constant Variables Suggestion	2	Optimization

2. Detailed Findings

2.1 Uninitialized Local Variable

- **Location:** `Foalca.executeProposal(uint256)` — `amount_scope_0`
- **Impact:** **Medium**
- **Description:**
  A local variable was declared without initialization. Uninitialized variables can result in unexpected behavior and should be properly set before use.

2.2 Missing Zero Address Check

- **Location:** `Foalca.setRewardsOperator(address _operator)`
- **Impact:** **Low**
- **Description:**
  There is no check ensuring that the provided address `_operator` is not the zero address (`address(0)`). This could allow setting invalid critical addresses.

2.3 Variable Scope Issue

- **Location:** `Foalca.executeProposal(uint256)` — `amount`
- **Impact:** **Low**
- **Description:**
  A variable (`amount`) might be used before proper declaration or initialization. This could introduce confusion or potential minor logic errors.

2.4 Timestamp Dependence

- **Location:**
    - `Foalca.distributeFromDevelopmentInternal()`
    - `Foalca.scheduledBurnInternal()`
- **Impact:** **Low**
- **Description:**
  The contract uses `block.timestamp` for time-based conditions. While common, this method can be slightly manipulated by miners/validators within acceptable margins.

2.5 Use of Inline Assembly

- **Location:** `ECDSA.tryRecover(bytes32, bytes)`
- **Impact:** **Informational**
- **Description:**
  Assembly is used inside OpenZeppelin's ECDSA library. While not a vulnerability per se, assembly code is harder to audit and prone to subtle mistakes if modified.

2.6 Costly Loop Operation

- **Location:** `Foalca.removeApprover(address)`
- **Impact:** **Informational**
- **Description:**
  A costly operation (`approverList.pop()`) is used inside a loop, potentially leading to increased gas consumption in transactions.

2.7 Dead Code

- **Location:**
    - `Context._contextSuffixLength()`
    - `Context._msgData()`
    - Several unused ECDSA functions
- **Impact:** **Informational**
- **Description:**
  Dead code (unused functions) slightly bloats the contract's size and can affect readability and maintenance but does not impact security.

2.8 Solidity Version Usage

- **Location:**
    - Contract and dependencies
- **Impact:** **Informational**
- **Description:**
  The pragma directive uses Solidity `^0.8.20` and compilation with `0.8.29`, which is relatively new. It is generally recommended to use well-tested versions (e.g., `0.8.16`) for production deployments.

2.9 Naming Convention Issues

- **Location:**
    - Variable: `_isExcludedFromFees`
    - Parameter: `_operator`
- **Impact:** **Informational**
- **Description:**
  Some variables do not adhere to the Solidity naming convention (`mixedCase`). This does not affect functionality but reduces code consistency and clarity.

2.10 Constant Variables Suggestion

- **Location:**
    - `burnPool`
    - `burnAddress`
- **Impact:** **Optimization**
- **Description:**
  These variables are immutable after deployment and should be marked as `constant`. This would optimize gas usage and clarify their purpose in the code.

3. General Conclusion

The analysis found no critical vulnerabilities. Most findings are related to code style, minor optimizations, or gas efficiency improvements.
The contract is currently secure for operation in its deployed state, with no immediate need for redeployment.