WAGA Token Ecosystem - Smart Contract Audit Report

Executive Summary

The WAGA Token ecosystem consists of four contracts implementing an ERC20 token with a shop mechanism for ETH/USDC purchases and a comprehensive vesting system. The audit identified several critical vulnerabilities, multiple high-severity issues, and various optimization opportunities that must be addressed before deployment.

Audited Contracts

1. **WagaToken.sol** - ERC20 token with minting controls and 1 billion max supply
2. **TokenShop2.sol** - Token sale contract accepting ETH and USDC payments
3. **TokenVesting.sol** - Vesting and distribution contract for various stakeholder categories
4. **OracleLib.sol** - Library for Chainlink price feed validation

---

1. Security Vulnerabilities

CRITICAL Issues

C1: Reentrancy Vulnerability in `TokenShop2.buyWithEth()`

Severity: Critical
Location: TokenShop2.sol, lines 78-91
Description: State update occurs after external call, creating reentrancy vulnerability through `receive()` function.

```
// Vulnerable code
senderToEthSpent[msg.sender] += msg.value; // State update
// ... calculations ...
i_wagaToken.mint(msg.sender, tokensToMint); // External call
```

Impact: Attackers could drain tokens by reentering through a malicious contract.
Recommendation: Move all state updates after external calls or use ReentrancyGuard.

C2: Front-running in TokenShop2 Price Updates

Severity: Critical
Location: TokenShop2.sol, `setTokenPriceUsd()`
Description: Price updates can be front-run, allowing MEV bots to buy tokens at old prices before update.

Impact: Economic exploitation through sandwich attacks.
Recommendation: Implement time-delayed price updates or use commit-reveal scheme.

HIGH Severity Issues

H1: Centralization Risk - Single Point of Failure

Severity: High
Location: All contracts
Description: Admin can mint unlimited tokens, change prices instantly, and revoke vesting without timelock.

Impact: Complete protocol control by single address.
Recommendation: Implement multi-sig, timelock, or DAO governance.

H2: Integer Division Precision Loss

Severity: High
Location: TokenShop2.sol, `_usdToTokens()` and `_ethToUsd()`
Description: Division before multiplication causes precision loss.

```
// Precision loss example
return ((usdAmount * 1e18) / tokenPriceUsd);
```

Impact: Users may receive fewer tokens than expected.
Recommendation: Reorder operations: multiply first, then divide.

H3: Vesting Contract Token Custody Issue

Severity: High
Location: TokenVesting.sol, `revokeVesting()`
Description: Revoked tokens are sent to token contract address, effectively burning them.

```
// Tokens sent to wrong address
```

```
bool success = i_token.transfer(address(i_token), unreleased);
```

Impact: Permanent loss of revoked tokens.
Recommendation: Send to owner or treasury address.

H4: Missing Slippage Protection

Severity: High
Location: TokenShop2.sol
Description: No minimum token amount parameter, users vulnerable to price changes.

Impact: Users might receive far fewer tokens than expected.
Recommendation: Add minTokensExpected parameter.

MEDIUM Severity Issues

M1: Inadequate Oracle Staleness Check

Severity: Medium
Location: OracleLib.sol
Description: 1-hour staleness threshold too long for volatile crypto markets.

Impact: Outdated prices during high volatility.
Recommendation: Reduce to 5-15 minutes for ETH/USD.

M2: Unchecked Transfer Return Values

Severity: Medium
Location: Multiple locations
Description: Some USDC transfers don't follow safe transfer pattern.

Impact: Silent failures with non-compliant tokens.
Recommendation: Use SafeERC20 library.

M3: Timestamp Manipulation

Severity: Medium
Location: TokenVesting.sol
Description: Heavy reliance on block.timestamp for vesting calculations.

Impact: Miners can manipulate up to ~15 seconds.
Recommendation: Add buffer or use block numbers.

M4: Missing Event Emissions

Severity: Medium
Location: WagaToken.sol, `transferOwnership()`
Description: Critical ownership changes not logged.

Impact: Difficult to track governance changes.
Recommendation: Emit events for all permission changes.

---

2. Business Logic Issues

B1: Vesting Schedule Inflexibility

Issue: Cannot modify vesting schedules after creation (e.g., for employee departures).
Impact: Tokens locked unnecessarily for terminated employees.
Recommendation: Add modification functions with proper controls.

B2: No Emergency Pause Mechanism

Issue: No way to pause token sales or vesting in emergencies.
Impact: Cannot stop exploits quickly.
Recommendation: Implement OpenZeppelin's Pausable.

B3: Insufficient Category Validation

Issue: initializeCategory can be called multiple times if allocation is 0.
Impact: Potential misconfiguration.
Recommendation: Track initialization state separately.

B4: Missing Refund Mechanism

Issue: No way to refund ETH/USDC if token sale fails.
Impact: User funds trapped.
Recommendation: Add refund functionality.

B5: Vesting Cliff Calculation Issue

Issue: Vesting uses linear calculation from cliff, not true cliff behavior.
Impact: Tokens vest during cliff period.
Recommendation: Return 0 tokens until cliff reached.

---

3. Gas Optimization Opportunities

G1: Redundant Balance Checks

Location: TokenShop2.sol, withdraw functions
Issue: Checking balance <= 0 instead of == 0.
Gas Savings: ~50 gas per call
Fix:

```
if (balance == 0) revert TokenShop2__InsufficientBalance_withdrawEth();
```

G2: Storage Variable Caching

Location: TokenVesting.sol, multiple reads of vestingSchedules
Issue: Multiple SLOAD operations.
Gas Savings: ~2,100 gas per transaction
Fix:

```
VestingSchedule memory schedule = s_vestingSchedules[beneficiary];
```

G3: Unnecessary Comparisons

Location: Multiple contracts
Issue: Using `<= 0` for uint256 (always false for < 0).
Gas Savings: ~30 gas per comparison
Fix: Use `== 0` for uint256.

G4: Event Ordering

Location: TokenShop2.sol
Issue: Events emitted before state changes complete.
Gas Savings: Better error handling
Fix: Emit events after all operations.

---

4. Standards Compliance

S1: ERC20 Compliance

Status: Compliant
Note: Properly implements ERC20 with AccessControl.

S2: Missing EIP-2612 Permit

Status: Non-compliant
Impact: Users must approve in separate transaction.
Recommendation: Add permit functionality.

S3: Custom Errors Usage

Status: Partially Compliant
Issue: Not using custom errors consistently.
Recommendation: Replace all require statements.

---

5. Code Quality Issues

Q1: Inconsistent Naming Conventions

Issue: Mix of i_, s_ prefixes not applied uniformly.
Recommendation: Apply to all storage variables.

Q2: Missing NatSpec Documentation

Issue: Many functions lack @notice, @param, @return.
Recommendation: Complete documentation for all external/public functions.

Q3: Magic Numbers

Issue: Hardcoded values (1e18, 1e12, etc.).
Recommendation: Use named constants.

Q4: Redundant Comments

Issue: Comments like "// test" in production code.
Recommendation: Remove debug comments.

Q5: Unsafe Type Conversions

Location: OracleLib.sol
Issue: int256 to uint256 without negative check.
Recommendation: Add explicit validation.

---

6. Protocol-Specific Issues

P1: Token Economics Risk

Issue: No burn mechanism or deflationary features.
Impact: Only inflationary pressure.
Recommendation: Consider burn on transfer or buyback mechanism.

P2: Vesting Category Overlap

Issue: Beneficiary can only have one vesting schedule.
Impact: Cannot be in multiple categories.
Recommendation: Support multiple schedules per address.

P3: Oracle Dependency

Issue: Single oracle point of failure.
Impact: System halts if Chainlink fails.
Recommendation: Add fallback oracle or circuit breaker.

P4: USDC Depeg Risk

Issue: Assumes 1 USDC = 1 USD always.
Impact: Incorrect token pricing during depeg.
Recommendation: Use USDC/USD price feed.

Severity Summary

- **Critical:** 2 issues
- **High:** 4 issues
- **Medium:** 4 issues
- **Low:** Multiple optimization and quality issues

Attack Vectors Identified

1. **Reentrancy Attack** - Drain tokens through receive() callback
2. **Front-running Attack** - Exploit price updates
3. **Precision Attack** - Exploit rounding errors for profit
4. **Griefing Attack** - Block vesting releases
5. **MEV Attack** - Sandwich trades around price updates

Recommendations Priority

1. **Immediate (Before Deployment):**
   - Fix reentrancy vulnerability
   - Implement slippage protection
   - Fix vesting contract token burning
   - Add access control timelocks
2. **Short-term (Within 1 week):**
   - Reduce oracle staleness threshold
   - Add emergency pause mechanism
   - Fix precision loss issues
   - Complete event emissions
3. **Long-term (Within 1 month):**
   - Implement multi-sig governance
   - Add permit functionality
   - Enhance vesting flexibility
   - Add comprehensive monitoring

Testing Recommendations

Security Testing:

```
 // Test reentrancy
contract AttackContract {
    TokenShop2 shop;
    function attack() external payable {
        shop.buyWithEth{value: msg.value}();
    }
    receive() external payable {
        if (address(shop).balance > 0) {
            shop.buyWithEth{value: 0}();
        }
    }
}
```

1.
2. **Edge Case Testing:**
   - Zero amount purchases
   - Maximum uint256 values
   - Vesting edge cases (0 duration, max duration)
   - Price manipulation scenarios
3. **Integration Testing:**
   - Full lifecycle token purchase → vesting → release
   - Multi-user vesting scenarios
   - Oracle failure scenarios

Deployment Checklist

- Fix all critical/high issues
- Add comprehensive test suite (>95% coverage)
- Deploy behind proxy for upgradeability
- Set up multi-sig for admin functions
- Configure monitoring and alerts
- Conduct external audit
- Set up bug bounty program

Conclusion

The WAGA Token ecosystem shows a well-structured design but contains several critical vulnerabilities that must be addressed before deployment. The centralization risks and potential for economic exploitation require immediate attention. The vesting system, while functional, needs enhancement for production use.

Recommendation: Don't deploy to mainnet until all critical and high-severity issues are resolved and a timelock/multi-sig governance system is implemented.

Audit conducted by: Antenhe Z Tena
Date: Tuesday June 17th 2025
Solidity Version: 0.8.20
Dependencies: OpenZeppelin 4.x, Chainlink Aggregator V3