Self-Audit Report

This document provides an overview of the internal self-audit process undertaken by the PeoPay-Core development team. It details the methodologies, tools, and best practices we employed to ensure that our smart contracts are secure, maintainable, and free from known vulnerabilities before considering external audits or mainnet deployment.

Scope

The self-audit focused on the following contracts and functionalities:

- **PeoCoin (PEO):** ERC-20 token logic, minting/burning, access control.
- **Staking:** Staking logic, APY calculations, reward distribution, lock periods.
- **Governance:** Proposal creation, voting mechanisms, DCS-based weighting, proposal execution conditions.
- **DCS (Dynamic Contribution Scoring):** Score calculation, dependency on token balances and staking data, updateable weighting factors.
- **Conversion (Crypto-to-Mobile):** Request/confirm flow, authorized backendService logic, transferFrom checks, and event emissions.

Methodology

1. **Code Review & Static Analysis:**
    - The team performed a line-by-line review of each contract.
    - Focused on identifying potential reentrancy attacks, overflow/underflow scenarios, missing access controls, and incorrect validation logic.
    - Ensured adherence to recommended Solidity patterns (e.g., checks-effects-interactions pattern).
    - Used static analysis tools such as:
        - Slither for detecting common vulnerabilities.
        - Mythril for symbolic execution and vulnerability detection.
    - Fixed all reported warnings and re-verified after changes.
2. **Testing & Coverage:**
    - Implemented a comprehensive test suite with unit tests for each contract function.
    - Achieved near 100% line coverage and high branch coverage to ensure all paths are tested.
    - Added property-based testing using Echidna to confirm that certain invariants hold under random inputs.
    - Verified that no test scenario produces unexpected behavior or indicates a potential exploit path.
3. **Dependency Review:**
    - Confirmed that contracts rely on well-reviewed and widely used OpenZeppelin libraries.

    - Checked for any known vulnerabilities in external dependencies.
    - Ensured that contracts remain compatible with the latest stable Solidity compiler version and safe arithmetic practices.
4. **Role & Access Control Checks:**
    - Verified that only the owner can mint or update weights in DCS, or change governance parameters.
    - Confirmed that `onlyOwner` and `require` statements are correctly placed to prevent unauthorized actions.
5. **Event & Logging Consistency:**
    - Ensured all critical state changes emit events.
    - Verified that events correctly index critical parameters (e.g., `indexed` user addresses) for easier off-chain monitoring.
6. **Time & State Machine Validation:**
    - Checked that staking lock periods and governance voting periods are enforced correctly.
    - Verified that proposals cannot be executed before voting ends or re-executed once completed.
    - Confirmed that conversion requests require token approval and that conversions cannot be confirmed by unauthorized addresses.
7. **Documentation & Comments:**
    - Added Natspec comments to all public functions and contracts for clarity.
    - Improved inline comments to explain complex logic.
    - Ensured that the code is self-explanatory and maintainable.

Findings & Resolutions

- **Minor Logic Adjustments:**
  Detected some scenarios where time-based calculations could be more explicit, improving code clarity. Refined these calculations to reduce confusion.

- **Error Messages & Revert Reasons:**
  Standardized revert messages for consistency and clarity.

- **Test Coverage Gaps:**
  Initially found a few lines and branches untested. Added additional test cases to cover all scenarios and edge conditions.

No critical security vulnerabilities were identified during the self-audit. The minor issues found were addressed promptly, and the team re-ran static analysis and tests to confirm that they are resolved.

Next Steps

- **Additional Property-Based Testing:**
  Continuously refining property-based tests to explore more extreme edge

cases.

- **External Audit:**
  Although the self-audit and test coverage are thorough, we recommend a professional third-party security audit to validate the findings. Independent auditors can provide fresh perspectives and identify issues that may have been overlooked internally.
- **Bug Bounty Program:**
  Consider establishing a bug bounty program to incentivize community-led vulnerability reporting once the code is stable and deployed to a testnet or mainnet.

Conclusion

Our self-audit has increased confidence in the security and reliability of the PeoPay-Core contracts. The internal team's review, combined with automated analysis tools, extensive testing, and careful scrutiny of logic and roles, has made the code more secure and maintainable. Still, external audits and ongoing security measures remain essential for comprehensive protection.