# SMART CONTRACTS REVIEW

zokyo

November 19th 2024 | v. 1.0

# Security Audit Score

## PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

# # ZOKYO AUDIT SCORING DEVVE

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

# SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 0 Medium issue: 0 points deducted
- 1 Low issue: 1 acknowledged = - 3 points deducted
- 2 Informational issues: 2 acknowledged = 0 points deducted

Thus, 100- 3 = 97

# TECHNICAL SUMMARY

This document outlines the overall security of the Devve smart contract evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Devve smart contract codebase for quality, security, and correctness.

## Contract Status


There were 0 critical issues found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Devve team put in place a bug bounty program to encourage further active analysis of the smart contract.

# Table of Contents

# AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Devve repository:

Repository: code provided as an archive
**Contracts under the scope:**

- Fias2.sol - SHA fd12c0750fb26b30e787c675fa8d10e0d3a9ccf8
- Fias2_V2.sol - SHA 557ebee12ba3e9472be8ad47d3c0b9cae8b2a862

**During the audit, Zokyo Security ensured that the contract:**

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Devve smart contract. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01 Due diligence in assessing the overall code quality of the codebase.

02 Cross-comparison with other, similar smart contract by industry leaders.

03 Thorough manual review of the codebase line by line.

# Executive Summary

The Fias contract is an ERC20 token contract based on OpenZeppelin's upgradeable implementation. It includes predefined minters with specific token limits, and a custom multiTransfer function for batch transfers. Key Components:

Constants:

- GLOBAL_LIMIT: Maximum token supply (350 million tokens).
- FOREVVER_MINTER: Predefined address with minting rights for 175 million tokens.
- LITCRAFT_MINTER: Another predefined address with minting rights for 175 million tokens.
- CONTRACT_ADMIN: Address designated as the contract administrator.

Initialization: The initialize function initializes the contract as an ERC20 token named "Fias" with the symbol "FIAS". During initialization:

- 175 million tokens are minted to LITCRAFT_MINTER.
- 175 million tokens are minted to FOREVVER_MINTER.

Token Transfer Functionality:

- Implements a custom multiTransfer function for batch transfers:
- Validates that the lengths of recipients and amounts arrays match.
- Ensures all transfer amounts are positive.
- Checks that the sender has sufficient balance for the total transfer amount.
- Executes transfers to the specified recipients by calling transferFrom.

# STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Devve team and the Devve team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

**Critical**

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

**High**

The issue affects the ability of the contract to compile or operate in a significant way.

**Medium**

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

**Low**

The issue has minimal impact on the contract’s ability to operate.

**Informational**

The issue has no impact on the contract’s ability to operate.

# COMPLETE ANALYSIS

## FINDINGS SUMMARY

| # | Title | Risk | Status |
|---|---|---|---|
| 1 | Addresses Should Not Be Hardcoded | Low | Acknowledged |
| 2 | Unused Constants in Fias Token Contract | Informational | Acknowledged |
| 3 | Lack of Event Emissions for Critical State Changes in Fias Token Contract | Informational | Acknowledged |

**Addresses Should Not Be Hardcoded**

In the Fias2.sol addresses such as FOREVERR_MINTER and LITCRAFT_MINTER have been hardcoded , it is possible that on different chains the addresses corresponding to these roles are different and hence instead of hardcoding the addresses they should be assigned in the initializer.

**Recommendation:**
It is recommended that addresses should be assigned in the initializer instead of being hardcoded.

Client comment: no impact because we are only using this contract on ETH and not other chains with other Forevver/LitCraft/Admin addresses.

**Unused Constants in Fias Token Contract**

Description:

The Fias token contract defines two constants, GLOBAL_LIMIT and CONTRACT_ADMIN, which are not utilized within the contract's implementation. The GLOBAL_LIMIT is set to 350,000,000, presumably intended to represent the maximum total supply of tokens. The CONTRACT_ADMIN is defined as an address, likely meant for administrative functions. However, neither of these constants is referenced in any of the contract's functions or logic.

**Recommendation:**

1. If these constants are intended for future use:
   a. Document their intended purpose in comments above each constant.
   b. Implement the functionality that utilizes these constants, such as enforcing the global token limit or adding admin-only functions.
2. If these constants are no longer needed:
   a. Remove the unused constants to improve code clarity and slightly reduce deployment gas costs.
3. For the GLOBAL_LIMIT:
   a. If it's meant to represent the maximum token supply, consider implementing a check in the initialize function to ensure the total minted amount doesn't exceed this limit.
4. For the CONTRACT_ADMIN:
   a. If administrative functions are planned, implement them with appropriate access control using this address.
   b. Consider using OpenZeppelin's Ownable or AccessControl contracts for more robust admin functionality.

Client comment: no impact, the admin functions that previously used those constants were removed and they only remain for clarity/readability.

**Lack of Event Emissions for Critical State Changes in Fias Token Contract**

Description:

The Fias token contract, including its V2 upgrade, fails to emit events for significant state changes, particularly during token minting in the initialize function and token burning in the burn function.

**Recommendation:**

1. For the initialize function:
   a. Define and emit a custom event for the initial token minting. For example:

```
event InitialMint(address indexed to, uint256 amount);

function initialize() public initializer {
    __ERC20_init("Fias", "FIAS");
    _mint(LITCRAFT_MINTER, LITCRAFT_LIMIT*(10**18));

    emit InitialMint(LITCRAFT_MINTER, LITCRAFT_LIMIT*(10**18));
    _mint(FOREVVER_MINTER, FOREVVER_LIMIT*(10**18));
    emit InitialMint(FOREVVER_MINTER, FOREVVER_LIMIT*(10**18));
}
```

2. For the burn function:
   b. Utilize OpenZeppelin's built-in _burn function, which already emits a Transfer event. If additional information is needed, consider adding a custom event:

```
event TokensBurned(address indexed burner, uint256 amount);

function burn(uint256 amount) public {
    _burn(_msgSender(), amount);
    emit TokensBurned(_msgSender(), amount);
}
```

Client comment: this would be nice to add if we change anything and it's odd that it isn't part of the default OpenZeppelin functions, but there is no security risk to this.

| | Fias2.sol<br>Fias2_V2.sol |
|---|---|
| Re-entrancy | Pass |
| Access Management Hierarchy | Pass |
| Arithmetic Over/Under Flows | Pass |
| Unexpected Ether | Pass |
| Delegatecall | Pass |
| Default Public Visibility | Pass |
| Hidden Malicious Code | Pass |
| Entropy Illusion (Lack of Randomness) | Pass |
| External Contract Referencing | Pass |
| Short Address/ Parameter Attack | Pass |
| Unchecked CALL<br>Return Values | Pass |
| Race Conditions / Front Running | Pass |
| General Denial Of Service (DOS) | Pass |
| Uninitialized Storage Pointers | Pass |
| Floating Points and Precision | Pass |
| Tx.Origin Authentication | Pass |
| Signatures Replay | Pass |
| Pool Asset Security (backdoors in the underlying ERC-20) | Pass |

We are grateful for the opportunity to work with the Devve team.

**The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.**

Zokyo Security recommends the Devve team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo