# FYEO

## Security Code Review of Natgold Contract

Natgold

January 2026
Version 1.0

Presented by:

FYEO Inc.

PO Box 147044
Lakewood CO 80214
United States

Security Level
Strictly Confidential

# TABLE OF CONTENTS

# Executive Summary

## Overview

Natgold engaged FYEO Inc. to perform a Security Code Review of Natgold.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on January 05 - January 08, 2026, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.

- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.

- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

## KEY FINDINGS

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-NAT-01 – Admin Can Accidentally Renounce Critical Roles

- FYEO-NAT-02 – Integer Division Precision Loss in Token Distribution

- FYEO-NAT-03 – Missing Safe UPGRADER_ROLE Validation in ProposeUpgrade

- FYEO-NAT-04 – Missing Storage Gaps in Upgradeable Contracts

- FYEO-NAT-05 – Missing Validation in Queue Project Removal

- FYEO-NAT-06 – No Duplicate Project Name Prevention

- FYEO-NAT-07 – Role Revocation Behavior Differs from OpenZeppelin

Based on our review process, we conclude that the reviewed code implements the documented functionality.

## Scope and Rules of Engagement

The FYEO Review Team performed a Security Code Review of Natgold. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/fpco/natgold with the commit hash 1093ca36b547a6b28a8afedcabc48a83ef368440.

Remediations were submitted with the commit hash a2332acbee0edfb5218da84a228aac6b503dbb2c.

| Files included in the code review |
|---|
| ```
natgold/
├── contract/
│   ├── script/
│   │   ├── helpers/
│   │   │   └── proposeSafeTx.js
│   │   ├── Deploy.s.sol
│   │   ├── ProposeUpgrade.s.sol
│   │   └── Upgrade.s.sol
│   └── src/
│       ├── NatGoldQueueOrchestrator.sol
│       └── NatGoldToken.sol
``` |

Table 1: Scope

# Technical Analyses and Findings

During the Security Code Review of Natgold, we discovered:

- 7 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

Figure 1: Findings by Severity

## Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

| Finding # | Severity | Description | Status |
|---|---|---|---|
| FYEO-NAT-01 | Informational | Admin Can Accidentally Renounce Critical Roles | Remediated |
| FYEO-NAT-02 | Informational | Integer Division Precision Loss in Token Distribution | Remediated |
| FYEO-NAT-03 | Informational | Missing Safe UPGRADER_ROLE Validation in ProposeUpgrade | Remediated |
| FYEO-NAT-04 | Informational | Missing Storage Gaps in Upgradeable Contracts | Remediated |
| FYEO-NAT-05 | Informational | Missing Validation in Queue Project Removal | Remediated |
| FYEO-NAT-06 | Informational | No Duplicate Project Name Prevention | Remediated |
| FYEO-NAT-07 | Informational | Role Revocation Behavior Differs from OpenZeppelin | Remediated |

Table 2: Findings Overview

## The Classification of vulnerabilities

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets

- The complexity to exploit is low

- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code

- All mismatches from the stated and actual functionality

- Unprotected key material

- Weak encryption of keys

- Badly generated key materials

- Txn signatures not verified

- Spending of funds through logic errors

- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries

- Use of untested or nonstandard or non-peer-reviewed crypto functions

- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions

- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations

## Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

## Conclusion

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

# Technical Findings

## General Observations

NatGold is a blockchain-based system designed to tokenize gold-backed mining projects on the Base blockchain. It converts approved mining projects into ERC-20 tokens through a structured smart contract process. The system uses two core contracts: `NatGoldToken.sol`, which manages token minting and distribution, and `NatGoldQueueOrchestrator.sol`, which manages project intake, approval, and lifecycle state. Projects move through a defined state machine (queued, minted, or rejected) and are processed using a First-In-First-Out queue to ensure deterministic ordering. Role-based access control limits administrative actions such as approvals and minting to authorized roles, and all project state changes are recorded on-chain.

Administrative operations are governed through a Gnosis Safe multisignature setup, requiring multiple approvals for actions such as project acceptance, mint execution, and contract upgrades. The system supports percentage-based token distribution across multiple wallets to reflect ownership structures. Both core contracts use the UUPS upgradeable proxy pattern to allow contract logic updates while preserving state. A React and TypeScript frontend provides functionality for project submission, queue visibility, and transaction tracking. Overall, the system provides a structured and auditable mechanism for managing the lifecycle and tokenization of gold mining projects on-chain.

## Admin Can Accidentally Renounce Critical Roles

Finding ID: FYEO-NAT-01
Severity: Informational
Status: Remediated

### Description

The contracts use OpenZeppelin's AccessControl which allows role holders to renounce their roles. If the admin accidentally renounces critical roles like DEFAULT_ADMIN_ROLE, MINTER_ROLE, or UPGRADER_ROLE, the contract could become unmanageable or lose essential functionality.

### Proof of Issue

**File name:** contract/src/NatGoldToken.sol
**File name:** contract/src/NatGoldQueueOrchestrator.sol

```
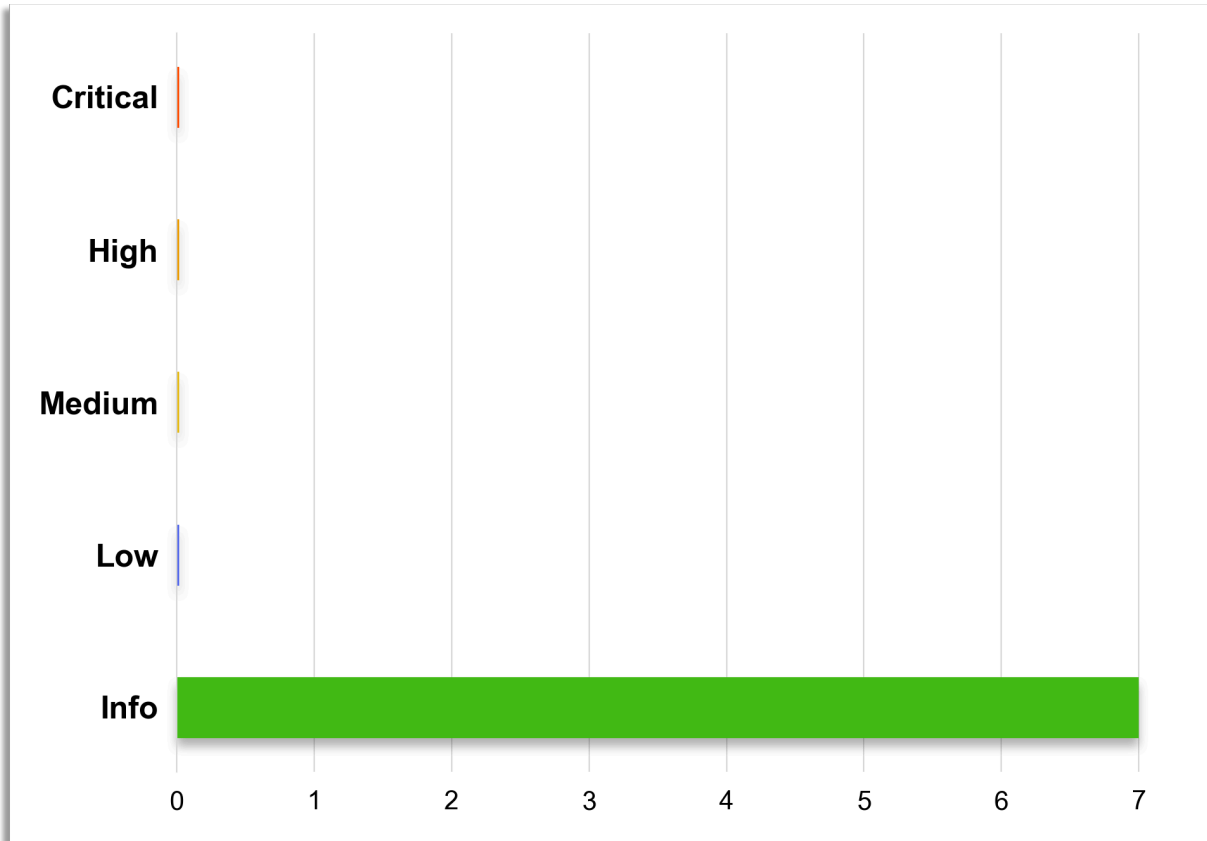// AccessControlUpgradeable allows role renunciation
// Admin could accidentally call:
// renounceRole(DEFAULT_ADMIN_ROLE, adminAddress)
// This would make the contract unmanageable
```

### Severity and Impact Summary

Accidental role renunciation could result in loss of administrative control, inability to upgrade contracts, or inability to mint tokens. This could effectively brick the contract functionality.

### Recommendation

Override the renounceRole function to add additional safeguards for critical roles, such as preventing renunciation of the last admin role holder.

# Integer Division Precision Loss in Token Distribution

Finding ID: FYEO-NAT-02
Severity: Informational
Status: Remediated

## Description

The token distribution calculation uses integer division which can result in precision loss, leaving "dust" tokens that are never distributed. The sum of distributed amounts may be less than the total authorized amount.

## Proof of Issue

**File name:** contract/src/NatGoldQueueOrchestrator.sol
**Line number:** 313

```
function executeMint(uint256 projectId) external onlyRole(QUEUE_ADMIN_ROLE) {
    ...
    for (uint256 i = 0; i < project.distribution.length; i++) {
        DistributionConfig memory dist = project.distribution[i];
        uint256 distributionAmount = (amount * dist.percentage) / 10_000;
        // Integer division can leave remainder - dust tokens lost

        if (distributionAmount > 0) {
            natGoldToken.mint(dist.wallet, distributionAmount);
        }
    }
```

## Severity and Impact Summary

Tokens can be permanently lost due to rounding errors. This creates accounting discrepancies between authorized and actual token supply.

## Recommendation

Implement a mechanism to handle remainder tokens, such as distributing the remainder to the last wallet in the distribution list.

# Missing Safe UPGRADER_ROLE Validation in ProposeUpgrade

Finding ID: FYEO-NAT-03
Severity: Informational
Status: Remediated

## Description

The ProposeUpgrade script does not verify that the Safe multisig has the UPGRADER_ROLE before proposing upgrade transactions. This could lead to wasted gas and confusion if upgrades are proposed to a Safe that cannot execute them.

## Proof of Issue

**File name:** contract/script/ProposeUpgrade.s.sol
**Line number:** 45

```
function _proposeTokenUpgrade(address proxyAddress) internal {
    console.log("--- Token Upgrade ---");

    // Get current state
    NatGoldToken proxy = NatGoldToken(proxyAddress);
    string memory currentVersion = proxy.version();

    // Deploy new implementation
    console.log("Deploying new implementation...");
    NatGoldToken newImpl = new NatGoldToken();
    // ... continues without checking Safe has UPGRADER_ROLE
}
```

## Severity and Impact Summary

Proposed upgrade transactions will fail during execution if Safe lacks UPGRADER_ROLE. Wastes time and creates confusion. Early validation would catch configuration errors before deployment.

## Recommendation

Add role validation before deploying implementation and proposing transaction to fail fast on misconfiguration.

## Missing Storage Gaps in Upgradeable Contracts

Finding ID: FYEO-NAT-04
Severity: Informational
Status: Remediated

### Description

Both NatGoldToken.sol and NatGoldQueueOrchestrator.sol are upgradeable contracts using the UUPS pattern but lack storage gap reservations. Storage gaps are important for upgradeable contracts to prevent storage layout conflicts during upgrades.

### Proof of Issue

**File name:** contract/src/NatGoldToken.sol

```
contract NatGoldToken is Initializable, ERC20Upgradeable, AccessControlUpgradeable,
UUPSUpgradeable {
    // Missing: uint256[50] private __gap;
}
```

**File name:** contract/src/NatGoldQueueOrchestrator.sol

```
contract NatGoldQueueOrchestrator is Initializable, AccessControlUpgradeable,
UUPSUpgradeable {
    // Missing: uint256[50] private __gap;
}
```

### Severity and Impact Summary

Future contract upgrades could corrupt storage layout, leading to data corruption or system failure.

### Recommendation

Add storage gap reservations to both contracts to reserve space for future storage variables during upgrades.

## Missing Validation in Queue Project Removal

Finding ID: FYEO-NAT-05
Severity: Informational
Status: Remediated

### Description

The `rejectProject` function searches through the `queuedProjects` array to find and remove a project, but does not verify that the project was actually found in the queue before completing the rejection process.

### Proof of Issue

**File name:** contract/src/NatGoldQueueOrchestrator.sol
**Line number:** 275

```
function rejectProject(uint256 projectId) external onlyRole(QUEUE_ADMIN_ROLE) {
    ...

    // Remove from queue array
    for (uint256 i = 0; i < queuedProjects.length; i++) {
        if (queuedProjects[i] == projectId) {
            // Shift left
            for (uint256 j = i; j < queuedProjects.length - 1; j++) {
                queuedProjects[j] = queuedProjects[j + 1];
            }
            queuedProjects.pop();
            break;
        }
    }
    ...
```

### Severity and Impact Summary

If a project exists but is not in the queue (should not happen), the function will silently fail to remove it from the queue while still marking it as rejected, leading to inconsistent contract state.

### Recommendation

Add validation to ensure the project was found and removed from the queue before proceeding with state changes, or revert if the project is not found in the queue.

# No Duplicate Project Name Prevention

Finding ID: FYEO-NAT-06
Severity: Informational
Status: Remediated

## Description

The contract allows multiple projects to be created with identical names, which could lead to confusion, or difficulties in project identification and auditing.

## Proof of Issue

**File name:** contract/src/NatGoldQueueOrchestrator.sol
**Line number:** 183

```
function approveProject(
    string calldata projectName,
    // ... other parameters
) external onlyRole(QUEUE_ADMIN_ROLE) returns (uint256 projectId) {
    // Required field validations
    require(bytes(projectName).length > 0, "Orchestrator: project name is empty");
    // No check for duplicate project names

    // Create project
    project.projectName = projectName;
    // ...
}
```

## Severity and Impact Summary

Multiple projects with identical names could cause confusion for users and administrators. This could potentially be exploited for fraudulent purposes or make project tracking and management more difficult.

## Recommendation

Consider implementing a mechanism to prevent duplicate / similar project names.

# Role Revocation Behavior Differs from OpenZeppelin

Finding ID: FYEO-NAT-07
Severity: Informational
Status: Remediated

## Description

The custom role management implementation reverts when attempting to revoke or renounce a role from an account that doesn't hold it. This differs from OpenZeppelin's standard behavior where revoking a non-existent role is a no-op. The revert occurs because the code checks `_roleMemberCount[role] <= 1` before verifying if the account actually has the role.

## Proof of Issue

**File name:** contract/src/NatGoldToken.sol
**File name:** contract/src/NatGoldQueueOrchestrator.sol

```
function revokeRole(bytes32 role, address account) public virtual override {
    if (_isCriticalRole(role) && _roleMemberCount[role] <= 1) {
        revert CannotRemoveLastRoleHolder(role);
    }
    bool hadRole = hasRole(role, account);
    super.revokeRole(role, account);
    if (hadRole) {
        _roleMemberCount[role]--;
    }
}
```

If `_roleMemberCount[role]` is 1 and the account doesn't have the role, the function reverts. OpenZeppelin's `_revokeRole` only emits an event if the account had the role.

## Severity and Impact Summary

Breaks compatibility with OpenZeppelin's AccessControl behavior. Scripts or integrations expecting no-op behavior on non-holders would fail. Could cause issues in automated role management systems.

## Recommendation

Check `hasRole(role, account)` before the count validation to maintain OpenZeppelin compatibility and only revert when actually removing the last holder.

# Our Process

## Methodology

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.



Figure 2: Methodology Flow

## Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact

- Communication methods and frequency

- Shared documentation

- Code and/or any other artifacts necessary for project success

- Follow-up meeting schedule, such as a technical walkthrough

- Understanding of timeline and duration

## Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers

- Reviewing programming language constructs for specific languages

- Researching common flaws and recent technological advancements

## Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol

2. Review of the code written for the project

3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

## Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues

- Poor coding practices and unsafe behavior

- Leakage of secrets or other sensitive data through memory mismanagement

- Susceptibility to misuse and system errors

- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

## Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases

- Proper error handling

- Adherence to the protocol logical description

## Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical

- High

- Medium

- Low

- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

## Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

## Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.