ODDSHOT V1 AUDIT REPORT

Smart Contract Security Assessment

Executive Summary

Audit Target: ODDToken.sol

Client: ODDSHOT (oddshot.tech)

Audit Period: August 12-14, 2025

Auditor: INVCBULL AUDIT GROUP (IAG)

Auditors Assigned: INVCBULL and YEANDAMEN

Report prepared by:
Okiki Omisande
Blockchain Security Researcher
Twitter: @okiki_omisande
Telegram: @Invcbull

Table of Contents

Introduction

This document includes the results of the security audit for ODDSHOT's smart contract code as found in the section titled 'Scope'. The security audit was performed by the INVCBULL AUDIT GROUP (IAG) security team from August 12th, 2025 to August 14th, 2025.

The purpose of this audit is to review the source code of certain ODDSHOT Solidity contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While IAG's review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Audit Overview

This audit focuses exclusively on the **ODDToken.sol** contract, which implements a BEP-20 token with governance features, daily inflation mechanisms, and role-based access controls for the ODDSHOT ecosystem.

Key Features Audited:

- ERC20 token implementation with inflation mechanics
- Multi-role access control system with timelock mechanisms
- Daily minting functionality for staking rewards
- Governance proposal and execution system

Audit Statistics:

- **Lines of Code:** 304
- **Contracts Audited:** 1
- **Critical Issues:** 0
- **High Severity Issues:** 2
- **Medium Severity Issues:** 2
- **Low Severity Issues:** 2
- **Informational Issues:** 2
- **Total Findings:** 8

Scope

In Scope:

- `ODDToken.sol` - Main token contract with governance and inflation features

Out of Scope:

- External dependencies (OpenZeppelin contracts)
- Integration contracts (StakingContract, VaultContract)
- Deployment scripts
- Frontend/backend implementation

Risk Classification

Severity	Impact	Likelihood	Description
Critical	High	High	Immediate threat to funds or core functionality
High	High	Medium	Significant security risk requiring immediate attention
Medium	Medium	Medium	Notable security concern with potential impact
Low	Low	Low	Minor issues with limited impact
Informational	-	-	Code quality and best practice recommendations

Findings Summary

ID	Title	Severity	Status
H-01	Governance Deadlock in Admin Role Revocation	High	Resolved
H-02	Role Change Proposal Bypass Vulnerability	High	Resolved
M-01	Zero Inflation Proposal Execution Denial of Service	Medium	Resolved
M-02	Block Time Dependency Risk in Timelock Mechanisms	Medium	Resolved
L-01	Daily Amount Minted in Mint Daily Inflation Will Always Be 0 When Annual Inflation is Set to a Value Less Than 365	Low	Acknowledged
L-02	Contract Version Inconsistency	Low	Resolved
I-01	Input Validation Check Should Be Moved to Function Start	Informational	Resolved
I-02	L-01 Partial Mitigation	Informational	Acknowledged

Risk Distribution:

- **High:** 2 findings
- **Medium:** 2 findings
- **Low:** 2 findings
- **Informational:** 2 findings
- **Total:** 8 findings

Detailed Findings

H-01: Governance Deadlock in Admin Role Revocation

Severity: High

Description

The `revokeDeployerAdminRole()` function creates a permanent governance deadlock when the contract itself becomes the sole `DEFAULT_ADMIN_ROLE` holder.

Root Cause

The function checks that `address(this)` has `DEFAULT_ADMIN_ROLE` before allowing revocation, but smart contracts cannot call their own external functions that require role-based access control.

```
function revokeDeployerAdminRole() external onlyRole(DEFAULT_ADMIN_ROLE) {
    // Only allow revoking if there's at least one other admin
    require(hasRole(DEFAULT_ADMIN_ROLE, address(this)), "Cannot revoke last admin");
    renounceRole(DEFAULT_ADMIN_ROLE, msg.sender);
}
```

Impact

- **Permanent loss of governance functionality**
- Critical admin functions become permanently inaccessible:
  - `proposeGrantRole()`
  - `proposeRevokeRole()`
  - `executeRoleChange()`
  - `cancelRoleChange()`

Proof of Concept

1. Deploy contract (deployer gets `DEFAULT_ADMIN_ROLE`)
2. Grant `DEFAULT_ADMIN_ROLE` to contract itself
3. Call `revokeDeployerAdminRole()` - succeeds
4. Now only the contract has admin role
5. All admin functions become permanently inaccessible

Recommendation

Simply remove the problematic check and ensure proper documentation:

```
function revokeDeployerAdminRole() external onlyRole(DEFAULT_ADMIN_ROLE) {
    // Remove the problematic require statement
    renounceRole(DEFAULT_ADMIN_ROLE, msg.sender);
}
```

Important: Before calling this function, ensure at least one other trusted address has `DEFAULT_ADMIN_ROLE`. This should be documented clearly and enforced through deployment procedures rather than code constraints.

Remediation

Status: ACKNOWLEDGED AND FIXED

The ODDSHOT team has acknowledged this issue and implemented the a fix.

H-02: Role Change Proposal Bypass Vulnerability

Severity: High

Description

A governance bypass vulnerability allows any privileged admin to propose a role change and then execute a completely different role change from the original proposal. This completely undermines the timelock security model and enables privilege escalation attacks.

Root Cause

The `executeRoleChange()` function accepts `role` and `grant` parameters from the caller without validating them against the original proposal. Only the target `account` address is bound to the proposal, allowing attackers to substitute different roles and operations during execution.

Affected Code

```
 // Lines 267-294: executeRoleChange function
function executeRoleChange(bytes32 proposalId, bytes32 role, bool grant) external whenNotPaused onlyRole(DEFAULT_ADMIN_ROLE) {
    require(proposedRoleChanges[proposalId] != 0, "Proposal does not exist");
    require(!executedRoleChanges[proposalId], "Proposal already executed");
    require(block.timestamp >= proposedRoleChangeTimestamps[proposalId], "Timelock not expired");

    address account = address(uint160(proposedRoleChanges[proposalId]));

    // VULNERABILITY: No validation of role and grant against original proposal
    if (grant) {
        super.grantRole(role, account); // Uses caller-provided role!
        if (role == DEFAULT_ADMIN_ROLE) {
            adminCount++;
        }
    } else {
        super.revokeRole(role, account); // Uses caller-provided role!
        if (role == DEFAULT_ADMIN_ROLE) {
            adminCount--;
        }
    }

    executedRoleChanges[proposalId] = true;
    emit RoleChangeExecuted(proposalId, role, account, grant);
}
```

Impact

- **Complete Governance Compromise:** Timelock security model is bypassed
- **Privilege Escalation:** Attackers can grant themselves admin privileges
- **Access Control Failure:** Unauthorized access to critical functions
- **Operational Risk:** Unexpected role revocations causing DoS
- **Trust Erosion:** Governance system becomes unreliable

Attack Scenarios

Attack Vector 1: Privilege Escalation

1. Admin proposes to grant low-privilege `MINTER_ROLE` to address 0x123
2. Wait for timelock period (3 days)
3. Execute with different role: `executeRoleChange(proposalId, DEFAULT_ADMIN_ROLE, true)`
4. Result: Address 0x123 receives `DEFAULT_ADMIN_ROLE` instead of `MINTER_ROLE`

Attack Vector 2: Operation Inversion

1. Admin proposes to grant `MINTER_ROLE` to address 0x456
2. Execute as revoke: `executeRoleChange(proposalId, MINTER_ROLE, false)`
3. Result: Address 0x456 loses `MINTER_ROLE` instead of gaining it

Attack Vector 3: Role Substitution

1. Admin proposes to revoke `MINTER_ROLE` from address 0x789
2. Execute with different role: `executeRoleChange(proposalId, GOVERNANCE_ROLE, false)`
3. Result: Address 0x789 loses `GOVERNANCE_ROLE` instead of `MINTER_ROLE`

Recommendation

Implement complete proposal data storage and validation:

```
// New struct to store complete proposal information
struct RoleChangeProposal {
    address account;
    bytes32 role;
    bool grant;
    uint256 executeAfterTimestamp;
    bool executed;
}

// Replace current mappings with comprehensive storage
mapping(bytes32 => RoleChangeProposal) public roleChangeProposals;

// Fixed execution function - removes role and grant parameters
function executeRoleChange(bytes32 proposalId) external whenNotPaused onlyRole(DEFAULT_ADMIN_ROLE) {
    RoleChangeProposal storage proposal = roleChangeProposals[proposalId];

    require(proposal.account != address(0), "Proposal does not exist");
    require(!proposal.executed, "Proposal already executed");
    require(block.timestamp >= proposal.executeAfterTimestamp, "Timelock not expired");

    // Execute using stored proposal data - no caller parameters
    if (proposal.grant) {
        super.grantRole(proposal.role, proposal.account);
        if (proposal.role == DEFAULT_ADMIN_ROLE) {
            adminCount++;
        }
    } else {
        super.revokeRole(proposal.role, proposal.account);
        if (proposal.role == DEFAULT_ADMIN_ROLE) {
            adminCount--;
        }
    }

    proposal.executed = true;
    emit RoleChangeExecuted(proposalId, proposal.role, proposal.account, proposal.grant);
}
```

Remediation

Status: ACKNOWLEDGED AND FIXED

The ODDSHOT team has acknowledged this issue and implemented a fix.

M-01: Zero Inflation Proposal Execution Denial of Service

Severity: Medium

Description

The contract uses hardcoded block counts to represent time delays, creating a critical dependency on BSC's current block time assumptions. This design is vulnerable to blockchain upgrades, network congestion, and protocol changes that could significantly alter the intended timing of governance operations.

Root Cause

The timelock delays are calculated based on a fixed assumption of 0.75 seconds per block:

```
// Lines 24-27
uint256 public constant INFLATION_CHANGE_DELAY_BLOCKS = 230400; // Assumes 48 hours
uint256 public constant ROLE_CHANGE_DELAY_BLOCKS = 345600; // Assumes 72 hours

// Comment states: "BSC produces ~1.33 blocks per second (0.75 seconds per block)"
```

Vulnerability Analysis

Current Assumption: 230,400 blocks = 48 hours (0.75s per block) **Risk Scenarios:**

1. **Network Congestion:** Block times increase to 2+ seconds
2. **Protocol Upgrades:** BSC hardforks modify block production (e.g., Planck fork)
3. **Validator Issues:** Network stress extends block intervals

Impact Scenarios

Block Time	230,400 Blocks Duration	Impact
0.75s (current)	48 hours	As intended
1.5s (2x slower)	96 hours	2x longer delays
2.0s (congestion)	128 hours	2.67x longer delays
3.0s (severe issues)	192 hours	4x longer delays

Real-World Impact

- **Governance Paralysis:** Critical updates delayed by days instead of hours
- **Emergency Response:** Security patches take 4x longer during network stress
- **User Experience:** Unpredictable timelock periods confuse stakeholders

Proof of Concept

```
// Current: 48-hour delay assumption
uint256 expectedHours = 48;
uint256 blocks = 230400;
uint256 assumedBlockTime = 0.75 seconds;

// Reality during network congestion
uint256 actualBlockTime = 2.0 seconds;
uint256 actualHours = (blocks * actualBlockTime) / 3600; // = 128 hours
uint256 delay_multiplier = actualHours / expectedHours; // = 2.67x longer
```

Historical Context

- **Planck Fork:** BSC has already modified block production parameters
- **Future Upgrades:** More changes are likely as the network evolves
- **Cross-Chain Deployments:** If deployed on other EVM chains, timing assumptions break entirely

Recommendation

Replace block-based delays with timestamp-based delays:

```
 // Replace block-based constants with time-based ones
uint256 public constant INFLATION_CHANGE_DELAY_TIME = 48 hours;
uint256 public constant ROLE_CHANGE_DELAY_TIME = 72 hours;

// Add new mapping for timestamp delays
mapping(bytes32 => uint256) public proposalExecuteAfterTime;

// Update proposeInflationChange
function proposeInflationChange(uint256 _newAnnualInflation) external onlyRole(GOVERNANCE_ROLE) {
    require(_newAnnualInflation <= MAX_ANNUAL_INFLATION, "Annual inflation cannot exceed 10 billion tokens");
    require(_newAnnualInflation != annualInflationAmount, "New inflation rate same as current");

    bytes32 proposalId = keccak256(abi.encodePacked(_newAnnualInflation, block.number, msg.sender));
    require(proposedInflationChanges[proposalId] == 0, "Proposal already exists");

    proposedInflationChanges[proposalId] = _newAnnualInflation;
    proposalExecuteAfterTime[proposalId] = block.timestamp + INFLATION_CHANGE_DELAY_TIME; // Use timestamp

    emit InflationChangeProposed(proposalId, _newAnnualInflation, block.timestamp + INFLATION_CHANGE_DELAY_TIME);
}

// Update executeInflationChange
function executeInflationChange(bytes32 proposalId) external onlyRole(GOVERNANCE_ROLE) {
    require(proposedInflationChanges[proposalId] != 0, "Proposal does not exist");
    require(!executedInflationChanges[proposalId], "Proposal already executed");
    require(block.timestamp >= proposalExecuteAfterTime[proposalId], "Timelock not expired"); // Use timestamp

    uint256 newInflation = proposedInflationChanges[proposalId];
    annualInflationAmount = newInflation;
    executedInflationChanges[proposalId] = true;

    emit InflationChangeExecuted(proposalId, newInflation);
}
```

Remediation

Status: ACKNOWLEDGED AND FIXED

The ODDSHOT team has acknowledged this issue and implemented the recommended fix.

M-02: Block Time Dependency Risk in Timelock Mechanisms

Severity: Medium

Description

The proposal system uses `0` as both a sentinel value for "proposal doesn't exist" and a valid inflation value, creating unexecutable proposals when zero inflation is proposed.

Root Cause

The storage and validation logic conflicts:

- Line 98: `require(proposedInflationChanges[proposalId] == 0, "Proposal already exists")`
- Line 100: `proposedInflationChanges[proposalId] = _newAnnualInflation` (can be 0)
- Line 111: `require(proposedInflationChanges[proposalId] != 0, "Proposal does not exist")`

Impact

- **Denial of Service** for zero inflation proposals
- **Governance disruption** for legitimate economic decisions
- **Gas waste** on unexecutable proposals

Proof of Concept

1. Call `proposeInflationChange(0)` - succeeds, stores 0
2. Wait for timelock period

3. Call `executeInflationChange(proposalId)` - fails with "Proposal does not exist"
4. Proposal becomes permanently unexecutable

Recommendation

Add a simple boolean mapping to track proposal existence:

```
// Add new state variable
mapping(bytes32 => bool) public proposalExists;

// Update proposeInflationChange
function proposeInflationChange(uint256 _newAnnualInflation) external onlyRole(GOVERNANCE_ROLE) {
    require(_newAnnualInflation <= MAX_ANNUAL_INFLATION, "Annual inflation cannot exceed 10 billion tokens");
    require(_newAnnualInflation != annualInflationAmount, "New inflation rate same as current");

    bytes32 proposalId = keccak256(abi.encodePacked(_newAnnualInflation, block.number, msg.sender));
    require(!proposalExists[proposalId], "Proposal already exists");

    proposedInflationChanges[proposalId] = _newAnnualInflation;
    proposedInflationBlockNumbers[proposalId] = block.number + INFLATION_CHANGE_DELAY_BLOCKS;
    proposalExists[proposalId] = true; // Add this line

    emit InflationChangeProposed(proposalId, _newAnnualInflation, block.number + INFLATION_CHANGE_DELAY_BLOCKS);
}

// Update executeInflationChange
function executeInflationChange(bytes32 proposalId) external onlyRole(GOVERNANCE_ROLE) {
    require(proposalExists[proposalId], "Proposal does not exist"); // Change this line
    require(!executedInflationChanges[proposalId], "Proposal already executed");
    require(block.number >= proposedInflationBlockNumbers[proposalId], "Timelock not expired");

    uint256 newInflation = proposedInflationChanges[proposalId];
    annualInflationAmount = newInflation;
    executedInflationChanges[proposalId] = true;

    emit InflationChangeExecuted(proposalId, newInflation);
}
```

Remediation

Status: ACKNOWLEDGED AND FIXED

The ODDSHOT team has acknowledged this issue and implemented the recommended fix.

L-01: Daily Amount Minted in Mint Daily Inflation Will Always Be 0 When Annual Inflation is Set to a Value Less Than 365

Severity: Low

Description

The `mintDailyInflation()` function uses integer division to calculate daily mint amounts, causing the function to revert when `annualInflationAmount` is less than 365, as the daily amount becomes 0.

Root Cause

Integer division in Solidity truncates fractional results to zero:

```
function mintDailyInflation() external onlyRole(MINTER_ROLE) {
    // ...
    uint256 dailyAmount = annualInflationAmount / 365; // Integer division
    require(dailyAmount > 0, "Daily mint amount too small"); // Fails when result is 0
    // ...
}
```

Impact Scenarios

Annual Inflation	Daily Amount	Status	Lost Tokens/Year
100	0	❌ DoS	N/A (function reverts)
364	0	❌ DoS	N/A (function reverts)
365	1	✅ Works	0
1000	2	✅ Works	270 (1000 - 2×365)
10000	27	✅ Works	145 (10000 - 27×365)

Real-World Impact

- **Function DoS:** Annual inflation below 365 makes `mintDailyInflation()` permanently unusable
- **Precision Loss:** Cumulative loss of intended inflation over time
- **Governance Mismatch:** Users can propose values (0-364) that create unusable proposals
- **Economic Inaccuracy:** Actual inflation differs from intended inflation

Proof of Concept

```
// Scenario 1: Complete DoS
annualInflationAmount = 100;
uint256 dailyAmount = 100 / 365; // = 0
// mintDailyInflation() reverts with "Daily mint amount too small"

// Scenario 2: Precision loss
annualInflationAmount = 1000;
uint256 dailyAmount = 1000 / 365; // = 2
uint256 actualYearlyInflation = 2 * 365; // = 730
uint256 lostTokens = 1000 - 730; // = 270 tokens lost per year
```

Recommendation

Use minimum thresholds and clear validation:

```
uint256 public constant MIN_ANNUAL_INFLATION = 365; // Minimum 1 token per day

function proposeInflationChange(uint256 _newAnnualInflation) external onlyRole(GOVERNANCE_ROLE) {
    require(_newAnnualInflation <= MAX_ANNUAL_INFLATION, "Annual inflation cannot exceed 10 billion tokens");
    require(_newAnnualInflation == 0 || _newAnnualInflation >= MIN_ANNUAL_INFLATION,
           "Annual inflation must be 0 or at least 365");
    require(_newAnnualInflation != annualInflationAmount, "New inflation rate same as current");
    // ... rest of function
}
```

Remediation

Status: ACKNOWLEDGED

The ODDSHOT team has acknowledged this issue.

L-02: Contract Version Inconsistency

Severity: Low

Description

During the mitigation review, it was discovered that two different versions of the ODDToken.sol contract exist in the codebase - one in the root directory and another in the src/ directory. This inconsistency could lead to confusion about which version is the current, production-ready implementation.

Root Cause

The codebase contains duplicate contract files with potentially different implementations:

- Root directory: `ODDToken.sol`
- Source directory: `src/ODDToken.sol`

Impact

- **Deployment Confusion:** Risk of deploying outdated contract version
- **Audit Scope Issues:** Security assessment may not cover the intended production contract
- **Maintenance Complexity:** Multiple versions require synchronization and version control
- **Security Risk:** Outdated versions may contain known vulnerabilities

Recommendation

1. **Remove Duplicate Files:** Eliminate the outdated version in the src/ directory
2. **Version Control:** Implement clear versioning and single source of truth
3. **Documentation:** Clearly indicate which contract file is the production version
4. **Deployment Scripts:** Ensure deployment scripts reference the correct contract file

Remediation

Status: ACKNOWLEDGED AND FIXED

The ODDSHOT team has acknowledged this issue and committed to resolving the version inconsistency.

I-01: Input Validation Check Should Be Moved to Function Start

Severity: Informational

Description

In the `setContracts()` function, the input validation check is performed after processing the contract addresses, leading to unnecessary gas consumption when the validation fails and suboptimal error handling flow.

Root Cause

The validation logic is placed at the end of the function (line 86) instead of at the beginning:

```
function setContracts(address _stakingContract, address _vaultContract) external onlyRole(STAKING_MANAGER_ROLE) {
    if (_stakingContract != address(0)) {
        require(_stakingContract != stakingContract, "Staking contract already set to this address");
        stakingContract = _stakingContract;
        emit StakingContractSet(_stakingContract);
    }

    if (_vaultContract != address(0)) {
        require(_vaultContract != vaultContract, "Vault contract already set to this address");
        vaultContract = _vaultContract;
        emit VaultContractSet(_vaultContract);
    }
    //info- this check should be moved up
    require(_stakingContract != address(0) || _vaultContract != address(0), "At least one contract must be set");
}
```

Impact

- **Gas Inefficiency:** Unnecessary processing before validation failure
- **Code Quality:** Violates fail-fast principle for input validation

Recommendation

Move the validation check to the beginning of the function:

```
function setContracts(address _stakingContract, address _vaultContract) external onlyRole(STAKING_MANAGER_ROLE) {
    // Move validation to the top for early failure
    require(_stakingContract != address(0) || _vaultContract != address(0), "At least one contract must be set");

    if (_stakingContract != address(0)) {
        require(_stakingContract != stakingContract, "Staking contract already set to this address");
        stakingContract = _stakingContract;
        emit StakingContractSet(_stakingContract);
    }

    if (_vaultContract != address(0)) {
        require(_vaultContract != vaultContract, "Vault contract already set to this address");
        vaultContract = _vaultContract;
        emit VaultContractSet(_vaultContract);
    }
}
```

Remediation

Status: ACKNOWLEDGED AND FIXED

The ODDSHOT team has acknowledged this issue and implemented the recommended fix.

I-02: L-01 Partial Mitigation

Severity: Informational

Description

During the mitigation review, it was discovered that the integer division issue in the `mintDailyInflation()` function (L-01) was only partially addressed. While the team acknowledged the issue, the core mathematical behavior remains unchanged.

Root Cause

The original issue involved integer division causing daily mint amounts to be 0 when annual inflation is less than 365. The mitigation focused on documentation rather than structural changes to the calculation logic.

Impact

- **Continued Precision Loss:** The mathematical precision issue persists
- **Documentation Gap:** Users may still be unaware of the minimum inflation requirements
- **Economic Inaccuracy:** Actual inflation continues to differ from intended inflation for values below 365

Recommendation

Implement a more robust solution that addresses the root cause:

```
// Add minimum validation in the proposal function
function proposeInflationChange(uint256 _newAnnualInflation) external onlyRole(GOVERNANCE_ROLE) {
    require(_newAnnualInflation <= MAX_ANNUAL_INFLATION, "Annual inflation cannot exceed 10 billion tokens");
    require(_newAnnualInflation == 0 || _newAnnualInflation >= 365, "Annual inflation must be 0 or at least 365");
    require(_newAnnualInflation != annualInflationAmount, "New inflation rate same as current");
    // ... rest of function
}

// Alternative: Use higher precision calculations
function mintDailyInflation() external onlyRole(MINTER_ROLE) {
    require(annualInflationAmount > 0, "No inflation configured");

    // Use higher precision: multiply by 1000 then divide by 365000
    uint256 dailyAmount = (annualInflationAmount * 1000) / 365000;
    require(dailyAmount > 0, "Daily mint amount too small");

    // ... rest of function
}
```

Remediation

Status: ACKNOWLEDGED

The ODDSHOT team has acknowledged this issue and implemented documentation improvements.

Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. INVCBULL AUDIT GROUP (IAG), of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, IAG provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

The scope of this report and review is limited to a review of only the code presented by the ODDSHOT team and only the source code IAG notes as being within the scope of IAG's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit.

Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than IAG. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that IAG is not responsible for the content or operation of such websites, and that IAG shall have no liability to you or any other person or entity for the use of third party websites. IAG assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by IAG.
