CERTIK

Goldfish Gold

Security Assessment

CertiK Assessed on Apr 7th, 2026

CertiK Assessed on Apr 7th, 2026

Goldfish Gold

The security assessment was prepared by CertiK.

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Ethereum (ETH)	Manual Review, Static Analysis

LANGUAGE	TIMELINE
Solidity	Preliminary comments published on 02/05/2026
	Final report published on 04/07/2026

Vulnerability Summary

16	12	2	0	2	0
Total Findings	Resolved	Multi-Sig	Partially Resolved	Acknowledged	Declined

3	Centralization	2 Multi-Sig, 1 Acknowledged	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
0	Major		Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
5	Medium	5 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
2	Minor	2 Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
6	Informational	5 Resolved, 1 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | GOLDFISH GOLD

AUDIT SCOPE | GOLDFISH GOLD

ardata-tech/goldfish-smart-contracts
contracts/GoldfishToken.sol
contracts/GoldfishTokenFactory.sol
contracts/Redemption.sol
contracts/KYCRegistry.sol

APPROACH & METHODS | GOLDFISH GOLD

This audit was conducted for Goldfish Gold to evaluate the security and correctness of the smart contracts associated with the Goldfish Gold project. The assessment included a comprehensive review of the in-scope smart contracts. The audit was performed using a combination of Manual Review and Static Analysis.

The review process emphasized the following areas:

- Architecture review and threat modeling to understand systemic risks and identify design-level flaws.
- Identification of vulnerabilities through both common and edge-case attack vectors.
- Manual verification of contract logic to ensure alignment with intended design and business requirements.
- Dynamic testing to validate runtime behavior and assess execution risks.
- Assessment of code quality and maintainability, including adherence to current best practices and industry standards.

The audit resulted in findings categorized across multiple severity levels, from informational to critical. To enhance the project's security and long-term robustness, we recommend addressing the identified issues and considering the following general improvements:

- Improve code readability and maintainability by adopting a clean architectural pattern and modular design.
- Strengthen testing coverage, including unit and integration tests for key functionalities and edge cases.
- Maintain meaningful inline comments and documentations.
- Implement clear and transparent documentation for privileged roles and sensitive protocol operations.
- Regularly review and simulate contract behavior against newly emerging attack vectors.

REVIEW NOTES | GOLDFISH GOLD

Overview

- Upgradeable `GoldfishToken` (`ERC20BurnableUpgradeable` + `AccessControl`) centralizes issuance through an authorized minter contract, while enforcing compliance controls such as pausing, address freezing/blacklisting, batch transfers, and admin force transfers; supply is capped and tracked via `totalMinted`/`totalBurned`.
- `GoldfishTokenFactory` is the sole minter/burner, routing base issuance to a treasury wallet and optionally minting configurable fee slices to multiple wallets; it also records issuance/burn statistics and restricts redemption burns to an authorized `Redemption` contract.
- `Redemption` orchestrates a custody-like flow where user deposits are locked in-contract, pass through a Pending → Approved → Processing → Completed/Rejection state machine, optionally deduct a basis-point fee to a designated house wallet, and ultimately burn tokens through the factory once delivery is confirmed.
- `KYCRegistry` provides the single source of truth for compliance checks; redemption requests validate against it before accepting deposits.
- All components follow the UUPS pattern and share a role-gated administrative surface, so upgrade authority and operational roles (mint, burn, redemption admin, KYC admin) are separated but centrally managed.

External Dependencies

- OpenZeppelin upgradeable modules: ERC20 Burnable, AccessControl, Pausable, ReentrancyGuard, UUPS, Initializable.
- `SafeERC20` wrappers for interacting with the GGBR token and fee transfers, assuming ERC20-compliant behavior with standard allowances.
- Internal interfaces (`IGoldfishToken`, `IGoldfishTokenFactory`, `IKYCRegistry`) define the cross-contract expectations; correctness relies on the factory retaining mint/burn authority and the registry being a trusted attestation source.

Privileged Functions

In the project, multiple roles are adopted to ensure the dynamic runtime updates of the project, which were specified in the centralized findings.

- `GoldfishToken`: `setMinterContract` (assigns the sole MINTER), `setMaxSupply` (SUPPLY_MANAGER cap control), `pause`/`unpause` (PAUSER), `freezeAccount`/`batchFreezeAccounts` (FREEZER), `setBlacklisted`/`batchBlacklistAccounts` (DEFAULT_ADMIN), `forceTransfer` (DEFAULT_ADMIN override), plus `_authorizeUpgrade`.
- `GoldfishTokenFactory`: `setTreasury`, `authorizeContracts`, `add/update/remove/activate/deactivateFeeWallet` (DEFAULT_ADMIN treasury/fee configuration), `issueTokens` (MINT_OPERATOR-controlled minting with fee fan-out), `burnTokens` (BURN_OPERATOR manual burns), `pause`/`unpause`, and `_authorizeUpgrade`.
- `KYCRegistry`: `addVerifiedKYCUser`/`removeVerifiedKYCUser` (KYC_ADMIN), `pause`/`unpause`, and `_authorizeUpgrade` (DEFAULT_ADMIN).

- `Redemption`: `updateKYCRegistry`, `setMinimumRedemption`, `setRedemptionFeeRate`, `setHouseWallet`, `pause`/`unpause`, `emergencyWithdraw` (DEFAULT_ADMIN configuration/custody control); `approveRedemption`, `processRedemption`, `completeRedemption`, `rejectRedemption` (REDEMPTION_ADMIN workflow execution); `_authorizeUpgrade` limits implementation changes.

The advantage of this privileged role in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community. It is also worth of note the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the private key of the privileged account is compromised, it could lead to devastating consequences for the project.

FINDINGS | GOLDFISH GOLD

16	0	3	0	5	2	6
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared for Goldfish Gold to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the audit, a total of 16 issues were identified. Leveraging a combination of Manual Review & Static Analysis the following findings were uncovered:

ID	Title	Category	Severity	Status
GOG-02	**Centralized Control Of Contract Upgrade**	**Centralization**	**Centralization**	**2/3 Multi-Sig**
GOG-03	**Centralized Balance Manipulation**	**Centralization**	**Centralization**	**Acknowledged**
GOG-04	**Centralization Risks**	**Centralization**	**Centralization**	**2/3 Multi-Sig**
GOG-05	Freeze/Blacklist Checks In `_update` Also Block Admin Compliance Actions In `forceTransfer`	Design Issue	Medium	Resolved
GOG-06	Unenforced `maxSupply` In `GoldfishToken` Minting Operations	Logical Issue	Medium	Resolved
GOG-07	Inherited `burnFrom()` Bypasses Intended Burn Authorization And Breaks `totalBurned/TokensBurned` Accounting	Inconsistency	Medium	Resolved
GOG-08	Self-Transfer In Batch Operations Leads To Unnecessary Gas Consumption And Potential Logic Errors	Coding Issue	Medium	Resolved
GOG-17	Hidden `MINTER_ROLE` Holders Can Bypass The Intended `minterContract` Issuance Flow	Access Control	Medium	Resolved
GOG-09	No Cap On Fees	Logical Issue	Minor	Resolved

ID	Title	Category	Severity	Status
GOG-11	Duplicate Fee Wallet Addition Leads To Array Pollution And Gas Inefficiency-Exported	Design Issue	Minor	Resolved
GOG-10	Missing Pause State Check In View Function Leads To Inconsistent Information	Coding Issue	Informational	Resolved
GOG-12	PII/Financial Information Leakage	Volatile Code	Informational	Acknowledged
GOG-13	Lack Of Persisting `processedBy` In `processRedemption`	Logical Issue	Informational	Resolved
GOG-14	Redundant Approval In `completeRedemption`	Logical Issue	Informational	Resolved
GOG-15	Unused Event Declaration Leads To Code Bloat And Maintenance Confusion-Exported	Code Optimization	Informational	Resolved
GOG-16	Missing `whenNotPaused` Modifier In `updateDeliveryDetails`	Inconsistency	Informational	Resolved

GOG-02 | Centralized Control Of Contract Upgrade

Category	Severity	Location	Status
Centralization	Centralization	contracts/GoldfishToken.sol (4a4163e69634397a066245a3384a81e16faa369e): 15; contracts/GoldfishTokenFactory.sol (4a4163e69634397a066245a3384a81e16faa369e): 16; contracts/KYCRegistry.sol (4a4163e69634397a066245a3384a81e16faa369e): 14; contracts/Redemption.sol (4a4163e69634397a066245a3384a81e16faa369e): 27	2/3 Multi-Sig

Description

In the contract `GoldfishToken`, the role `DEFAULT_ADMIN_ROLE` has the authority to update the implementation contract behind the proxy contract.

```
 88  function _authorizeUpgrade(address newImplementation) internal override
onlyRole(DEFAULT_ADMIN_ROLE) {}
```

In the contract `GoldfishTokenFactory`, the role `DEFAULT_ADMIN_ROLE` has the authority to update the implementation contract behind the proxy contract.

```
126  function _authorizeUpgrade(address newImplementation) internal override
onlyRole(DEFAULT_ADMIN_ROLE) {}
```

In the contract `KYCRegistry`, the role `DEFAULT_ADMIN_ROLE` has the authority to update the implementation contract behind the proxy contract.

```
57  function _authorizeUpgrade(address newImplementation) internal override onlyRole
(DEFAULT_ADMIN_ROLE) {}
```

In the contract `Redemption`, the role `DEFAULT_ADMIN_ROLE` has the authority to update the implementation contract behind the proxy contract.

```
160  function _authorizeUpgrade(address newImplementation) internal override
onlyRole(DEFAULT_ADMIN_ROLE) {}
```

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow a hacker to take advantage of this authority and change the implementation contract which is pointed by proxy and therefore execute potential malicious functionality in the implementation contract.

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

Short Term:

A combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.

- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

- Renounce the ownership and never claim back the privileged role;
  OR
- Remove the risky functionality.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[Goldfish Gold, 03/23/2026]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

> Implement the time-lock and multi sig: We've implemented Gnosis Safe as DEFAULT_ADMIN_ROLE approach with admin UI adjustments.

[CertiK, 03/23/2026]: In order for CertiK to update the status of this finding, please kindly provide the multi-signers addresses and we will verify the information and update the report:

Multi-sig wallet address: 0x…

Signer 1: 0x…

Signer 2: 0x…

Signer 3: 0x…

[Goldfish Gold, 04/01/2026]: The team acknowledged the issue and adopted the multi-sig solution at the current stage. The GoldfishToken has transferred the `DEFAULT_ADMIN_ROLE` role to a Gnosis in transaction.

The GoldfishTokenFactory has transferred the `DEFAULT_ADMIN_ROLE` role to a Gnosis in transaction.

The Redemption has transferred the `DEFAULT_ADMIN_ROLE` role to a Gnosis in transaction.

The KYCRegistry has transferred the `DEFAULT_ADMIN_ROLE` role to a Gnosis in transaction.

- Gnosis safe contract address: https://etherscan.io/address/0xd0022c3eD6A11BF6791BA2C9d0b394021a6096e4#code
- The 2/3 multisign addresses:
  - 0x69e91940154bbdbabca6ca0b0ec3cb2a59aebf2b
  - 0xdfbb0b971a9f1915a083d5555ce20a36dd0b7aeb

- 0x19a1fd04761c9a0b873bec440caf02ab0dba056e

GOG-03 | Centralized Balance Manipulation

Category	Severity	Location	Status
Centralization	**Centralization**	**contracts/GoldfishToken.sol (4a4163e69634397a066245a3384a81e16faa369e): 181, 197; contracts/GoldfishTokenFactory.sol (4a4163e69634397a066245a3384a81e16faa369e): 292**	**Acknowledged**

Description

In the contract `GoldfishToken`, the role `MINTER_ROLE` and `minterContract` has the authority to update the token balance of an arbitrary account without sanity restriction.

Any compromise to the role `MINTER_ROLE` and `minterContract` account may allow a hacker to take advantage of this authority and manipulate users' balances.

Recommendation

We recommend the team makes efforts to restrict access to the private key of the privileged account. A strategy of multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to mint more tokens or engage in similar balance-related operations.
Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently *fully* resolve the risk:

Short Term:

A multi signature (⅔, ⅗) wallet *mitigate* the risk by avoiding a single point of key management failure.

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers' addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A DAO for controlling the operation *mitigate* the risk by applying transparency and decentralization.

- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the multi-signers' addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

The following actions can *fully* resolve the risk:

- Renounce the ownership and never claim back the privileged role.
  OR
- Remove the risky functionality.
  OR
- Add minting logic (such as a vesting schedule) to the contract instead of allowing the owner account to call the sensitive function directly.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

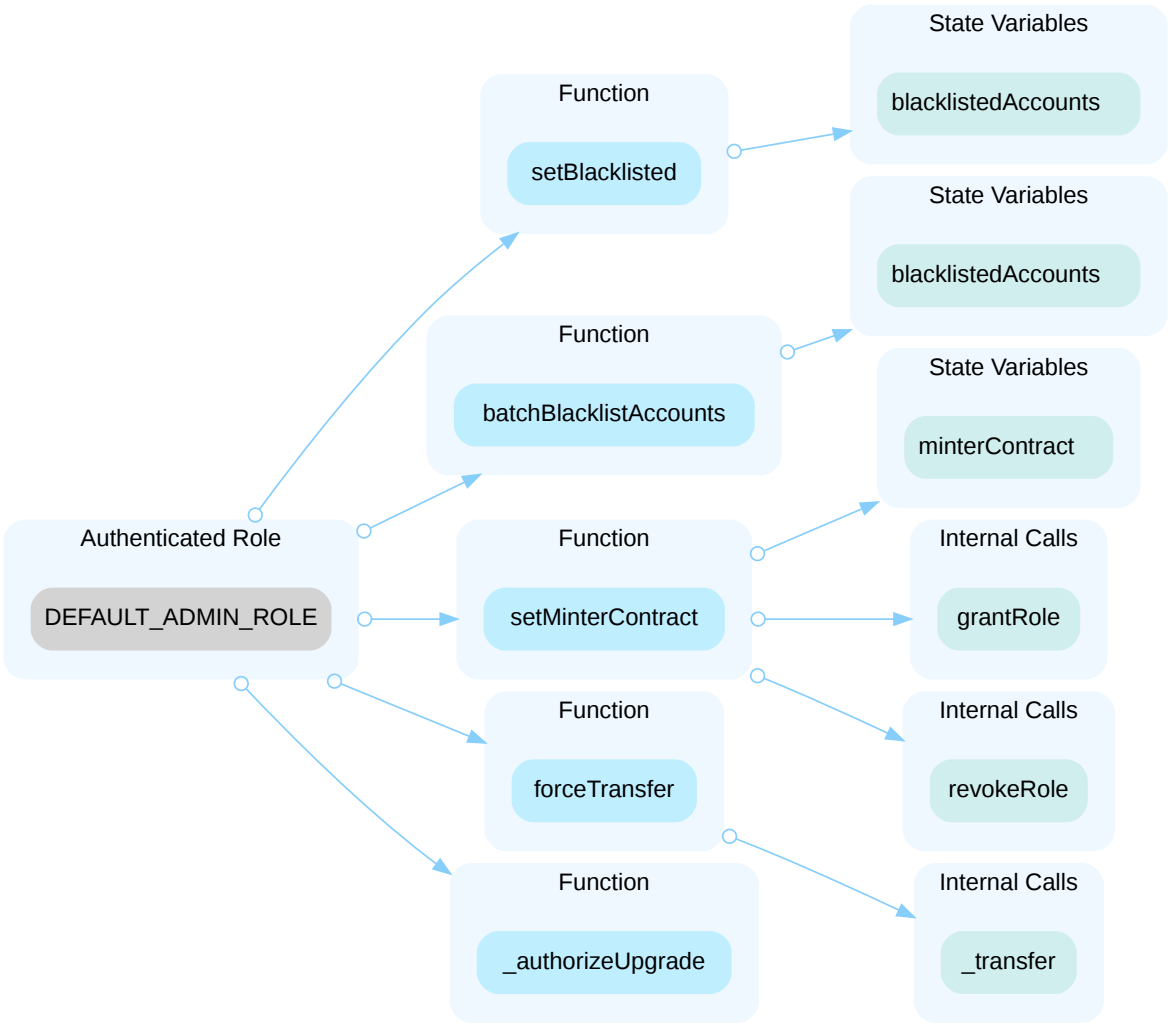
[Goldfish Gold, 03/23/2026]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

GOG-04 | Centralization Risks

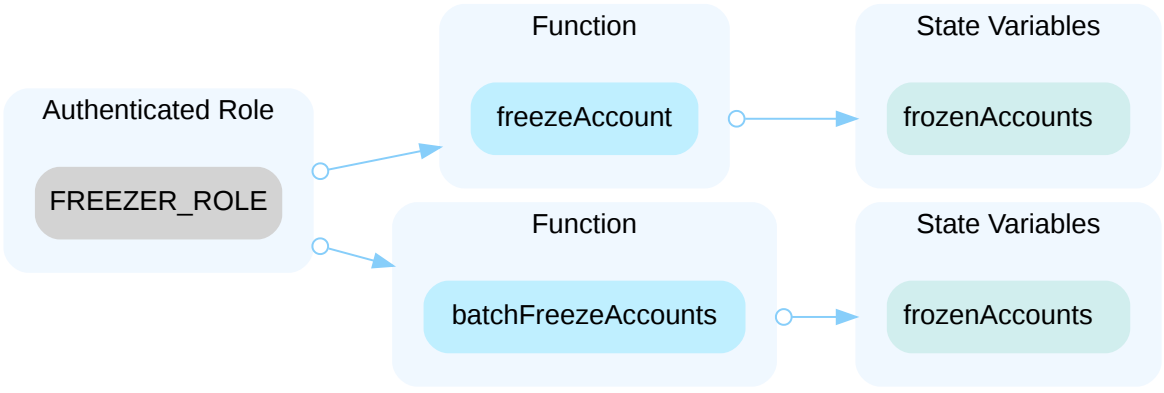
Category	Severity	Location	Status
Centralization	Centralization	contracts/GoldfishToken.sol (4a4163e69634397a066245a3384a81e16faa369e): 88, 93, 112, 124, 131, 140, 148, 158, 166, 181, 197, 218; contracts/GoldfishTokenFactory.sol (4a4163e69634397a066245a3384a81e16faa369e): 126, 131, 139, 155, 175, 188, 208, 219, 232, 292, 398, 402; contracts/KYCRegistry.sol (4a4163e69634397a066245a3384a81e16faa369e): 57, 65, 76, 108, 115; contracts/Redemption.sol (4a4163e69634397a066245a3384a81e16faa369e): 160, 166, 179, 192, 205, 265, 282, 312, 359, 393, 576, 583, 592	2/3 Multi-Sig

Description

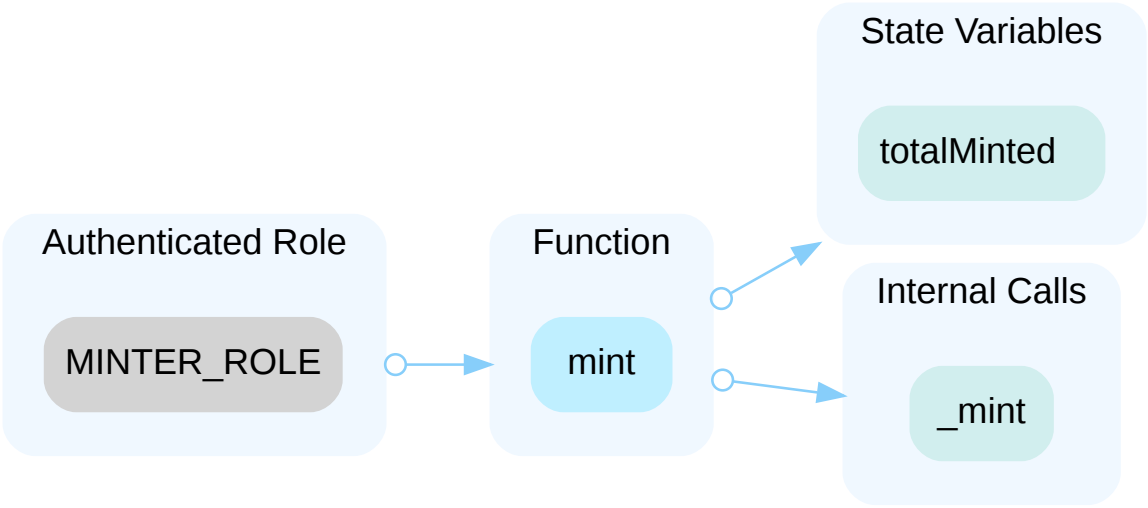
In the contract `GoldfishToken`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and have unspecified privileges.



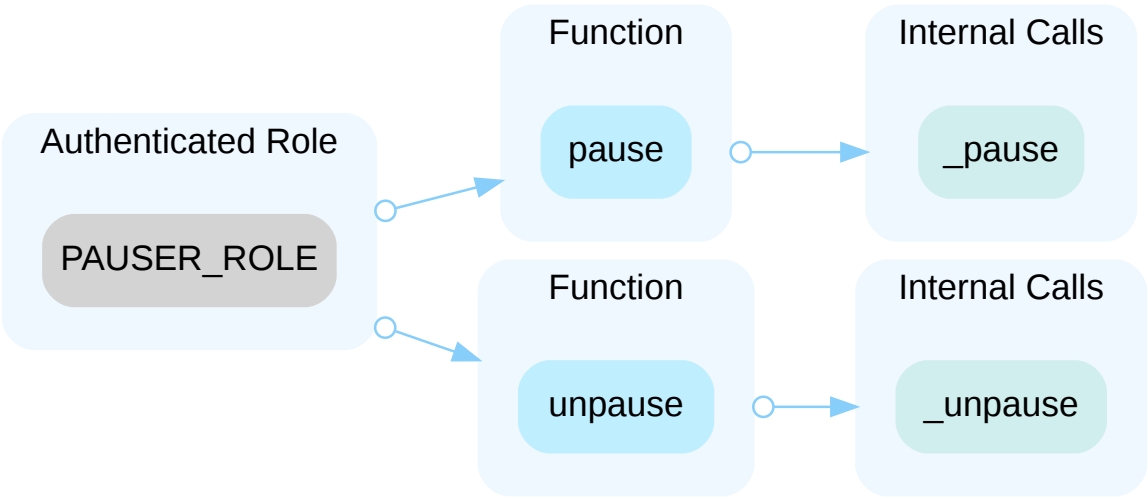
In the contract `GoldfishToken`, the role `FREEZER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `FREEZER_ROLE` account may allow the hacker to take advantage of this authority.



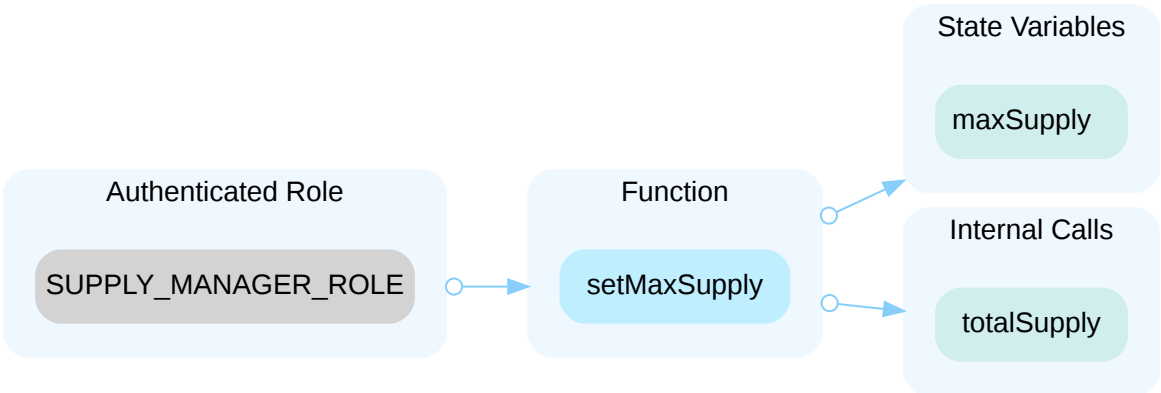
In the contract `GoldfishToken`, the role `MINTER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `MINTER_ROLE` account may allow the hacker to take advantage of this authority and exercise any associated minting privileges.



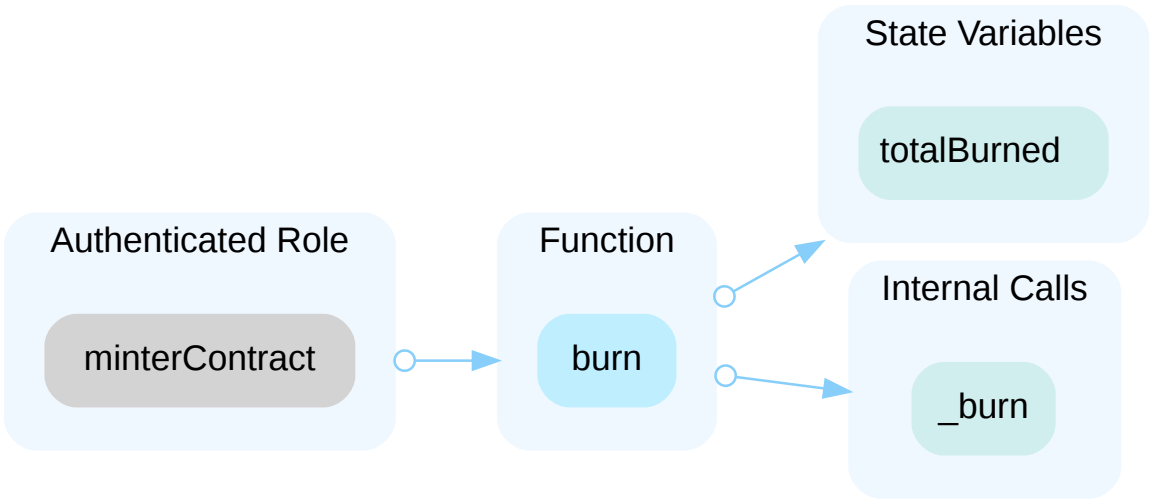
In the contract `GoldfishToken`, the role `PAUSER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `PAUSER_ROLE` account may allow the hacker to take advantage of this authority.



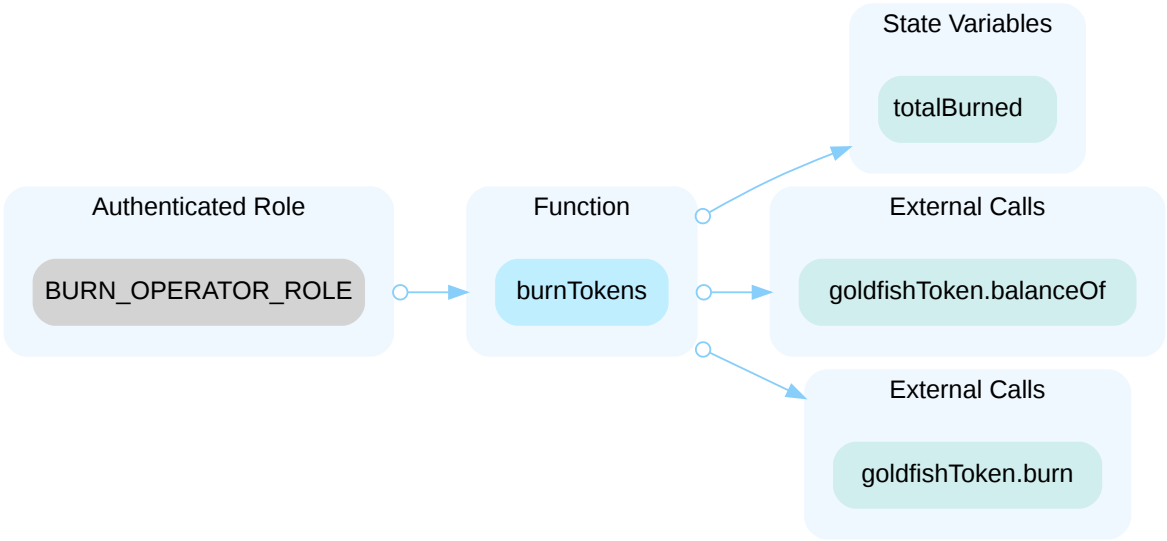
In the contract `GoldfishToken`, the role `SUPPLY_MANAGER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `SUPPLY_MANAGER_ROLE` account may allow the hacker to take advantage of this authority.



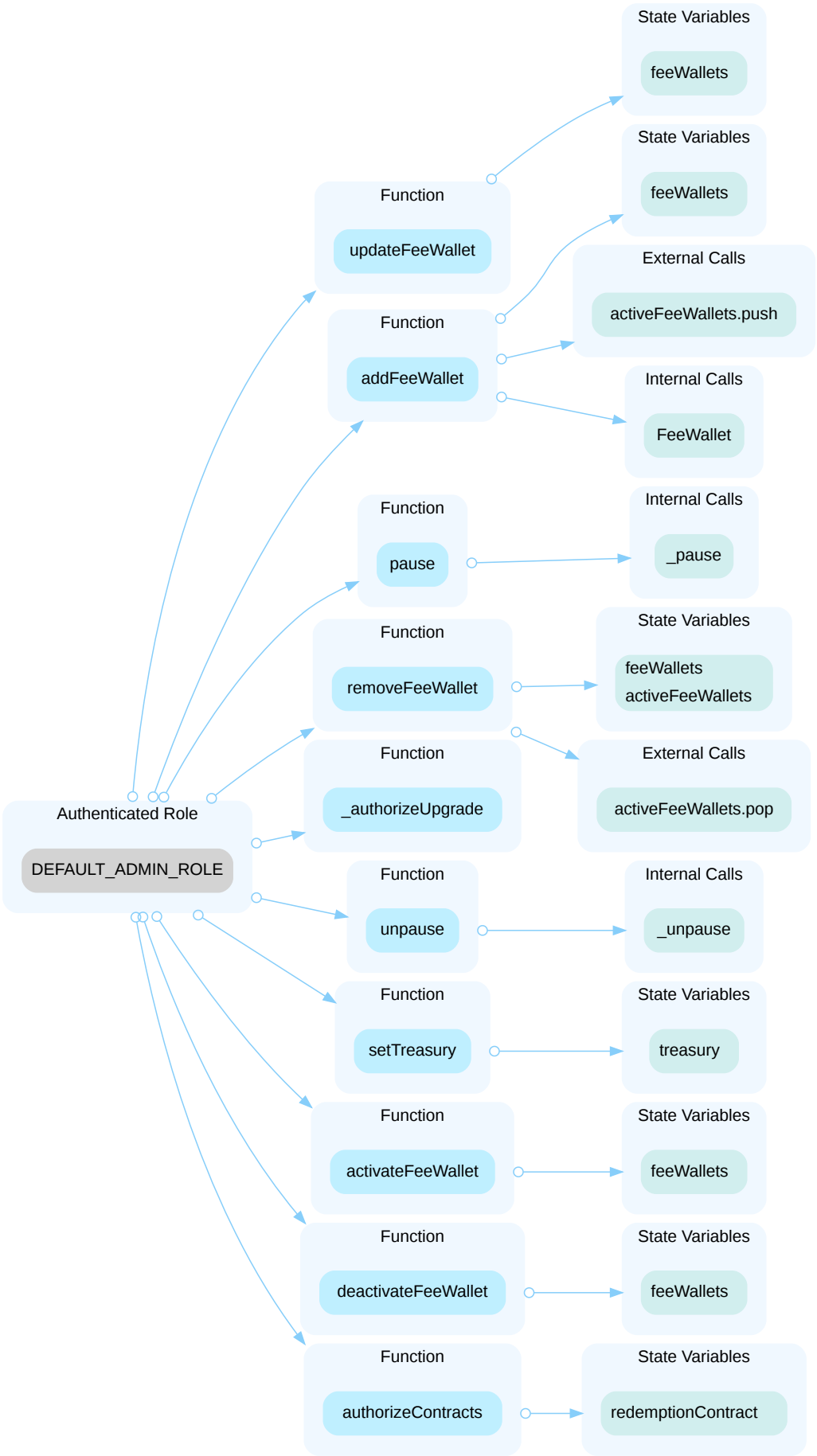
In the contract `GoldfishToken`, the role `minterContract` has authority over the functions shown in the diagram below. Any compromise to the `minterContract` account may allow the hacker to take advantage of this authority.



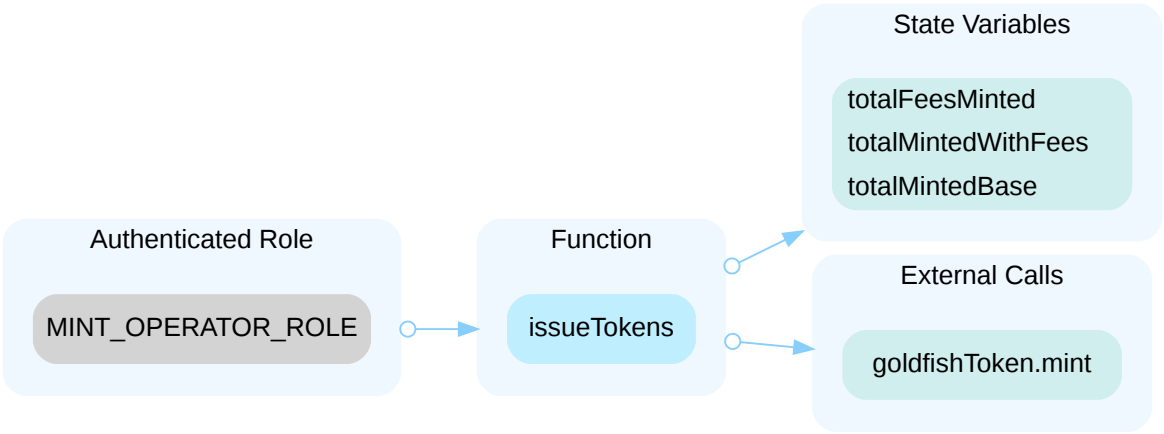
In the contract `GoldfishTokenFactory`, the role `BURN_OPERATOR_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `BURN_OPERATOR_ROLE` account may allow the hacker to take advantage of this authority and exercise relevant privileges.



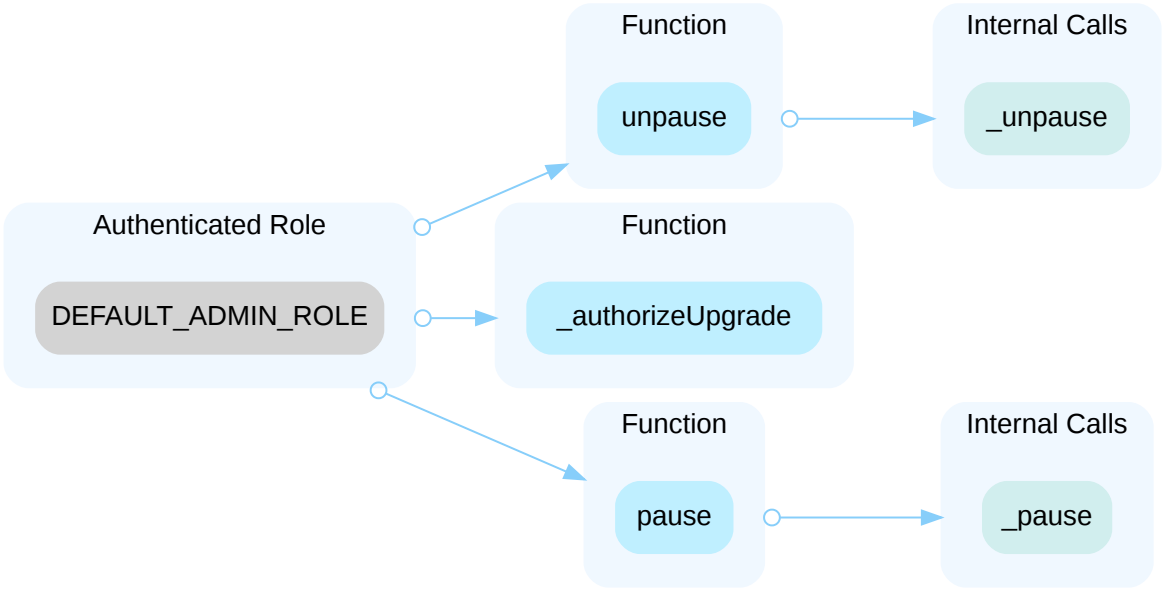
In the contract `GoldfishTokenFactory`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and assume control or improperly manage the administrative functions.


State Variables
feeWallets
State Variables
feeWallets
Function
updateFeeWallet
External Calls
activeFeeWallets.push
Function
addFeeWallet
Internal Calls
FeeWallet
Function
pause
Internal Calls
_pause
Function
removeFeeWallet
State Variables
feeWallets
activeFeeWallets
Function
_authorizeUpgrade
External Calls
activeFeeWallets.pop
Authenticated Role
DEFAULT_ADMIN_ROLE
Function
unpause
Internal Calls
_unpause
Function
setTreasury
State Variables
treasury
Function
activateFeeWallet
State Variables
feeWallets
Function
deactivateFeeWallet
State Variables
feeWallets
Function
authorizeContracts
State Variables
redemptionContract

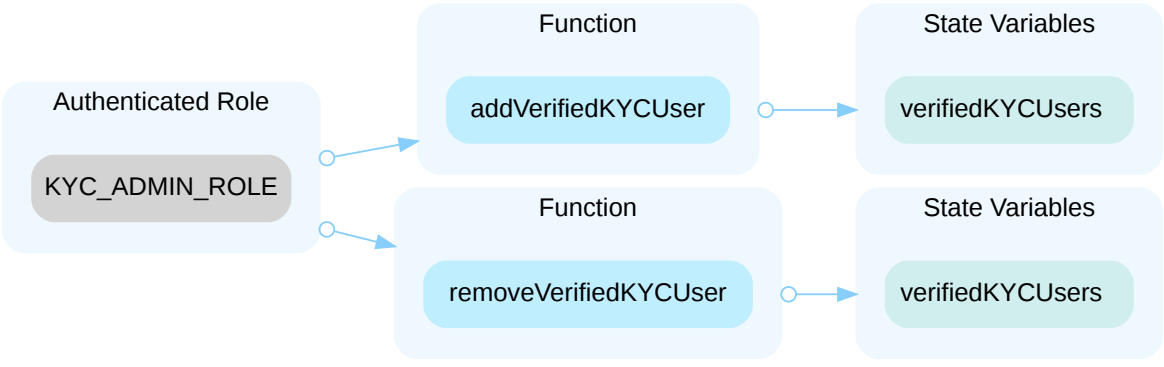
In the contract `GoldfishTokenFactory`, the role `MINT_OPERATOR_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `MINT_OPERATOR_ROLE` account may allow the hacker to take advantage of this authority.



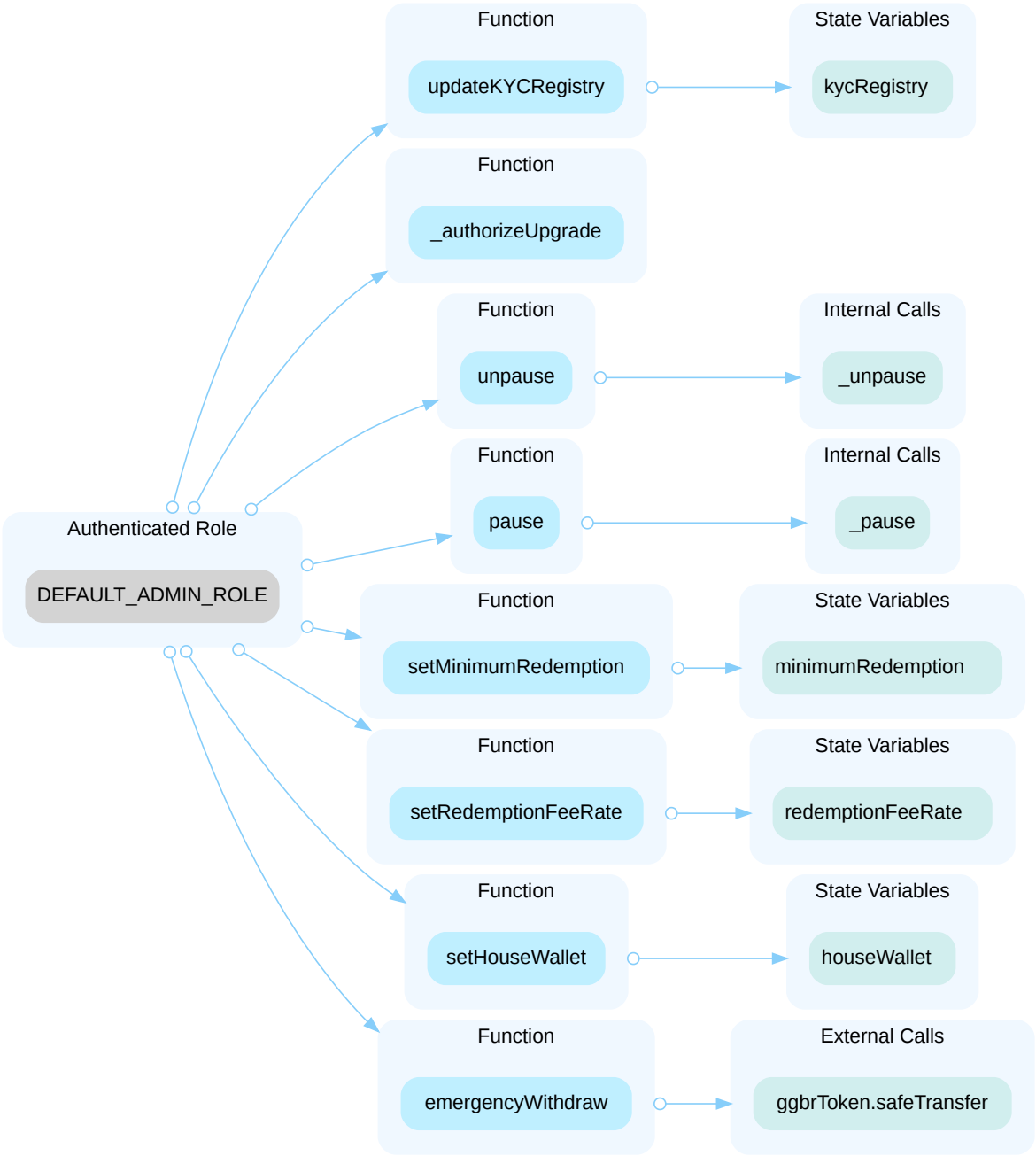
In the contract `KYCRegistry`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and exercise privileges associated with it.



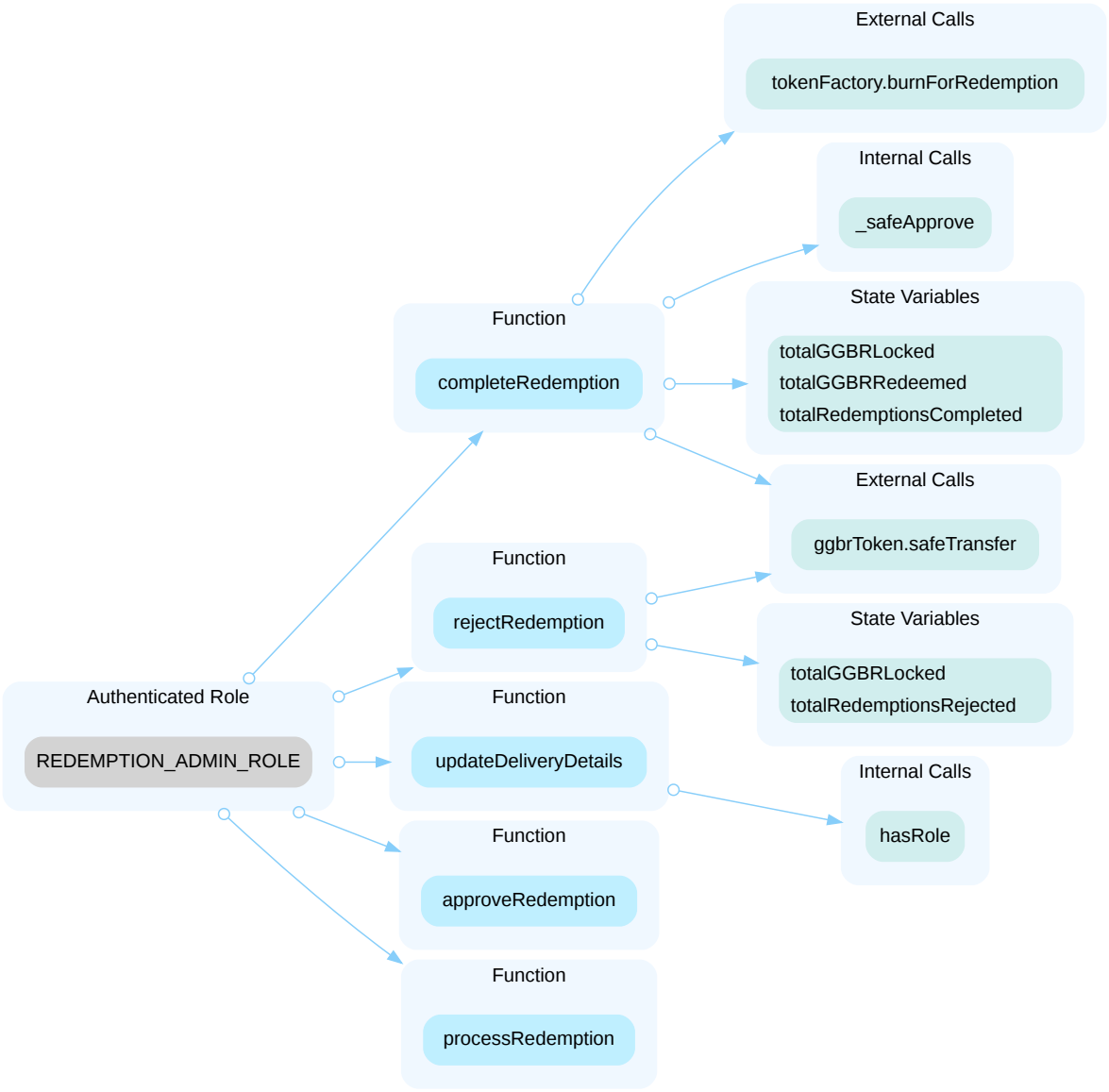
In the contract `KYCRegistry`, the role `KYC_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `KYC_ADMIN_ROLE` account may allow the hacker to take advantage of this authority.



In the contract `Redemption`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority.



In the contract `Redemption`, the role `REDEMPTION_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `REDEMPTION_ADMIN_ROLE` account may allow the hacker to take advantage of this authority.



Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Goldfish Gold, 03/23/2026]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

[Goldfish Gold, 04/01/2026]: The team acknowledged the issue and adopted the multi-sig solution at the current stage. The GoldfishToken has transferred the `DEFAULT_ADMIN_ROLE` role to a Gnosis in transaction.

The GoldfishToken has transferred the `FREEZER_ROLE` role to a Gnosis in transaction.

The GoldfishToken has transferred the `PAUSER_ROLE` role to a Gnosis in transaction.

The GoldfishToken has transferred the `SUPPLY_MANAGER_ROLE` role to a Gnosis in transaction.

The GoldfishTokenFactory has transferred the `DEFAULT_ADMIN_ROLE` role to a Gnosis in transaction.

The GoldfishTokenFactory has transferred the `BURN_OPERATOR_ROLE` role to a Gnosis in transaction.

The GoldfishTokenFactory has transferred the `MINT_OPERATOR_ROLE` role to a Gnosis in transaction.

The Redemption has transferred the `DEFAULT_ADMIN_ROLE` role to a Gnosis in transaction.

The Redemption has transferred the `REDEMPTION_ADMIN_ROLE` role to a Gnosis in transaction.

The KYCRegistry has transferred the `DEFAULT_ADMIN_ROLE` role to a Gnosis in transaction.

- Gnosis safe contract address: https://etherscan.io/address/0xd0022c3eD6A11BF6791BA2C9d0b394021a6096e4#code
- The 2/3 multisign addresses:
  - 0x69e91940154bbdbabca6ca0b0ec3cb2a59aebf2b
  - 0xdfbb0b971a9f1915a083d5555ce20a36dd0b7aeb
  - 0x19a1fd04761c9a0b873bec440caf02ab0dba056e

GOG-05 Freeze/Blacklist Checks In `_update` Also Block Admin Compliance Actions In `forceTransfer`

Category	Severity	Location	Status
Design Issue	Medium	contracts/GoldfishToken.sol (4a4163e69634397a066245a3384a81e16faa369e): 218~227, 268~278	Resolved

Description

The `forceTransfer()` function in `GoldfishToken` is designed for compliance purposes (e.g., regulatory seizure or remediation), but its effectiveness is undermined by unconditional checks in `_update()`. Since `forceTransfer()` relies on `_transfer()`, which triggers `_update()`, frozen or blacklisted accounts cannot be force-transferred—even by admins. This creates a critical contradiction: a compliance feature that fails when compliance actions are most needed.

The current implementation:

```
218  function forceTransfer(
219          address from,
220          address to,
221          uint256 amount,
222          string calldata reason
223      ) external onlyRole(DEFAULT_ADMIN_ROLE) {
224          require(from != address(0) && to != address(0), "Invalid addresses");
225          _transfer(from, to, amount);
226          emit ForceTransfer(from, to, amount, reason);
227      }
```

it calls `_transfer(from, to, amount)`. The `_transfer()` invokes `_update()`, which enforces:

```
268  function _update(address from, address to, uint256 amount) internal
override whenNotPaused {
269          // Check frozen accounts
270          require(!frozenAccounts[from], "Sender frozen");
271          require(!frozenAccounts[to], "Recipient frozen");
272
273          // Check blacklist
274          require(!blacklistedAccounts[from], "Sender blacklisted");
275          require(!blacklistedAccounts[to], "Recipient blacklisted");
276
277          super._update(from, to, amount);
278      }
```

As a result, if an account is frozen/blacklisted, `forceTransfer()` reverts, preventing admins from moving funds for compliance reasons.

Recommendation

Consider redesigning the logic of `forceTransfer` function.

Alleviation

[Goldfish Gold, 03/23/2026]: The team heeded the advice and resolved the issue in commit 2e52a0c46fa9db7b26a3485109bf4fbbbe557b24.

GOG-06 | Unenforced `maxSupply` In `GoldfishToken` Minting Operations

Category	Severity	Location	Status
Logical Issue	Medium	contracts/GoldfishToken.sol (4a4163e69634397a066245a3384a81e16faa369e): 112~117	Resolved

Description

The `GoldfishToken` contract contains a serious economic control vulnerability where its documented maximum supply cap (`maxSupply`) is never enforced during token minting operations. Despite initializing and allowing updates to `maxSupply` , the contract fails to validate whether minting operations would exceed this limit. This oversight renders the supply cap purely decorative, allowing unlimited token issuance regardless of the configured `maxSupply` value.

`initialize()` sets a `maxSupply` hard cap (e.g., “1 billion tokens max”) and `setMaxSupply()` only updates the `maxSupply` state variable (only checking it isn’t set below `totalSupply()`), but no minting path enforces `totalSupply() + amount <= maxSupply` :

```
112  function setMaxSupply(uint256 _newCap) external onlyRole(SUPPLY_MANAGER_ROLE) {
113          require(_newCap >= totalSupply(), "Below current supply");
114          uint256 oldCap = maxSupply;
115          maxSupply = _newCap;
116          emit MaxSupplyUpdated(oldCap, _newCap);
117      }
```

Specifically, `GoldfishToken.mint()` unconditionally calls `_mint(to, amount)` :

```
181  function mint(address to, uint256 amount, string calldata memo) external
onlyRole(MINTER_ROLE) {
182          require(to != address(0), "Invalid recipient");
183
184          _mint(to, amount);
185          totalMinted += amount;
186
187          emit TokensMinted(to, amount, memo);
188      }
```

and the factory’s `issueTokens` (including any factory fee mints) likewise does not apply any cap invariant. As a result, the system can mint beyond the configured maximum supply even when operators act normally, making `setMaxSupply` ineffective and undermining the intended hard cap and any integrations that rely on it.

Recommendation

Enforce the maxSupply invariant on every mint path.

Alleviation

[Goldfish Gold, 03/31/2026]: The team heeded the advice and resolved the issue by adding the maxSupply check in `mint` in commit 843e93fcc6595bfd8ffc171e5849d8f4a5b0a1c4.

GOG-07 Inherited `burnFrom()` Bypasses Intended Burn Authorization And Breaks `totalBurned/TokensBurned` Accounting

Category	Severity	Location	Status
Inconsistency	Medium	contracts/GoldfishToken.sol (4a4163e69634397a066245a3384a81e16faa369e): 204~211	Resolved

Description

`GoldfishToken` inherits `ERC20BurnableUpgradeable`:

```
15 contract GoldfishToken is
16     Initializable,
17     ERC20BurnableUpgradeable,
18     PausableUpgradeable,
19     AccessControlUpgradeable,
20     UUPSUpgradeable
21 {
```

which exposes `burnFrom(address account, uint256 amount)`, but the contract only overrides `burn(uint256)` (self-burn) and adds a custom `burn(address,uint256)` restricted to `msg.sender == minterContract`, leaving burnFrom unmodified and callable:

```
197 function burn(address from, uint256 amount) external {
198         require(msg.sender == minterContract, "Not authorized");
199         _burn(from, amount);
200         totalBurned += amount;
201         emit TokensBurned(from, amount, "factory_burn");
202     }
203
204     /**
205      * @dev Burn own tokens
206      */
207     function burn(uint256 amount) public override {
208         super.burn(amount);
209         totalBurned += amount;
210         emit TokensBurned(msg.sender, amount, "self_burn");
211     }
```

Consequently, any spender with an allowance can call `burnFrom()` to irreversibly destroy the owner's tokens without being the `minterContract`, bypassing the apparent intent that third-party burns should occur only via the minter/factory path. Because `totalBurned` is incremented only in the overridden `burn(uint256)` and the custom `burn(address,uint256)`, `burnFrom` decreases `totalSupply` / `currentSupply` without updating `totalBurned` and without emitting the custom `TokensBurned` event, causing `getTokenStats` (`totalSupply`, `totalMinted`, `totalBurned`) and any off-

chain/accounting/audit/compliance/reporting processes relying on `totalBurned` or `TokensBurned` to become inconsistent and potentially manipulable.

Recommendation

The fix is to override `burnFrom()` to either disable/revert it if not intended, or enforce the same authorization model (e.g., restrict to `minterContract` /role) while also updating `totalBurned` and emitting `TokensBurned` consistently.

Alleviation

[Goldfish Gold, 03/23/2026]: The team heeded the advice and resolved the issue in commit 2e52a0c46fa9db7b26a3485109bf4fbbbe557b24.

GOG-08 Self-Transfer In Batch Operations Leads To Unnecessary Gas Consumption And Potential Logic Errors

Category	Severity	Location	Status
Coding Issue	Medium	contracts/GoldfishToken.sol (4a4163e69634397a066245a3384a81e16faa369e): 236~261	Resolved

Description

The `batchTransfer` function in `GoldfishToken.sol` allows users to include their own address as a recipient, which creates unnecessary gas consumption and potential confusion in transfer logic. The function is designed to send tokens to multiple recipients in a single transaction but lacks validation to prevent self-transfers.

When a user calls `batchTransfer` and includes their own address in the recipients array, the function will execute a transfer from the user to themselves. While this doesn't change the user's balance, it consumes gas unnecessarily and emits transfer events that could mislead external systems monitoring token movements. This could also interfere with accounting systems that track actual token distributions.

```
236     function batchTransfer(
237         address[] memory _recipients,
238         uint256[] memory _amounts
239     ) external {
240         require(_recipients.length == _amounts.length,
"Mismatched input arrays");
241         require(_recipients.length > 0, "Empty recipients array");
242 
243         uint256 totalAmount = 0;
244 
245         // Calculate the total amount of tokens to be transferred
246         for (uint256 i = 0; i < _amounts.length; i++) {
247             require(_recipients[i] != address(0), "Invalid recipient address");
248             require(_amounts[i] > 0, "Amount must be greater than zero");
249             totalAmount += _amounts[i];
250         }
251 
252         // Check if sender has sufficient balance
253         require(balanceOf(msg.sender) >= totalAmount, "Insufficient balance");
254 
255         // Perform transfers
256         for (uint256 i = 0; i < _recipients.length; i++) {
257             _transfer(msg.sender, _recipients[i], _amounts[i]);
258         }
259 
260         emit BatchTransfer(msg.sender, _recipients, _amounts);
261     }
```

Scenario

1. Alice has 1000 GGBR tokens and wants to distribute them to multiple recipients
2. Alice calls `batchTransfer` with recipients `[Bob, Alice, Charlie]` and amounts `[100, 200, 300]`
3. The function executes successfully, transferring 100 tokens to Bob, 200 tokens to Alice (herself), and 300 tokens to Charlie
4. Alice's balance remains 800 tokens (1000 - 100 - 300), but the transaction consumed extra gas for the self-transfer
5. External monitoring systems see a transfer event from Alice to Alice for 200 tokens, which could cause confusion in analytics

Recommendation

Add a validation check in the `batchTransfer` function to prevent self-transfers by ensuring `_recipients[i] != msg.sender` for all recipients in the array.

Alleviation

[Goldfish Gold, 03/23/2026]: The team heeded the advice and resolved the issue by checking the recipients in commit 2e52a0c46fa9db7b26a3485109bf4fbbbe557b24

GOG-17 Hidden `MINTER_ROLE` Holders Can Bypass The Intended `minterContract` Issuance Flow

Category	Severity	Location	Status
Access Control	Medium	contracts/GoldfishToken.sol (4a4163e69634397a066245a3384a81e16faa369e): 68~81	Resolved

Description

`GoldfishToken` appears to model `minterContract` as the single authorized minter, but `mint()` does not enforce that assumption. Instead, any address holding `MINTER_ROLE` may mint. Since `initialize()` grants `MINTER_ROLE` to the **deployer**, and `setMinterContract()` only revokes the role from the previous `minterContract`, other historical or forgotten role holders can retain mint privileges indefinitely. This creates a hidden mint path outside the intended factory-controlled issuance flow and can lead to unauthorized supply inflation.

The contract documentation and state naming indicate that minting is intended to be restricted to `minterContract`:

```
address public minterContract;      // Only this contract can mint
```

```
/**
 * @dev Mint new tokens (only minter contract)
 */
function mint(address to, uint256 amount, string calldata memo) external
onlyRole(MINTER_ROLE) {
    require(to != address(0), "Invalid recipient");

    _mint(to, amount);
    totalMinted += amount;

    emit TokensMinted(to, amount, memo);
}
```

However, the actual authorization check is `onlyRole(MINTER_ROLE)`, not `msg.sender == minterContract`. As a result, mint authority is role-based, not contract-address-based.

The issue is amplified by initialization and minter rotation logic:

```
function initialize() initializer public {
    ...
    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _grantRole(MINTER_ROLE, msg.sender);
    ...
}
```

```
function setMinterContract(address _minter) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(_minter != address(0), "Invalid address");

    if (minterContract != address(0)) {
        revokeRole(MINTER_ROLE, minterContract);
    }

    minterContract = _minter;
    grantRole(MINTER_ROLE, _minter);
}
```

This only removes `MINTER_ROLE` from the previous minterContract. It does not revoke the role from:

- the deployer initialized with `MINTER_ROLE`,
- any address previously granted `MINTER_ROLE` manually,
- any forgotten operational wallet that once needed mint permissions.

This breaks the trust model implied by `minterContract`, bypasses factory-side issuance controls and accounting, and can cause unauthorized token inflation through an undocumented privilege path.

Scenario

A practical example is:

1. The token is deployed and the deployer receives `MINTER_ROLE`.
2. The protocol later calls `setMinterContract(factory)` and assumes minting is now factory-only.
3. The deployer still retains `MINTER_ROLE`.
4. The deployer can continue calling `mint()` directly, bypassing the intended factory issuance flow.

Recommendation

If the intended design is that only the configured factory/minter contract may mint, `mint()` should enforce `msg.sender == minterContract` directly instead of relying on a broad `MINTER_ROLE` check. In addition, initialization and minter-rotation flows should remove any unnecessary legacy mint permissions so that mint authority is reduced to a single explicit address.

Alleviation

[Goldfish Gold, 03/26/2026]: The team heeded the advice and resolved the issue by stoping using `MINTER_ROLE` for minting in commit fe3dcb85985685fc7291454ca125e516e879bc60.

GOG-09 | No Cap On Fees

Category	Severity	Location	Status
Logical Issue	Minor	contracts/GoldfishTokenFactory.sol (4a4163e69634397a066245a3384a81e16faa369e): 180	Resolved

Description

The `feeWallets[wallet].feeBps` variables in the contract have no set limits, allowing fees to potentially reach 100%. If this occurs, users would receive no tokens from the contract, resulting in a complete loss of their investment.

Recommendation

We recommend setting a reasonable cap on fees and providing adequate disclosure to the community.

Alleviation

[Goldfish Gold, 03/31/2026]: The team heeded the advice and resolved the issue in commit 843e93fcc6595bfd8ffc171e5849d8f4a5b0a1c4.

GOG-11 Duplicate Fee Wallet Addition Leads To Array Pollution And Gas Inefficiency-Exported

Category	Severity	Location	Status
Design Issue	Minor	contracts/GoldfishTokenFactory.sol (4a4163e69634397a066245a3384a81e16faa369e): 158~170	Resolved

Description

```
158         if (feeWallets[wallet].wallet != address(0)) revert WalletAlreadyExists
();
159
160         feeWallets[wallet] = FeeWallet({
161             wallet: wallet,
162             feeBps: feeBps,
163             active: true
164         });
165
166         activeFeeWallets.push(wallet);
167         emit FeeWalletAdded(wallet, feeBps);
168     }
```

The `addFeeWallet` function in `GoldfishTokenFactory.sol` prevents adding the same wallet address to the `feeWallets` mapping but fails to prevent adding the same address to the `activeFeeWallets` array. This creates a scenario where the same wallet address can exist multiple times in the array while only having one entry in the mapping, leading to inefficient fee calculations and unnecessary gas consumption.

The function checks if `feeWallets[wallet].wallet != address(0)` to prevent duplicate entries in the mapping, but this check occurs after the wallet is already added to the `activeFeeWallets` array. If the transaction reverts due to the duplicate check, the array remains unchanged. However, if there's a race condition or the mapping gets cleared while the array retains entries, the same wallet could be added multiple times to the array.

Recommendation

Add a check to ensure the wallet address doesn't already exist in the `activeFeeWallets` array before adding it, or implement a more robust data structure that maintains consistency between the mapping and array.

Alleviation

[Goldfish Gold, 03/23/2026]: The team heeded the advice and resolved the issue by checking the existence of the wallet in commit 2e52a0c46fa9db7b26a3485109bf4fbbbe557b24

GOG-10 Missing Pause State Check In View Function Leads To Inconsistent Information

Category	Severity	Location	Status
Coding Issue	Informational	contracts/GoldfishToken.sol (4a4163e69634397a066245a3384a81e16faa369e): 292~296	Resolved

Description

The `canTransfer` function in `GoldfishToken.sol` checks if an account is frozen or blacklisted but fails to consider the contract's pause state. This creates inconsistency between the view function's response and the actual transfer behavior when the contract is paused.

The `canTransfer` function is designed to help external systems determine if a transfer would succeed before attempting it. However, when the contract is paused, all transfers are blocked regardless of account status, but the view function doesn't reflect this state. This could lead to failed transactions and poor user experience in frontend applications.

```
292     function canTransfer(address account) external view returns (bool) {
293         if (frozenAccounts[account]) return false;
294         if (blacklistedAccounts[account]) return false;
295         return true;
296     }
```

Recommendation

Add a pause state check `if (paused()) return false;` at the beginning of the `canTransfer` function to provide accurate transfer eligibility information.

Alleviation

[Goldfish Gold, 03/23/2026]: The team heeded the advice and resolved the issue by adding a pause state check in commit 2e52a0c46fa9db7b26a3485109bf4fbbbe557b24

GOG-12 | PII/Financial Information Leakage

Category	Severity	Location	Status
Volatile Code	Informational	contracts/Redemption.sol (4a4163e69634397a066245a3384a81e16faa369e): 243	Acknowledged

Description

The `requestRedemption()` takes `deliveryDetails` (documented as "Delivery address or bank details") as a calldata string and stores it in contract storage. This makes the user's sensitive data permanently public in multiple places: (1) the transaction calldata (visible to mempool watchers, RPC providers, and indexers even if not stored), and (2) contract state (queryable forever). This is a serious confidentiality/compliance risk (doxxing, bank-detail leakage, identity theft) and is irreversible once broadcast.

Recommendation

Fix by not putting sensitive delivery/bank details on-chain at all; instead store only a non-sensitive reference/commitment.

Alleviation

[Goldfish Gold, 03/23/2026]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

> This is not issue, we'll just transfer the normal msg texts for deliveryDetails The requestRedemption() function only stores dollar amount of ggbr and not delivery address or any bank details so with that no further action is needed here.

[CertiK, 03/31/2026]:

> Accepted in principle only if treated as an operational assumption, not a code-level mitigation. The current contract still accepts and persists arbitrary deliveryDetails on-chain, so the privacy issue remains unless the field is removed, replaced with an opaque reference/hash, or otherwise prevented from containing sensitive user data.

GOG-13 | Lack Of Persisting `processedBy` In `processRedemption`

Category	Severity	Location	Status
Logical Issue	Informational	contracts/Redemption.sol (4a4163e69634397a066245a3384a81e16faa369e): 282~295	Resolved

Description

`processRedemption` emits `RedemptionProcessing(..., msg.sender)` but never updates `RedemptionRequest.processedBy`. Since `processedBy` is set in `approveRedemption` (and later possibly overwritten in `rejectRedemption`), reading `redemptionRequests[requestId].processedBy` can incorrectly indicate who processed the redemption. Any off-chain system or UI relying on the stored `processedBy` (rather than events) can be misled, reducing accountability and making it easier for an admin to misattribute actions.

Recommendation

Consider persisting `processedBy` In `processRedemption`.

Alleviation

[Goldfish Gold, 03/23/2026]: The team heeded the advice and resolved the issue in commit 2e52a0c46fa9db7b26a3485109bf4fbbbe557b24.

GOG-14 | Redundant Approval In `completeRedemption`

Category	Severity	Location	Status
Logical Issue	Informational	contracts/Redemption.sol (4a4163e69634397a066245a3384a81e16faa369e): 345	Resolved

Description

The `Redemption.completeRedemption` calls `_safeApprove(ggbrToken, address(tokenFactory), amountToBurn)` before invoking `tokenFactory.burnForRedemption`:

```
344         // First approve factory to burn tokens
345         _safeApprove(ggbrToken, address(tokenFactory), amountToBurn);
346
347         // Burn tokens via factory (tokens are held by this contract)
348         bool success = tokenFactory.burnForRedemption(address(this),
 amountToBurn);
349         if (!success) revert BurnFailed();
```

The factory's `burnForRedemption` ignores allowances and burns directly from the `from` address:

```
277  function burnForRedemption(address from, uint256 amount) nonReentrant external
returns (bool) {
278         if (msg.sender != redemptionContract) revert UnauthorizedCaller();
279         if (from == address(0)) revert ZeroAddress();
280         if (goldfishToken.balanceOf(from) < amount) revert InvalidAmount();
281
282         goldfishToken.burn(from, amount);
283         totalBurned += amount;
284
285         emit TokensBurned(from, amount, "redemption", msg.sender);
286         return true;
287     }
```

```
197  function burn(address from, uint256 amount) external {
198         require(msg.sender == minterContract, "Not authorized");
199         _burn(from, amount);
200         totalBurned += amount;
201         emit TokensBurned(from, amount, "factory_burn");
202     }
```

so the approval in `completeRedemption` is unused. This redundant approval increases gas cost and surface area without changing behavior.

Recommendation

Remove the approval step in `completeRedemption` (or make `burnForRedemption` enforce allowances if that's the intended model).

Alleviation

[Goldfish Gold, 03/23/2026]: The team heeded the advice and resolved the issue in commit 2e52a0c46fa9db7b26a3485109bf4fbbbe557b24.

GOG-15 Unused Event Declaration Leads To Code Bloat And Maintenance Confusion-Exported

Category	Severity	Location	Status
Code Optimization	Informational	contracts/GoldfishToken.sol (4a4163e69634397a066245a3384a81e16faa369e): 49	Resolved

Description

```
142         event TransfersEnabledUpdated(bool enabled);
```

The `TransfersEnabledUpdated` event is declared in `GoldfishToken.sol` but is never emitted anywhere in the contract code. This creates unnecessary code bloat and could confuse developers and auditors about the contract's intended functionality.

Unused event declarations suggest incomplete implementation or removed functionality that wasn't properly cleaned up. This can lead to confusion during code reviews, audits, and maintenance activities. It also increases the contract's bytecode size unnecessarily, potentially affecting deployment costs.

Recommendation

Remove the unused `TransfersEnabledUpdated` event declaration to clean up the code and eliminate confusion about the contract's intended functionality.

Alleviation

[Goldfish Gold, 03/23/2026]: The team heeded the advice and resolved the issue in commit 2e52a0c46fa9db7b26a3485109bf4fbbbe557b24

GOG-16 | Missing `whenNotPaused` Modifier In `updateDeliveryDetails`

Category	Severity	Location	Status
Inconsistency	Informational	contracts/Redemption.sol (4a4163e69634397a066245a3384a81e16faa369e): 393~396	Resolved

Description

The `updateDeliveryDetails` function lacks the `whenNotPaused` modifier present in most other state-changing functions in the `Redemption` contract:

```
393 function updateDeliveryDetails(
394         uint256 requestId,
395         string calldata newDetails
396     ) external {
397     ...
398     ...
```

This creates an inconsistency in the contract's security model and could allow state modifications during emergency pauses, potentially undermining the purpose of the pausable functionality.

Recommendation

Consider adding `whenNotPaused` in `updateDeliveryDetails`.

Alleviation

[Goldfish Gold, 03/23/2026]: The team heeded the advice and resolved the issue in commit 2e52a0c46fa9db7b26a3485109bf4fbbbe557b24.

OPTIMIZATIONS | GOLDFISH GOLD

ID	Title	Category	Severity	Status
GOG-01	`getUserRedemptionHistory` Can Exhaust Gas On Large User Histories	Coding Issue	Optimization	Acknowledged

GOG-01 `getUserRedemptionHistory` Can Exhaust Gas On Large User Histories

Category	Severity	Location	Status
Coding Issue	Optimization	contracts/Redemption.sol (4a4163e69634397a066245a3384a81e16faa369e): 441~456	Acknowledged

Description

The `getUserRedemptionHistory` function does `uint256[] memory requestIds = userRequests[user];`, which copies the *entire* dynamic storage array into memory before applying `limit`. If a user accumulates a large number of requests, calls to `getUserRedemptionHistory(user, smallLimit)` can become extremely expensive and may revert due to out-of-gas / RPC gas caps, effectively preventing retrieval of even the most recent entries. This defeats the purpose of `limit` and can break UIs/backends that rely on this getter.

Recommendation

Avoid copying `userRequests[user]` into memory. Read directly from storage and index only the required elements.

Alleviation

[Goldfish Gold, 03/23/2026]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

APPENDIX | GOLDFISH GOLD

Finding Categories

Categories	Description
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Code Optimization	No description available.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevate Your Web3 Journey

CertiK is the largest Web3 security platform combining formal verification with audits and comprehensive security solutions.