



# AUDIT REPORT

---




October 2025

For





Global Gold

# Table of Content

Executive Summary	04
Number of Security Issues per Severity	06
Summary of Issues	07
Checked Vulnerabilities	09
Techniques and Methods	11
Types of Severity	13
Types of Issues	14
Severity Matrix	15
 <b>Critical Severity Issues</b>	16
1. Missing Access Control for GUARDIAN_RESET_ACTION_THRESHOLD	16
2. Complete Recovery Cancellation Bypass - Execution Flow Bug Combined with Logic Error Enables Asset Theft	17
 <b>High Severity Issues</b>	19
3. Absence of Guardians Permanently Locks Assets	19
4. Guardian Recovery Attack Vulnerability	20
5. Vault Owner Can Bypass Guardian System by Adding Themselves as Guardian	21
 <b>Medium Severity Issues</b>	22
6. Use a safe transfer helper library for ERC20 and ERC721 transfers	22
7. Silent Failure in Recovery Cancellation	23
8. Action Threshold Can Be Set Higher Than Guardian Count	24
9. Unrestricted initialize() Allows Unauthorized Ownership Takeover	25
10. Unrestricted minting, unsafe external call ordering, and weak post-mint verification	26



 <b>Low Severity Issues</b>	27
11. Missing Balance Validation in ERC1155 Deposit	27
12. Use Ownable2Step version rather than Ownable version	28
13. Missing return value check for ERC-20 transferFrom	29
14. Inconsistent assetCount variable	30
 <b>Informational Issues</b>	31
15. Errors and modifiers declared locally instead of reusing centralized definitions	31
16. Interfaces do not expose all implemented function signatures	32
Additional Global Gold Team Notes	33
Functional Tests	36
Automated Tests	39
Threat Model	39
Closing Summary & Disclaimer	40



# Executive Summary

<b>Project Name</b>	Global Gold
<b>Protocol Type</b>	Vault
<b>Project URL</b>	<a href="https://globalgold.finance/">https://globalgold.finance/</a>
<b>Overview</b>	<p>The Global Gold Vault system provides a secure, upgradeable smart contract framework for managing digital assets (ERC20, ERC721, ERC1155) with multi-signature guardian protection. Each vault is represented as an NFT, allowing for easy ownership transfer and verification.</p> <p><b>Key Features</b></p> <ul style="list-style-type: none"><li><b>NFT-Based Vaults:</b> Each vault is represented by a unique NFT, making ownership transfer seamless</li><li><b>Guardian Protection:</b> Multi-signature protection with customizable thresholds</li><li><b>Asset Management:</b> Support for ERC20, ERC721, and ERC1155 tokens</li><li><b>Recovery System:</b> Guardian-based recovery mechanism for lost access</li><li><b>Upgradeable:</b> Uses OpenZeppelin's upgradeable contracts pattern</li><li><b>Emergency Functions:</b> Emergency withdrawal capabilities for non-protected assets</li></ul>
<b>Audit Scope</b>	The scope of this Audit was to analyze the GlobalGold Smart Contracts for quality, security, and correctness.
<b>Source Code link</b>	<a href="https://github.com/EnterTheRYFT/global_gold_contracts">https://github.com/EnterTheRYFT/global_gold_contracts</a>
<b>Branch</b>	Main
<b>Contracts in Scope</b>	contracts/interfaces/IGlobalGoldVault.sol contracts/interfaces/IGlobalGoldVaultNFT.sol contracts/interfaces/IGlobalGoldVaultAdmin.sol contracts/GlobalGoldVaultNFT.sol contracts/Utils/WithdrawableUpgradeable.sol contracts/Utils/Modifiers.sol contracts/Utils/ProxyableUpgradeable.sol contracts/Utils/Errors.sol contracts/GlobalGoldVault.sol contracts/Guardian.sol contracts/GlobalGoldVaultAdmin.sol
<b>Commit Hash</b>	286861c68f0e630e99ef3958bfe4e21e86be877d



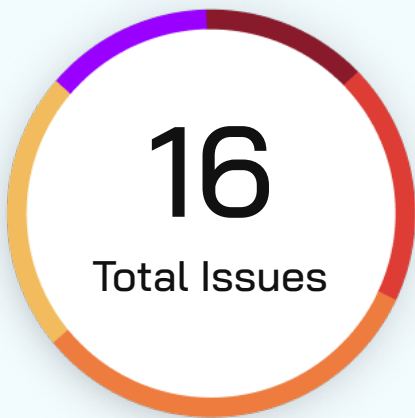
<b>Language</b>	Solidity
<b>Blockchain</b>	EVM
<b>Method</b>	Manual Analysis, Functional Testing, Automated Testing
<b>Review 1</b>	1st September 2025 - 22nd September 2025
<b>Updated Code Received</b>	25th September 2025
<b>Review 2</b>	26th September 2025 - 2nd October 2025
<b>Fixed In</b>	<a href="https://github.com/EnterTheRYFT/global_gold_contracts/commits/audit_fixes-v1.0/contracts">https://github.com/EnterTheRYFT/global_gold_contracts/commits/audit_fixes-v1.0/contracts</a>

**Verify the Authenticity of Report on QuillAudits Leaderboard:**

<https://www.quillaudits.com/leaderboard>



# Number of Issues per Severity



Critical	2 (12.5%)
High	3 (18.75%)
Medium	5 (31.25%)
Low	4 (25.0%)
Informational	2 (12.5%)

		Severity				
		Critical	High	Medium	Low	Informational
Issues	Open	0	0	0	0	0
	Acknowledged	0	0	0	1	1
	Partially Resolved	0	0	0	0	0
	Resolved	2	3	5	3	1



# Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Missing Access Control for GUARDIAN_RESET_ACTION_THRESHOLD	Critical	Resolved
2	Complete Recovery Cancellation Bypass - Execution Flow Bug Combined with Logic Error Enables Asset Theft	Critical	Resolved
3	Absence of Guardians Permanently Locks Assets	High	Resolved
4	Guardian Recovery Attack Vulnerability	High	Resolved
5	Vault Owner Can Bypass Guardian System by Adding Themselves as Guardian	High	Resolved
6	Use a safe transfer helper library for ERC20 and ERC721 transfers	High	Resolved
7	Silent Failure in Recovery Cancellation	Medium	Resolved
8	Action Threshold Can Be Set Higher Than Guardian Count	Medium	Resolved
9	Unrestricted initialize() Allows Unauthorized Ownership Takeover	Medium	Resolved



Issue No.	Issue Title	Severity	Status
10	Unrestricted minting, unsafe external call ordering, and weak post-mint verification	Medium	Resolved
11	Missing Balance Validation in ERC1155 Deposit	Low	Resolved
12	Use Ownable2Step version rather than Ownable version	Low	Acknowledged
13	Missing return value check for ERC-20 transferFrom	Low	Resolved
14	Inconsistent assetCount variable	Low	Resolved
15	Errors and modifiers declared locally instead of reusing centralized definitions	Informational	Resolved
16	Interfaces do not expose all implemented function signatures	Informational	Acknowledged





# Checked Vulnerabilities

✓ Access Management

✓ Arbitrary write to storage

✓ Centralization of control

✓ Ether theft

✓ Improper or missing events

✓ Logical issues and flaws

✓ Arithmetic Computations  
Correctness

✓ Race conditions/front running

✓ SWC Registry

✓ Re-entrancy

✓ Timestamp Dependence

✓ Gas Limit and Loops

✓ Exception Disorder

✓ Gasless Send

✓ Use of tx.origin

✓ Malicious libraries

✓ Compiler version not fixed

✓ Address hardcoded

✓ Divide before multiply

✓ Integer overflow/underflow

✓ ERC's conformance

✓ Dangerous strict equalities

✓ Tautology or contradiction

✓ Return values of low-level calls



✓ **Missing Zero Address Validation**

✓ **Private modifier**

✓ **Revert/require functions**

✓ **Multiple Sends**

✓ **Using suicide**

✓ **Using delegatecall**

✓ **Upgradeable safety**

✓ **Using throw**

✓ **Using inline assembly**

✓ **Style guide violation**

✓ **Unsafe type inference**

✓ **Implicit visibility level**

# Techniques and Methods

**Throughout the audit of smart contracts, care was taken to ensure:**

- The overall quality of code
- Use of best practices
- Code documentation and comments, match logic and expected behavior
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Implementation of ERC standards
- Efficient use of gas
- Code is safe from re-entrancy and other vulnerabilities

**The following techniques, methods, and tools were used to review all the smart contracts:**

## Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

## Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.



### Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

### Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

### Tools and Platforms Used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity Static Analysis.



# Types of Severity

Every issue in this report has been assigned to a severity level. There are five levels of severity, and each of them has been explained below.

## ■ **Critical: Immediate and Catastrophic Impact**

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

## ■ **High (H): Significant Risk of Major Loss or Compromise**

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

## ■ **Medium (M): Potential for Moderate Harm Under Specific Circumstances**

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

## ■ **Low (L): Minor Imperfections with Limited Repercussions**

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

## ■ **Informational (I): Opportunities for Improvement, Not Immediate Risks**

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.



# Types of Issues

<div>Open</div> <p>Security vulnerabilities identified that must be resolved and are currently unresolved.</p>	<div>Resolved</div> <p>These are the issues identified in the initial audit and have been successfully fixed.</p>
<div>Acknowledged</div> <p>Vulnerabilities which have been acknowledged but are yet to be resolved.</p>	<div>Partially Resolved</div> <p>Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.</p>



# Severity Matrix

		Impact		
		High	Medium	Low
Likelihood	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low

## Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

## Likelihood

- **High** - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- **Medium** - only a conditionally incentivized attack vector, but still relatively likely.
- **Low** - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.



# Critical Severity Issues

## Missing Access Control for GUARDIAN\_RESET\_ACTION\_THRESHOLD

**Resolved**

### Path

contracts/GlobalGoldVault.sol

### Function Name

`submitActionRequest()`

### Description

The `GUARDIAN_RESET_ACTION_THRESHOLD` action is missing from all access control classification functions, which should cause the function to revert with `InvalidActionId()`.

The action `GUARDIAN_RESET_ACTION_THRESHOLD` is implemented in the guardian system but not properly classified in the access control system. This creates an inconsistency between the action's availability and its permissions.

When `submitActionRequest()` is called with `GUARDIAN_RESET_ACTION_THRESHOLD`, the `_checkActionPermissions()` function should:

1. Check if it's a vault owner only action → NO (not in `_isVaultOwnerOnlyAction`)
2. Check if it's a guardian only action → NO (not in `_isGuardianOnlyAction`)
3. Check if it's a vault owner or guardian action → NO (not in `_isVaultOwnerOrGuardianAction()`)
4. Reach the final line and revert with `InvalidActionId()`

### Impact

Would not be able to call `GUARDIAN_RESET_ACTION_THRESHOLD` as everytime it would revert.

### Recommendation

Add `GUARDIAN_RESET_ACTION_THRESHOLD` to the appropriate classification function. Based on its functionality (resetting action thresholds), it should be a vault owner only action

### Specific Fixed In Commit

5c6460e - feat: Comprehensive audit response implementation

### Global Gold Team's Response

Added `GUARDIAN_RESET_ACTION_THRESHOLD` to `GlobalGoldVault._isVaultOwnerOnlyAction`

### QuillAudits Team's Response

The issue has been fixed by adding `GUARDIAN_RESET_ACTION_THRESHOLD` to `GlobalGoldVault._isVaultOwnerOnlyAction`





## Complete Recovery Cancellation Bypass - Execution Flow Bug Combined with Logic Error Enables Asset Theft

Resolved

### Path

contracts/Guardian.sol

### Function Name

`_executeAction()`

### Description

A critical vulnerability exists in the recovery cancellation system involving two interconnected bugs that together create a complete bypass of the cancellation mechanism. Users attempting to cancel recovery requests through `GUARDIAN_CANCEL_REQUEST` receive false confirmation of cancellation while the recovery remains fully executable after the time delay.

Bug #1: Incomplete Guardian Request Cancellation (Execution Flow)  
When `GUARDIAN_CANCEL_REQUEST` is executed, the system only partially cancels the request due to an execution flow bug in the inheritance chain.

```
// Guardian.sol _executeAction() - Executes first
else if (actionId_ == GUARDIAN_CANCEL_REQUEST) {
    uint256 requestId = uintParams_[0];
    _cancelRequest(requestId); // ✅ Executes - Guardian request cancelled
    emit ActionCancelled(requestId, msg.sender);
    return; // ⚠️ Execution stops here - CRITICAL BUG
}

// GlobalGoldVault.sol _executeCustomAction() - NEVER reached
else if (actionId_ == GUARDIAN_CANCEL_REQUEST) {
    uint256 requestId = uintParams_[0];
    _cancelRecoveryIfExists(requestId); // ❌ Never executes - Recovery remains active
}
```

Bug #2: Recovery Execution Logic Error

The `executeRecovery` function contains flawed validation logic that allows execution even when the guardian request has been cancelled. Vulnerable Code:

```
function executeRecovery(bytes32 requestId_) external {
    // ... other checks ...

    // FLAWED LOGIC: Uses AND instead of proper validation
    if (
        !IGlobalGoldVault(request.requestingVault).isPendingRequest(
            request.guardianRequestId
        ) &&
        !IGlobalGoldVault(request.requestingVault)
            .getRequest(request.guardianRequestId)
            .finalized
    ) revert GuardianRequestCancelled();

    // Recovery proceeds despite cancellation
}
```



### Combined Attack Vector

The two bugs create a perfect storm for asset theft:

State After "Cancellation":

- Guardian request: `isPendingRequest()` = false (removed from pending set)
- Guardian request: `finalized` = true (marked as finalized during cancellation)
- NFT Recovery request: Still active (never cancelled due to Bug #1)

Logic Evaluation in `executeRecovery`:

```
!isPendingRequest(guardianRequestId) = !false = true  
!getRequest(guardianRequestId).finalized = !true = false
```

**Condition: `true && false = false`**

**Result: Does NOT revert - Recovery executes successfully**

### Impact

Guardian system shows the request as cancelled while the NFT recovery process remains active in the external contract. Recovery operations could potentially continue executing even after users believe they have been cancelled. Vault ownership transferred to an address which was not supposed to receive.

### Recommendation

The immediate priority is implementing a two-pronged fix to address both underlying issues simultaneously. First, the execution flow must be corrected by moving the recovery cancellation logic from the child contract's `_executeCustomAction` method into the parent contract's `_executeAction` method, ensuring that when `GUARDIAN_CANCEL_REQUEST` is processed, both the guardian request cancellation and the associated NFT recovery cancellation occur in sequence before the early return statement.

### Specific Fixed In Commit

5c6460e - feat: Comprehensive audit response implementation

### Global Gold Team's Response

Removed the return statement for all actions in `Guardian._executeAction`. This provides more flexibility for inheriting contracts to perform additional functions for these standard actions.

### QuillAudits Team's Response

The issue has been fixed by removing the return statement for all the actions.



# High Severity Issues

## Absence of Guardians Permanently Locks Assets

**Resolved**

### Path

contracts/GlobalGoldVault.sol

### Function

`submitActionRequest()`

### Description

The vault recovery mechanism requires guardian approval to execute VAULT\_RECOVER and VAULT\_EXECUTE\_RECOVERY actions. However, if no guardians are added to the vault during setup, or all guardians are removed/compromised, there is no way to recover the vault even in legitimate emergency scenarios. The vault owner loses permanent access to their assets with no recovery path.

### Impact

Assets become permanently inaccessible if guardians are unavailable.

### Remediation

Require at least one guardian to be added during vault creation, preventing users from accidentally deploying an unrecoverable vault.

### Specific Fixed In Commit

5c6460e - feat: Comprehensive audit response implementation

### Global Gold Team's Response

Modified GlobalGoldVault.\_initialize to require at least one guardian address and default threshold as parameters. This ensures there are one or more guardians. Additionally, modifications were made to Guardian.\_bulkRemoveGuardians, Guardian.\_removeGuardian, and Guardian.\_bulkReplaceGuardians to ensure that at least one guardian is always present.

### QuillAudits Team's Response

The issue has been fixed by requiring at least one guardian address and default threshold as parameters.



## Guardian Recovery Attack Vulnerability

**Resolved**

### Path

contracts/GlobalGoldVault.sol

### Function

`submitActionRequest()`

### Description

A vulnerability exists in the GlobalGoldVault contract that allows a single malicious guardian to steal all vault assets through the recovery mechanism.

### Attack Scenario:

- A malicious guardian (or compromised guardian account) initiates a VAULT\_RECOVER action
- If the action threshold for VAULT\_RECOVER is set to 1 (or defaults to 1), the recovery is automatically executed
- The guardian specifies themselves or an attacker-controlled address as the new owner
- The vault ownership is transferred to the attacker
- The attacker can then withdraw all assets from the vault

### This attack is particularly dangerous because:

- It bypasses the intended multi-signature security model
- Recovery actions are meant for emergency situations but can be abused
- The victim vault owner has no opportunity to intervene once the action is submitted

### Impact

Permanent loss of vault ownership to malicious actors if guardian is compromised.

### Remediation

Enforce minimum thresholds for Critical actions and vault owner should ensure only trusted guardians should be added.

### Global Gold Team's Response

This is per design and similar to other multi-sig systems. It is always the case that a 0 of 1 threshold, regardless of number of addresses added as signators, transactions can be executed by a single signer.

### QuillAudits Team's Response

The issue has been acknowledged by the team.



## Vault Owner Can Bypass Guardian System by Adding Themselves as Guardian

**Resolved**

### Path

contracts/GlobalGoldVault.sol

### Function

`submitActionRequest()`

### Description

The vault's guardian system is designed to enforce separation of duties, preventing a compromised vault owner from unilaterally approving sensitive actions. While the contract prevents the vault owner from adding themselves as a guardian through the GUARDIAN\_ADD action, this restriction does not exist in bulk operations.

Specifically, in `_bulkAddGuardians`, there is no validation to prevent the vault owner from being added as a guardian. Additionally, if the vault owner was previously added as a guardian (e.g., via bulk add), transferring the vault NFT does not invalidate that guardian role.

This allows the vault owner to bypass the intended guardian system and approve their own actions, rendering the mechanism ineffective.

### Impact

Vault owner can add themselves as a guardian through bulk operations.

### Remediation

Add validation in bulk guardian operations to prevent the vault owner from being added as a guardian.

### Specific Fixed In Commit

5c6460e - feat: Comprehensive audit response implementation

### Global Gold Team's Response

Modified all bulk operations to ensure vault owner address cannot be added as a guardian.

### QuillAudits Team's Response

The issue has been fixed by adding a check that vault owner address cannot be added as a guardian



# Medium Severity Issues

## Use a safe transfer helper library for ERC20 and ERC721 transfers

**Resolved**

### Path

contracts/GlobalGoldVault.sol

### Path

`depositERC20()` and other functions

### Description

TheGlobalGoldVault function's assumes that every ERC-20/ERC721 token it handles conforms strictly to the OpenZeppelin-style interface and returns a Boolean flag on transfer. In practice a significant subset of production tokens, either return void (no data) or revert on failure while returning no value. When such a token is passed to transfer(IERC20 token), the high-level Solidity call expects a bool but receives zero bytes, causing ABI decoding to revert. The revert bubbles up, cancelling the entire transaction.

### Specific Fixed In Commit

17708a1 - feat: Enhance guardian management by capping bulk additions and implementing SafeERC20 for secure token transfers

### Global Gold Team's Response

Modified all token transfers to use safeTransfer.



## Silent Failure in Recovery Cancellation

**Resolved**

### Path

contracts/GlobalGoldVault.sol

### Path

`_cancelRecoveryIfExists()`

### Description

The `_cancelRecoveryIfExists` function in the `GlobalGoldVault` contract uses a try-catch block that silently swallows all exceptions, creating a false sense of security when recovery cancellation fails. This design flaw allows the system to continue execution even when critical cancellation operations fail, potentially leaving users vulnerable to unauthorized recovery executions.

### Specific Fixed In Commit

5c6460e - feat: Comprehensive audit response implementation

### Global Gold Team's Response

Added `GlobalGoldVaultNFT.hasActiveRecoveryRequest` that responds true if a recovery request is found and the request has not been execute or cancelled. Modified `GlobalGoldVault._cancelRecoveryIfExists` to check for active recovery, properly cancel recovery if it exists, and properly revert if checking for active recovery fails.



## Action Threshold Can Be Set Higher Than Guardian Count

**Resolved**

### Path

contracts/Guardian.sol

### Path

`_setThreshold()`

### Description

The `_setThreshold` function allows setting an action-specific guardian threshold higher than the number of action-specific guardians, as long as the value is less than or equal to the total number of default guardians.

This introduces a mismatch:

- When `useActionGuardians[actionId]` is enabled, approvals are counted only from action-specific guardians.
- However, `_setThreshold` only validates the threshold against the global guardian count (`guardians.length`).

As a result, it is possible to set thresholds higher than the available action-specific guardians, making certain actions impossible to finalize.

### Specific Fixed In Commit

f054e2c - feat: Implement auto-adjustment of guardian thresholds during bulk removal and replacement

### Global Gold Team's Response

Modified `Guardian._setThreshold` to use guardian count from the action if the action is not using default guarding threshold.





## Unrestricted initialize() Allows Unauthorized Ownership Takeover

**Resolved**

### Path

contracts/GlobalGoldVaultAdmin.sol

### Path

`initialize()`

### Description

The `GlobalGoldVaultAdmin.initialize()` function is declared as public and can be called by any external account. Since this function sets the contract owner (`__Ownable_init(msg.sender)`) and grants proxy privileges (`setProxyState(msg.sender,true)`), the first caller to invoke `initialize()` after deployment will gain full administrative control over the contract. If the contract is deployed without immediately calling `initialize()` in the deployment script, a malicious actor can frontrun and seize ownership and proxy rights, resulting in a permanent takeover of the system. Call `_disableInitializers()` in the constructor to lock the implementation contract and prevent it from being initialized directly.

### Specific Fixed In Commit

5c6460e - feat: Comprehensive audit response implementation

### Global Gold Team's Response

Added `GlobalGoldVaultAdmin.constructor` that calls `_disableInitializers()`



## Unrestricted minting, unsafe external call ordering, and weak post-mint verification

**Resolved**

### Description

The safeMint function is exposed as external and only checks that the to address is not zero. This allows any caller to mint a new token and force deployment of a new vault, which is likely unintended. In addition, the function invokes IGlobalGoldVaultNFT.safeMint before updating internal state (tokenIdToVault, \_tokenId). Because ERC-721 safeMint will call onERC721Received if the recipient is a contract, a malicious recipient could exploit this ordering to reenter the function before state changes occur, increasing the reentrancy risk. Finally, the post-mint verification only checks that ownerOf(\_tokenId) is non-zero, but does not verify that the token was minted to the intended to address.

### Impact

High

### Likelihood

Medium

### Recommendation

Restrict access with an onlyOwner or onlyMinter modifier.

Apply checks-effects-interactions: increment \_tokenId before external calls, and consider nonReentrant protection.

Replace the current ownership check with a strict validation that the minted token belongs to to (e.g., require(ownerOf(tid) == to)).

### Specific Fixed In Commit

286861c - feat: Add error handling for GlobalGoldVault minting process and improve safeMint function

### Global Gold Team's Response

GlobalGoldVaultAdmin.safeMint is expected to be a public function. Added nonReentrant to GlobalGoldVaultAdmin.safeMint. Modified order of operations to increment tokenId early, deploy the GlobalGoldVault contract, confirm the owner of token ID is the to\_ address, and emit a proper SafeMint event.



# Low Severity Issues

## Missing Balance Validation in ERC1155 Deposit

**Resolved****Path**

contracts/GlobalGoldVault.sol

**Function name**

`depositERC1155()`

**Description**

The `depositERC1155` function lacks a balance check before attempting the transfer, unlike `depositERC20` which validates `IERC20(erc20Contract_).balanceOf(msg.sender) >= amount_`. This causes unhelpful error messages when users attempt to deposit more ERC1155 tokens than they own, as the revert will come from the ERC1155 contract rather than a clear "InsufficientBalance" error. The function should check `IERC1155(erc1155Contract_).balanceOf(msg.sender, tokenId_) >= amount_` before the transfer.

**Specific Fixed In Commit**

5c6460e - feat: Comprehensive audit response implementation

**Global Gold Team's Response**

Added balance check.



## Use Ownable2Step version rather than Ownable version

**Acknowledged**

### Path

contracts/GlobalGoldVaultAdmin.sol

### Function name

`constructor()`

### Description

Ownable2Step prevent the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner's permissions actively accept via a contract call of its own.

Consider using Ownable2Step from OpenZeppelin Contracts to enhance the security of your contract ownership management. This contract prevents the accidental transfer of ownership to an address that cannot handle it, such as due to a typo, by requiring the recipient of owner permissions to actively accept ownership via a contract call.

### Global Gold Team's Response

Determined to not be necessary.



## Missing return value check for ERC-20 transferFrom

**Resolved**

### Path

contracts/GlobalGoldVault.sol

### Function name

```
depositERC20(), depositERC721(), emergencyWithdrawERC721(),  
_executeTransferERC721(), _executeWithdrawERC721(), _transferVault()
```

### Description

In the current implementation, if transferFrom fails silently by returning false, the function will continue execution and emit a DepositERC20 event even though no tokens were transferred. This creates a mismatch between the contract's state and its actual token holdings, potentially leading to accounting inconsistencies and loss of user funds.

### Impact

low

### Likelihood

Low

### Recommendation

It is recommended to use OpenZeppelin's SafeERC20.safeTransferFrom library function instead of calling transferFrom directly. The SafeERC20 wrapper ensures that transfers revert on failure whether the token reverts or returns false. The redundant balance check (IERC20.balanceOf) can also be removed, since a failed transferFrom will be safely handled by SafeERC20.

### Specific Fixed In Commit

17708a1 - feat: Enhance guardian management by capping bulk additions and implementing SafeERC20 for secure token transfers

### Global Gold Team's Response

Modified to use safeTransferFrom.



## Inconsistent assetCount variable

**Resolved**

### Path

contracts/GlobalGoldVaultAdmin.sol, contracts/GlobalGoldVaultNFT.sol

### Function name

`addAsset()`, `removeAsset()`

### Description

The contract `contracts/GlobalGoldVaultAdmin.sol` maintains an `assetCount` variable that is incremented whenever a new asset is added. However, this variable is not decremented when an asset is removed. As a result, `assetCount` does not accurately reflect the current number of assets and only ever increases over time. Additionally the `contracts/GlobalGoldVaultNFT.sol` contract declares the `assetCount` but never uses it.

This creates several risks:

- Incorrect state representation: The variable gives a misleading view of how many assets are present.
- Future misuse: If other developers or external systems later rely on `assetCount` (e.g., for reporting, validation, or iteration), they may work with inaccurate data.
- Wasted gas and storage: Maintaining and updating a variable that is never consumed adds unnecessary cost.

### Impact

Low

### Likelihood

Medium

### Recommendation

- If `assetCount` is intended to track the number of active assets, update the logic to also decrement it when assets are removed.
- If it is not needed, remove the variable entirely to simplify the contract and reduce gas/storage costs.

### Specific Fixed In Commit

5c6460e - feat: Comprehensive audit response implementation

### Global Gold Team's Response

`GlobalGoldVault.assetCount` is used as a count of assets that have been added to a mapping, not a count of active assets. This view may have been caused by the function `removeAsset`, which simply removed the `Asset` object from the mapping at the associated index, but it did not (and cannot) remove the asset index from the mapping. To resolve this issue in perception, the function `removeAsset` was removed, and `setAssetDepositAllowedByAddress` and `setAssetDepositAllowedById` functions were added that modify the `depositAllowed` property for existing assets.

Removed `GlobalGoldVaultNFT.assetCount`.



# Informational Issues

## Errors and modifiers declared locally instead of reusing centralized definitions

**Resolved**

### Path

All contracts

### Description

The codebase includes a dedicated files like Errors.sol intended to hold reusable custom errors, and likely a similar modular structure was intended for modifiers. However, in practice, all contracts declare their own errors and modifiers locally instead of importing them from the shared module.

This approach does not cause a functional vulnerability, but it breaks the intended modularity and introduces risks of inconsistency:

- Code duplication: The same error or modifier may exist in multiple places, increasing maintenance overhead.
- Inconsistent updates: If an error or modifier definition needs to change, not all contracts may be updated consistently.
- Reduced clarity: External reviewers or integrators may assume all custom errors/modifiers are centralized, but in reality they must scan multiple files to understand usage.

### Recommendation

- Consolidate common errors into Errors.sol and use them across contracts.
- Apply the same approach to modifiers if they are designed to be shared.
- This will restore the intended modular design, improve maintainability, and reduce the risk of inconsistent logic.

### Specific Fixed In Commit

5c6460e - feat: Comprehensive audit response implementation

### Global Gold Team's Response

Refactored errors and modifiers



## Interfaces do not expose all implemented function signatures

**Acknowledged**

### Path

All contracts

### Description

Some contracts in the codebase implement public or external functions that are not declared in their corresponding interfaces. This creates inconsistency between the actual contract surface and its advertised interface.

### Impact

- Integration gaps: External contracts, tools, and off-chain systems relying on the interfaces may be unaware of available functions.
- Auditability: Reviewers and integrators cannot fully understand the contract's callable surface from the interface alone, which complicates security review and integration.
- Maintainability: Future developers may assume the interface is complete and miss important functionality that exists only in the implementation.
- Risk of divergence: If additional functions are added to the contract but not mirrored in the interface, consumers of the interface may be out of sync with the actual behavior.

### Recommendation

- Ensure that all public/external functions implemented by the contract are declared in the corresponding interface..
- Keep events, errors, and type definitions centralized in the interface to provide a single source of truth for integrators.

### Global Gold Team's Response

This is intentional due to contract maximum size issues.





# Additional Global Gold Team Notes - Major Architectural Refactoring

Beyond addressing the specific audit findings, the GlobalGold team implemented several significant architectural improvements to enhance security, efficiency, and usability. These changes were driven by contract size constraints, edge case discoveries, and fundamental logic improvements.

## 1. GlobalGoldVaultFactory Contract Implementation

**Challenge:** The GlobalGoldVaultAdmin contract exceeded Ethereum's maximum contract size limit due to the complexity of vault deployment logic.

**Solution:** Created a dedicated GlobalGoldVaultFactory contract implementing the EIP-1167 minimal proxy pattern for ultra-efficient vault deployment.

**Key Benefits:**

1. 90% Gas Reduction: Vault deployment costs reduced from ~3M gas to ~300K gas
2. Size Optimization: Removed deployment logic from admin contract, staying within size limits
3. Consistency: All vaults use identical implementation, eliminating deployment variations
4. Upgradeability: Centralized implementation enables future improvements

**Architecture Changes:**

- GlobalGoldVaultAdmin focuses solely on management and asset configuration
- GlobalGoldVaultFactory handles all vault deployment using minimal proxies
- Single implementation contract deployed once, cloned for each new vault
- Maintains full functionality while dramatically reducing deployment costs

## 2. Submitter vs. Approver Separation

**Original Problem:** The legacy approval system created mathematical impossibilities and user confusion:

- Submitting an action automatically counted as 1 approval
- With 5 guardians and a 5-of-5 threshold, only 4 additional approvals were needed
- The 5th guardian's approval was mathematically impossible to require
- Thresholds could never utilize the full guardian count

**Fundamental Redesign:** Complete separation of submission and approval roles:

**New Logic:**

- Submitters cannot approve their own requests (prevents self-approval attacks)
- Thresholds apply exclusively to approvers (clear mathematical model)
- Zero thresholds allowed (enables immediate execution for trusted operations)
- Full guardian utilization possible (N-1 of N maximum prevents deadlock)



Implementation Details:

```
// Prevents self-approval
if (msg.sender == r.submitter) revert CannotApproveOwnRequest();

// Approval count starts at 0, not 1
r.approvalCount = 0; // Submitter doesn't auto-approve

// Threshold validation prevents deadlock
if (threshold_ > uint8(guardianCount - 1)) revert
ThresholdWouldCauseDeadlock();
```

### 3. Sequential Transaction Ordering System

Original Issue: Random transaction execution order created race conditions where:

- Guardian configurations could change between submission and execution
- Threshold modifications could invalidate pending requests
- System state could become inconsistent during execution

Solution: Implemented sequential execution with intelligent cancellation handling:

Features:

- Sequential Processing: Transactions execute in submission order (FIFO)
- Cancellation Flexibility: Cancel requests can execute out-of-order to prevent deadlock
- State Consistency: Guardian/threshold changes apply predictably to future requests
- Automatic Progression: System automatically advances to next executable request

Benefits:

- Eliminates race conditions between configuration changes and pending actions
- Provides predictable execution order for complex multi-step operations
- Prevents system deadlock through flexible cancellation mechanism

### 4. Deadlock Prevention Mathematics

Critical Discovery: The submitter/approver separation introduced potential deadlock scenarios in N-of-N configurations.

Deadlock Scenario Example:

Configuration: 3 guardians, 3-of-3 threshold

Action: Guardian A submits recovery request

Problem: Guardian A cannot approve (submitter restriction)

Result: Maximum possible approvals = 2 (Guardians B + C)

Outcome: 2-of-3  $\neq$  3-of-3  $\rightarrow$  Permanent deadlock



Mathematical Solution: Enforce maximum threshold of N-1 to guarantee executability:

#### Threshold Matrix:

Guardian Count	Allowed Thresholds	Blocked Threshold	Reasoning
1 guardian	0	1-of-1	Submitter cannot self-approve
2 guardians	0, 1	2-of-2	Max 1 approver available
3 guardians	0, 1, 2	3-of-3	Max 2 approvers available
N guardians	0 to N-1	N-of-N	Max N-1 approvers available

Implementation:

```
function _setDefaultGuardianThreshold(uint8 threshold) internal {
    if (guardians.length == 0) revert InsufficientGuardians();

    // Prevent mathematical deadlock: max threshold = N-1
    if (threshold > uint8(guardians.length - 1))
        revert ThresholdWouldCauseDeadlock();

    defaultGuardianThreshold = threshold;
}
```

## 5. Enhanced Security Model

Comprehensive Improvements:

- Separation of Duties: Submitters and approvers have distinct, non-overlapping roles
- Deadlock Immunity: Mathematical guarantees prevent system lockup
- Predictable Execution: Sequential ordering eliminates race conditions
- Flexible Operations: Zero thresholds enable immediate execution when appropriate
- Cost Efficiency: Factory pattern reduces deployment costs by 90%

Security Benefits:

- Prevents single-point-of-failure scenarios
- Eliminates self-approval attack vectors
- Ensures system remains operational under all configurations
- Provides clear audit trail with predictable execution order
- Maintains backward compatibility while enhancing security

These architectural improvements transform the GlobalGold system from a basic multi-signature implementation into an enterprise-grade vault management platform with mathematical security guarantees and significant operational efficiencies.



# Functional Tests

Some of the tests performed are mentioned below:

- ✓ Initial owner should be set correctly
- ✓ Proxy state should be enabled for initializer caller
- ✓ Treasury and GlobalGoldVaultNFT addresses should be set correctly on initialize
- ✓ Initialize should revert on zero treasury or zero NFT address
- ✓ Only owner can update GlobalGoldVaultNFT address
- ✓ Only owner can update treasury address
- ✓ Updating GlobalGoldVaultNFT should store new address correctly
- ✓ Updating treasury should store new address correctly
- ✓ Only proxy can add assets
- ✓ Adding asset should emit AssetAdded event
- ✓ Adding duplicate asset should revert with AssetAlreadyExists
- ✓ Only proxy can remove assets
- ✓ Removing non-existing asset should revert with NotAllowed
- ✓ Removing asset should clear it from allowedAssets mapping
- ✓ allowedAsset should return correct boolean for allowed and disallowed assets
- ✓ getVaultsForAddress should return correct vaults for user-owned NFTs
- ✓ ownerOf should return correct owner from GlobalGoldVaultNFT
- ✓ safeMint should revert if GlobalGoldVaultNFT not set
- ✓ safeMint should mint NFT and assign vault correctly
- ✓ safeMint should revert with GlobalGoldVaultMintFailed if mint fails
- ✓ safeMint should map tokenId to new vault contract correctly
- ✓ Only proxy can call withdrawNativeToTreasury
- ✓ withdrawNativeToTreasury should transfer ETH balance to treasury
- ✓ Only proxy can call withdrawERC20ToTreasury
- ✓ withdrawERC20ToTreasury should transfer ERC20 tokens to treasury



- ✓ Initial owner should be set correctly
- ✓ Proxy state should be enabled for initializer caller
- ✓ BaseURI should be set correctly on initialize
- ✓ Initialize should revert if called again (initializer protected)
- ✓ Only proxy can call safeMint
- ✓ safeMint should mint NFT to correct owner with correct tokenId
- ✓ safeMint should revert if tokenId already exists
- ✓ setBaseURI should only be callable by proxy
- ✓ setBaseURI should update baseURI correctly
- ✓ Only owner can set vaultAdmin
- ✓ Setting vaultAdmin should revert on zero address
- ✓ Setting vaultAdmin should update vaultAdmin correctly
- ✓ tokenURI should return correct URI when baseURI is set
- ✓ tokenURI should return empty string if baseURI is empty
- ✓ tokenURI should revert with TokenNotFound if token does not exist
- ✓ getOwnedTokenIds should return all tokenIds owned by user
- ✓ RequestRecovery should only be callable by correct vault
- ✓ RequestRecovery should revert if requestId already exists
- ✓ RequestRecovery should store recovery request correctly
- ✓ RequestRecovery should map guardianRequestId to recovery requestId
- ✓ RequestRecovery should emit RecoveryRequested event
- ✓ CancelRecoveryFromGuardian should revert if request not found
- ✓ CancelRecoveryFromGuardian should revert if not called by requesting vault
- ✓ CancelRecoveryFromGuardian should revert if already executed
- ✓ CancelRecoveryFromGuardian should revert if already cancelled
- ✓ CancelRecoveryFromGuardian should set cancelled flag and emit event
- ✓ ExecuteRecovery should revert if already executed
- ✓ ExecuteRecovery should revert if already cancelled
- ✓ ExecuteRecovery should revert if recovery delay not met



- ✓ ExecuteRecovery should revert if not called by requesting vault
- ✓ ExecuteRecovery should revert if guardian request is cancelled
- ✓ ExecuteRecovery should transfer ownership of token to new owner
- ✓ ExecuteRecovery should mark request as executed and emit event
- ✓ supportedInterface should return true for ERC721 and ERC721Enumerable
- ✓ \_increaseBalance override should update balances correctly
- ✓ \_update override should update ownership correctly
- ✓ Should deposit ERC1155 tokens and emit DepositERC1155
- ✓ Should revert if caller is not vaultId holder
- ✓ Should revert if asset is not allowed
- ✓ Should revert if vault is locked
- ✓ Should revert if amount is zero
- ✓ Should transfer tokens from user to vault
- ✓ Should deposit ERC20 tokens and emit DepositERC20
- ✓ Should revert if caller is not vaultId holder
- ✓ Should revert if asset is not allowed
- ✓ Should revert if vault is locked
- ✓ Should revert if amount is zero
- ✓ Should revert if user has insufficient balance
- ✓ Should transfer tokens from user to vault
- ✓ Should deposit ERC721 tokens and emit DepositERC721
- ✓ Should revert if caller is not vaultId holder
- ✓ Should revert if asset is not allowed
- ✓ Should revert if vault is locked
- ✓ Should revert if array length is zero
- ✓ Should revert if array length exceeds MAX\_TOKEN\_ARRAY\_LENGTH
- ✓ Should revert if user has insufficient balance
- ✓ Should transfer all specified tokens from user to vault



# Threat Model

Contract	Function	Threats
GlobalGoldVaultAdmin	initialize	Anyone can call initialize and take ownership if not properly restricted.
	addAsset	Malicious proxy can add arbitrary assets, including fake ERC20/ERC721 contracts.
	removeAsset	Allowed assets could be removed, breaking vault functionality.
	safeMint	NFT minting could fail if NFT contract misbehaves.
	setGlobalGoldVaultNFT	Malicious contract could be set as NFT registry.
	setTreasury	Treasury could be set to attacker address.
	resolve	Consensus position calculation errors
	withdrawNativeToTreasury	Treasury drain if proxy/owner compromised.

## Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.



# Closing Summary

In this report, we have considered the security of GlobalGold. We performed our audit according to the procedure described above.

issues of Critical, High, Medium and Low severity were found. At the end Global Gold Team Acknowledged two issues and resolved the others.

# Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.





# About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers.

With seven years of expertise, we've secured over 1400 projects globally, averting over \$3 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

**7+**

Years of Expertise

**1M+**

Lines of Code Audited

**50+**

Chains Supported

**1400+**

Projects Secured

Follow Our Journey



# AUDIT REPORT

---

October 2025

For



Global Gold



Canada, India, Singapore, UAE, UK

[www.quillaudits.com](http://www.quillaudits.com)

[audits@quillaudits.com](mailto:audits@quillaudits.com)