CERTIK

# Security Assessment

# Cache Private Limited

Jul 29th, 2022

# Table of Contents

## Summary

This report has been prepared for Cache Private Limited to discover issues and vulnerabilities in the source code of the Cache Private Limited project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

# Overview

## Project Summary

| | |
|---|---|
| Project Name | Cache Private Limited |
| Platform | Ethereum, Polygon |
| Language | Solidity |
| Codebase | https://github.com/cache-token/cgt-bridge |
| Commit | 0fa745ecddc9749d19d75dc8bf6579615094d82f |

## Audit Summary

| | |
|---|---|
| Delivery Date | Jul 29, 2022 UTC |
| Audit Methodology | Static Analysis, Manual Review |

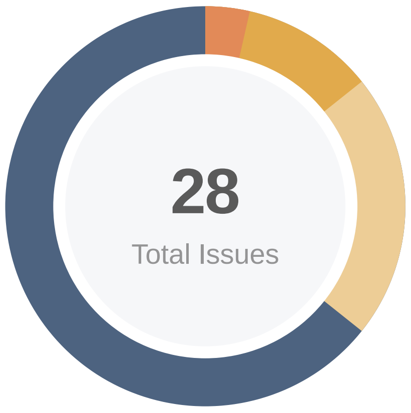
## Vulnerability Summary

| Vulnerability Level | Total | Pending | Declined | Acknowledged | Mitigated | Partially Resolved | Resolved |
|---|---|---|---|---|---|---|---|
| ● Critical | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ● Major | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| ● Medium | 3 | 0 | 0 | 0 | 0 | 0 | 3 |
| ● Minor | 6 | 0 | 0 | 0 | 0 | 0 | 6 |
| ● Optimization | 3 | 0 | 0 | 1 | 0 | 0 | 2 |
| ● Informational | 18 | 0 | 0 | 2 | 0 | 0 | 16 |
| ● Discussion | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## Audit Scope

| ID | File | SHA256 Checksum |
|---|---|---|
| FER | contracts/L2_polygon/cgt/erc20-transfer/FxERC20ChildTunnel.sol | f37b8067fe7526c7f4d9bfc02468df678e672536b194cee6974b3b69ed8048f6 |
| FEC | contracts/L2_polygon/cgt/erc20-transfer/FxERC20RootTunnel.sol | 5c8b1ed30b421f685d37182e40ce28b5323ce390793afb23f3cd8cbfed1e98e1 |
| CGC | contracts/L2_polygon/tokens/CacheGoldChild.sol | 0dea289138594b6a26558574433f587f576f2ceed9a46a39d5bb27ea27a162e3 |

# Findings



| | | |
|---|---|---|
| 🟥 **Critical** | **0** (0.00%) |
| 🟧 **Major** | **1** (3.57%) |
| 🟨 **Medium** | **3** (10.71%) |
| 🟨 **Minor** | **6** (21.43%) |
| 🟦 **Informational** | **18** (64.29%) |
| 🟩 **Discussion** | **0** (0.00%) |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| CGC-01 | Centralization Risks | **Centralization / Privilege** | ● **Major** | Mitigated |
| CGC-02 | Missing Input Validation For Initialize | Volatile Code | ● Medium | ✓ Resolved |
| CGC-03 | User Can Avoid Storage Fees By Depositing Gold | Logical Issue | ● Minor | ✓ Resolved |
| CGC-06 | Unnecessary Braces + Irrelevant Comment | Coding Style | ● Informational | ✓ Resolved |
| CGC-07 | Should Make Old Owner `unsetfeeexempt()` | Inconsistency | ● Medium | ✓ Resolved |
| CGC-08 | Push Over Pull For `TransferOwnership` Method Implementation | Control Flow | ● Informational | ✓ Resolved |
| CGC-09 | Transfer Fee Can Be Skipped | Mathematical Operations | ● Informational | ✓ Resolved |
| CGC-10 | Redundant `if` Statement | Logical Issue | ● Informational | ✓ Resolved |
| CGC-11 | Missing Error Messages | Coding Style | ● Informational | ✓ Resolved |
| CGC-12 | Specific Event For Burning | Coding Style | ● Informational | ✓ Resolved |
| CGC-13 | Missing Override Specifier | Compiler Error | ● Minor | ✓ Resolved |
| CGC-14 | Fee Exemption Inconsistency In `initialize()` | Volatile Code, Inconsistency | ● Medium | ✓ Resolved |

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| CGC-15 | `initialize()` Is Unprotected | Control Flow | ● Minor | ✓ Resolved |
| CGC-16 | Missing Initialization For The `_redeemAddress` And The `_oracle` | Volatile Code | ● Minor | ✓ Resolved |
| CGC-17 | Missing Input Validation | Volatile Code, Gas Optimization | ● Minor | ✓ Resolved |
| CGC-18 | Third Party Dependencies | Volatile Code | ● Minor | ✓ Resolved |
| CGC-19 | Redundant Variable Initialization | Coding Style | ● Informational | ✓ Resolved |
| CGC-20 | Too Many Digits | Coding Style | ● Informational | ✓ Resolved |
| CGC-21 | Redemption Is A Centralized Manual Process | Control Flow | ● Informational | ✓ Resolved |
| CGC-22 | Mint Event Missing Information | Data Flow | ● Informational | ✓ Resolved |
| CGC-23 | Missing Emit Events | coding | ● Informational | ✓ Resolved |
| CGC-24 | Teaching Users How To Avoid Inactive Fees | Data Flow | ● Informational | ✓ Resolved |
| CGC-25 | Different Solidity Versions | Language Specific | ● Informational | ✓ Resolved |
| CGC-26 | Information Returned In `BalanceOf()` Could Cause Confusion | Logical Issue | ● Informational | ⓘ Acknowledged |
| CGC-27 | Unclear Function Naming `calcSendAllBalance` | Coding Style | ● Informational | ✓ Resolved |
| CGC-28 | Order Of Layout | Language Specific, Coding Style | ● Informational | ✓ Resolved |
| CGC-29 | Outside Dependency For Redemption Fee | Logical Issue | ● Informational | ⓘ Acknowledged |
| L2P-02 | Spelling Mistakes | Coding Style | ● Informational | ✓ Resolved |

## CGC-01 | Centralization Risks

| Category | Severity | Location | Status |
|---|---|---|---|
| **Centralization / Privilege** | ● **Major** | contracts/L2_polygon/tokens/CacheGoldChild.sol: 89~90 | Mitigated |

### Description

In the contract `CacheGoldChild.sol` the `_owner` has authority over the following critical functions:

- `setFxManager()`
- `setAccountInactive()`
- `setNewOwner()`
- `forcePayFees()`
- `setFeeAddress()`
- `setRedeemAddress()`
- `setStorageFeeGracePeriodDays()`
- `setTransferFeeExempt()`
- `setStorageFeeExempt()`
- `setStorageFeeGracePeriodDays()`
- `setStorageFeeExempt()`
- `unsetFeeExempt()`
- `setTransferFeeBasisPoints()`
- `setFeeExempt()`

Furthermore the `_fxManager` has authority over the burn and mint functionality. Any compromise to the `_owner` account can cause the entire project to fail and cause every user to lose their funds as this particular project relies on the integrity of the contract as a proof of ownership to the underlying physical gold assets kept in the reserve, and if that is compromised, users will no longer be able to realistically trade their tokens for physical gold. With the amount of authority the `_owner` role has, there are numerous ways the attacker can cause the project to fail if that account is compromised.

### 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

`[Cache Gold]` For the said functions, in mainnet we make use of a multisig, with multiple signatories as per business defined specifications.

For polygon where the FxManager comes into play it is the same mechanism as used by Polygon team and the CACHE team does not have control other than setting the address of the FxManager.

For the token itself it is managed by a centralized entity CACHE Pte Ltd and not a DAO as of now.

Centralization Mitigation Info

Multi-signature

Platform: ETH

Multi-sign proxy address: https://etherscan.io/address/0xb779efeeda6cf887b80bc386e7eb9fdced6753f6

Transaction proof for transferring ownership to multi-signature proxy:

https://etherscan.io/tx/0xb640387a1b8d4b962da4809f866d11899a25e504118477020960af57ea3210fb

Internal multi-signature address:<br>
https://etherscan.io/address/0xe682BEC7719c8199bc0c0BE5cAd9e795E20cC9A5<br>
https://etherscan.io/address/0xba88A7025e206C5Fbd8b8Da5b606e51b317aD484<br>
https://etherscan.io/address/0xF9fB92Df1ebC5272847CFB09AedA9fba8418d3e8<br>
https://etherscan.io/address/0xDbb7Ab8C92dFf0C8836FAc029d6B72DeD09e0328<br>
https://etherscan.io/address/0x92249Ef4B3Dceb2827390Bf9bDd3E82C5Fa19dEe<br>
https://etherscan.io/address/0x38A5dA84cb8d7D8ea27bfED6e13E4B9F1b7E8e96<br>
https://etherscan.io/address/0x0df142197Cc48701C69AA7d96A6f20189A3E267a<br>
https://etherscan.io/address/0x032c079DB811541501Bd769836490Ce0f5999264<br>

## CGC-02 | Missing Input Validation For Initialize

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | ● Medium | contracts/L2_polygon/tokens/CacheGoldChild.sol: 109~126 | ✓ Resolved |

### Description

Due to the require statement:

```
require(_fxManager == address(0x0) && _connectedToken == address(0x0), "Token is already
initialized");
```

There is increased emphasis of having a check that the owners do no input the 0 address into these fields. Furthermore, the contract logic depends on the `__decimals` fields being 8. There should be no option for it to be changed from this value, otherwise the contracts mathematical operations will be incorrect. An example of this is: `because 1^8 / STORAGE_FEE_DENOMINATOR`, is 0.25%. If the decimal value changes, the entire fee structure changes.

### Recommendation

We advise the team to put this check in the `initialize` function:

```
require(__fxManager_ != address(0x0) && __connectedToken != address(0x0), "Zero address
inputted");
```

And to hard code the `decimals` metadata in the contract as 8, and remove the option to alter it from the initialize function.

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash 254536364090b7c898336714be769519b7bd047d.

## CGC-03 | User Can Avoid Storage Fees By Depositing Gold

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Minor | contracts/L2_polygon/tokens/CacheGoldChild.sol: 274~275 | ✓ Resolved |

### Description

When a user deposits physical gold and gets tokens minted into their account, `_timeStorageFeePaid[user]` gets set to the current `block.timestamp`. Therefore if a user periodically deposits physical gold into their account, they can avoid storage fees altogether.

### Recommendation

If this is an un intended by-product of depositing gold, we advice the team the design a new mechanism whereby storage fees are not avoided simply by depositing need gold into their account. For example having a check like this would suffice:

```
 function mint(address user, uint256 amount)
        public
        override
    {
        require(msg.sender == _fxManager, "Invalid sender");
        _totalSupply = _totalSupply + amount;
        _balances[user] = _balances[user] + amount;
        emit Mint(amount);
        if(_timeStorageFeePaid[user] == 0){ //checks if it is the first time the user is
depositing gold into the account
        _timeStorageFeePaid[user] = block.timestamp;
        }
    }
```

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash 0e8b091c14c83bc5939fc404b7d169ebb5450697.

## CGC-06 | Unnecessary Braces + Irrelevant Comment

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Style | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 291~294, 295 | ✓ Resolved |

### Description

This piece of code in the `burn` functions logic:

```
unchecked
{//https://github.com/OpenZeppelin/openzeppelin-contracts/issues/2665
_approve(account, msg.sender, currentAllowance - amount);
}
```

Is unnecessary and unclear. It is unknown to us why a structure called `unchecked` is used, and why there is a comment linking to an irrelevant ERC721 issue.

### Recommendation

We recommend cleaning up the code like so:

```
_approve(account, msg.sender, currentAllowance - amount);
```

To achieve the same results and improve readability.

Unchecked can be put onto

```
_balances[account] = _balances[account] - amount;
```

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash c5f0eca7bbe7ffc4150396e12d0de2e8dee18af5.

## CGC-07 | Should Make Old Owner `unsetfeeexempt()`

| Category | Severity | Location | Status |
|---|---|---|---|
| Inconsistency | ● Medium | contracts/L2_polygon/tokens/CacheGoldChild.sol: 379~388, 395~404, 411~416 | ✓ Resolved |

### Description

When setting a new owner, the new owner is made fee exempt using the `setFeeExempt` function. However the old owner also stays fee exempt because there is no logic in the `setNewOwner` function that revokes the right of the old owner to be fee exempt.

### Recommendation

We recommend implementing the `unsetFeeExempt` on the old owner account like so:

```
function setNewOwner(address __owner)
        external
        onlyOwner
        returns (bool)
    {
        require(__owner != address(0));
        unsetFeeExempt(msg.sender);
        _owner = __owner;
        setFeeExempt(__owner);
        return true;
    }
```

We recommend following a similar methodology in `setFeeAddress` and `setRedeemAddress`. Note that in the current implementation, `unsetFeeExempt` is an external function, to apply these changes, this function must be make public:

```
 function unsetFeeExempt(address account) public onlyOwner {
        _transferFeeExempt[account] = false;
        _storageFeeExempt[account] = false;
    }
```

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash 937822fe52e4a261a7be0f710a702fb4bab20079.

## CGC-08 | Push Over Pull For `TransferOwnership` Method Implementation

| Category | Severity | Location | Status |
|---|---|---|---|
| Control Flow | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 379~388, 395~404, 411~416 | ✓ Resolved |

### Description

The public `setNewOwner` method in the `CacheGoldChild.sol` class is implemented using a typical "Push" pattern, which can result in the loss of contract ownership if an invalid owner address is supplied.

### Recommendation

Consider refactoring the ownership transfer using the "Pull over Push" pattern, requiring the designated new owner of the contract to call the contract in order to claim ownership. Repeat this process for the functions that set the `_redeemAddress` and the `feeAddress`.

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash 613455330cd7e84c680337e65d18088bc107cd71.

## CGC-09 | Transfer Fee Can Be Skipped

| Category | Severity | Location | Status |
|---|---|---|---|
| Mathematical Operations | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 784~785 | ✓ Resolved |

### Description

Since the transfer fee is `(value * (_transferFeeBasisPoints) / (Basis_points_multiplier))`. since basis points multiplier is 10000. if value is set such that `(value * (_transferFeeBasisPoints)` < 10000, there will be no transfer fee. in the current implimentation with a fee of 10, any value below 1000 will have no transfer fee.

### Recommendation

Confirm that mathematic operations are consistent with the goals of the business.

### Alleviation

`[Cache Gold]`: We checked on this and this is as per business expectations that there are no fees on dust amounts. https://github.com/cache-token/cgt-bridge/blob/0fa745ecddc9749d19d75dc8bf6579615094d82f/test/CGT.test.ts#L993

## CGC-10 | Redundant `if` Statement

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 1057~1060 | ✓ Resolved |

### Description

The `if` statement:

```
if (_storageFeeGracePeriod[account] > 0) {
_storageFeeGracePeriod[account] = 0;
        }
```

Is redundant because `_storageFeeGracePeriod[account]` is always either 0 or greater than 0.

### Recommendation

You can refactor the code like so to improve readability:

```
function _endGracePeriod(address account) internal {
_storageFeeGracePeriod[account] = 0;
    }
```

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash df598501b1a1bcd7bcabe8a0b611972ea3f41842.

## CGC-11 | Missing Error Messages

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Style | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 343, 352, 384, 400, 412, 732, 823~824, 1138, 1139 | ✓ Resolved |

### Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

### Recommendation

We advise adding error messages to the linked **require** statements. For example, in the `forcePayFees` function you can refactor the require statement to:

```
require(account != address(0), "ERROR, ZERO ADDRESS USED");
```

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash 00779b6fe4cfb3d73f690e3f2ff03c27fd40584e.

## CGC-12 | Specific Event For Burning

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Style | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 297~298 | ✓ Resolved |

### Description

It seems that the `FxTunnel` will be listening for a transfer to `address(0)` denoting a burn to withdraw on the mainnet. If this the case, the `FxTunnel` will be listening to all transfers to the 0 address which may not necessarily indicate a withdraw to the mainnet. This may cause unexpected results and the project may be subject to grief attacks as an attacker can continuously send >= 0 tokens to the 0 address and cause these transfer events to be emitted which the `FxTunnel` is listening to, thinking this is a withdraw to the main-net.

### Recommendation

We recommend making a specific `burn` which only gets triggered in the `burn` function and is the event that the `FxTunnel` is listening to for withdrawals into the mainnet. E.g.

```
event withdrawBurn(address _from, uint256 _amount);
emit withdrawBurn(account, amount);
```

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash c5f0eca7bbe7ffc4150396e12d0de2e8dee18af5.

## CGC-13 | Missing Override Specifier

| Category | Severity | Location | Status |
|---|---|---|---|
| Compiler Error | ● Minor | contracts/L2_polygon/tokens/CacheGoldChild.sol: 282~284, 808~810 | ✓ Resolved |

### Description

The contract does not compile because overriding functions `burn` and `totalSupply` are missing override specifiers.

### Recommendation

For the inheriting functions, it is required to add virtual to every non-interface function intended to override, and to add override to the overriding functions, according to the Solidity 0.6.0 Breaking Changes. For the `burn` function:

```
function burn(address account, uint256 amount) override public {
```

And for the `totalSupply` function:

```
function totalSupply() external override view returns (uint256) {
```

### Alleviation

`[Cache Gold]` Issue acknowledged. Changes have been reflected in the commit hash c5f0eca7bbe7ffc4150396e12d0de2e8dee18af5.

## CGC-14 | Fee Exemption Inconsistency In `initialize()`

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code, Inconsistency | ● Medium | contracts/L2_polygon/tokens/CacheGoldChild.sol: 123~124 | ✓ Resolved |

### Description

In the `initialize` function, when the `_feeAddress` is set, it is made exempt from all fees by this call: `setFeeExempt(_feeAddress);`. However `_owner`, `_fxManager` and `_connectedToken` are not set as fee exempt although these are critical addresses. Furthermore it seems that it is intended that `_owner` is meant to be fee exempt because in the `setNewOwner` function, `setFeeExempt` is called.

### Recommendation

We recommend making the appropriate accounts fee exempt in the initialize function to avoid unexpected transfer and storage fees onto admin accounts. Have these checks in the `Initialize` function:

```
setFeeExempt(_fxManager);
setFeeExempt(_owner);
setFeeExempt(_connectedToken);
setFeeExempt(_feeAddress);
```

### Alleviation

`[Cache Gold]`: _connectedToken is the token address on mainnet. _owner and _fxManager have been exempted for consistency. Commit hash: 36b1fcf20a388987162334df8853ea105c3eb7ef

## CGC-15 | `initialize()` Is Unprotected

| Category | Severity | Location | Status |
|---|---|---|---|
| Control Flow | ● Minor | contracts/L2_polygon/tokens/CacheGoldChild.sol: 109~126 | ✓ Resolved |

### Description

The function `initialize()` is `public` and can be called by anyone as long as the contract is deployed.

### Recommendation

We recommend placing restrictions on the function to only allow a trusted party to initialize the contract. E.g. by setting an owner through a constructor and initializing later if needed with the `onlyOwner()` modifier.

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash 613455330cd7e84c680337e65d18088bc107cd71.

## CGC-16 | Missing Initialization For The `_redeemAddress` And The `_oracle`

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | ● Minor | contracts/L2_polygon/tokens/CacheGoldChild.sol: 109~117 | ✓ Resolved |

### Description

In the current implementation, `_redeemAddress` is not set in the initialize function. `_oracle` is not defined as well, however its functionality is missing from the contract as discussed in another finding.

### Recommendation

We recommend the team include `_redeemAddress` and `_oracle` in the `initialize` function. For example:

```
function initialize(
        address __feeAddress,
        address __owner,
        address __fxManager_,
        address __connectedToken,
        address __redeemAddress,
        address __oracle,
        string memory __name,
        string memory __symbol,
        uint8 __decimals
     ) public override {
```

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash cfa1e135e17aca63efbf39b2a58ff61a0cb06acc.

## CGC-17 | Missing Input Validation

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code, Gas Optimization | ● Minor | contracts/L2_polygon/tokens/CacheGoldChild.sol: 156~160 | ✓ Resolved |

### Description

There is not a check to see whether the senders balance is greater than the inputted `value`.

Also there is no check to see whether the value being transferred is greater than 0.

### Recommendation

Although the call should still fail if the value inserted is greater than the senders balance due the the arithmetic present in the `_transfer` function (due to solidity fixing the overflow and underflow issue in version `^0.8.0`, you can avoid unnecessary function calls just by putting this check at the start:

```
require(_balances[msg.sender] >= value, "Insufficient Balance To Make This Transfer")
```

We recommend adding this check to the `_transfer` logic:

```
require(value > 0, "Cannot transfer 0 tokens");
```

### Alleviation

`[Cache Gold]`: Issue acknowledged. In order to allow to the specific CGT use case on mainnet of paying storage fees by sending 0 tokens to self, we only do the balance check. Changes have been reflected in the commit hash 54cdb94384cc3da5c1d1f63a9005ef50c49cea87.

## CGC-18 | Third Party Dependencies

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | ● Minor | contracts/L2_polygon/tokens/CacheGoldChild.sol: 77 | ✓ Resolved |

### Description

The contract is serving as the underlying entity to interact with third party `oracle` protocols. The scope of the audit treats 3rd party entities as black boxes and assume their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

### Recommendation

We understand that the business logic of `CacheGold` requires interaction with `oracle`. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

### Alleviation

`[Cache Gold]`: Currently, the cache gold PoR is live on mainnet at https://data.chain.link/ethereum/mainnet/reserves/cachegold-por-usd

We do have this check on mainnet, however in polygon, we decided to remove it because only the fxManager is allowed to mint new tokens.

Adding a LockedGoldOracle on polygon would be under the assumption that emitting the oracle data onto polygon would also require going through the fxManager arbitrary message system, which does not add any additional security measure.

## CGC-19 | Redundant Variable Initialization

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Style | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 11~13 | ✓ Resolved |

### Description

The initialization of this variables are redundant:

```
string public name = "CACHE Gold";
string public symbol = "CGT";
uint8 public decimals = 8;
```

because the values of these variables are set when the contract is initialized.

Also the `_setupMetaData` function that is called inside `initialize` is redundant as the metadata is already hard coded into the contract and any changes can lead to the contract failing, for example, changing the decimals value.

### Recommendation

We recommend the client having a clear path on how they want to setup the contract on deployment. You may initialize the values like so:

```
string public name;
string public symbol;
uint8 public decimals;
```

or if you're sure that these are the values intended for use, remove the option to change them in the initialize function and make these values immutable.

```
string public immutable name = "CACHE Gold";
string public immutable symbol = "CGT";
uint8 public immutable decimals = 8;
```

We recommend removing the `_setupMetaData` from the contract as the fields are already initialized by being hard coded into the contract.

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash 3a90bb22bad0ce8a227f0e4c4c2305dad615e790.

## CGC-20 | Too Many Digits

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Style | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 28~29 | ✓ Resolved |

### Description

Literals with too many digits are difficult to read and review, such as:

```
uint256 private constant STORAGE_FEE_DENOMINATOR = 40000000000;
uint256 private constant INACTIVE_FEE_DENOMINATOR = 20000000000;
```

### Recommendation

We advise the client to use the scientific notation to improve readability:

```
uint256 private constant STORAGE_FEE_DENOMINATOR = 4e10;
uint256 private constant INACTIVE_FEE_DENOMINATOR = 2e10;
```

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash 8df22b7bc2daff57de6027ba7aadbfb50161fc55.

## CGC-21 | Redemption Is A Centralized Manual Process

| Category | Severity | Location | Status |
|---|---|---|---|
| Control Flow | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 83~84 | ✓ Resolved |

### Description

To redeem tokens for gold, the user sends funds to redeem address. We recommend ensuring the redeem address multi-sig to ensure funds cannot be lost easily. The funds being sent to the centralized wallet and being redeemed for real gold is outside the scope of the audit. We treat this as a black box and assume functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to loss of assets.

### Recommendation

We recommend the client to constantly monitor how the redemption process takes place to ensure mitigate side effects or unexpected activities.

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash 8ac2ea839b0c5e526dc935c5aa48a728f5cb09da.

## CGC-22 | Mint Event Missing Information

| Category | Severity | Location | Status |
|---|---|---|---|
| Data Flow | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 100~101 | ✓ Resolved |

### Description

In the current implementation the `Mint` event only emits the amount that is minted without showing which account the contract minted to.

### Recommendation

We advise adding and address field to the mint event for complete information flow:

```
event Mint(uint256 amount, address account);
```

```
function mint(address user, uint256 amount)
        external
        override
    {
        require(msg.sender == _fxManager, "Invalid sender");
        _totalSupply = _totalSupply + amount;
        _balances[user] = _balances[user] + amount;
        emit Mint(amount,user);
        _timeStorageFeePaid[user] = block.timestamp;
    }
```

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash 8df22b7bc2daff57de6027ba7aadbfb50161fc55.

## CGC-23 | Missing Emit Events

| Category | Severity | Location | Status |
|---|---|---|---|
| coding | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 379~388, 395~404, 411 ~416, 424~429, 462 | ✓ Resolved |

### Description

The function that affects the status of sensitive variables should be able to emit events as notifications. In the current contract implementation, no events are emitted when a new owner, fee address, redeem address are set. Nor when the storage fee grace period or the transfer fee basis points are changed.

### Recommendation

Consider adding events for sensitive actions, and emit them in the respective functions.

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash 8df22b7bc2daff57de6027ba7aadbfb50161fc55.

## CGC-24 | Teaching Users How To Avoid Inactive Fees

| Category | Severity | Location | Status |
|---|---|---|---|
| Data Flow | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 1155~1172 | ✓ Resolved |

### Description

Since not all users are very technically proficient, the team should make an effort to educate users on all the ways they can update their activity to avoid paying inactive fees as there is a culture in the crypto currency community (and in the physical gold community) to simply hold their assets. If they are not fully aware of the fees and how to avoid them, they could be unknowingly charged periodically.

### Recommendation

Make medium blog posts detailing the fee structure and how to keep accounts active. If token are listed on exchanges, briefly describe the fee structure in the token summary section.

### Alleviation

`[Cache Gold]`: This is clearly disclosed in the documentation, on our website and in our YouTube videos. Any transaction, receipt, transfer or call of payStorageFee() keeps the account active.

## CGC-25 | Different Solidity Versions

| Category | Severity | Location | Status |
|---|---|---|---|
| Language Specific | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 2~3 | ✓ Resolved |

### Description

Multiple Solidity versions are used in the codebase. This leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one. `CacheGoldChild.sol` uses version `^0.8.11` which inherits from: `IFxERC20`, which uses version `^0.8.0`

### Recommendation

We recommend using one Solidity version. We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. for example version `0.8.11`.

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash f8f81cc0319bb19121049befba76d7af5e3df50a.

## CGC-26 | Information Returned In `BalanceOf()` Could Cause Confusion

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 477~479 | ⓘ Acknowledged |

### Description

The function `balanceOf()` calls `calcSendAllBalance` and returns its value. However the balance of an account is not the same as the `calcSendAllBalance` (the maximum amount an account can transfer when fee's are taken into account). This misleading information may lead to confusion and distress to users. For example, if a user gets minted 10000000000 tokens to their account, and immediately checks their balance using `balanceOf`, the value that is returned is: 9990009991, when they actually still have 10000000000 tokens in their account.

### Recommendation

We recommend the function `balanceOf` to simply return `_balances[account]`

```
function balanceOf(address owner) external view returns (uint256) {
return _balances[owner];
    }
```

Furthermore it is unknown why an additional function is needed to return the value of `calcSendAllBalance` when `calcSendAllBalance` can just be called on its own.

### Alleviation

`[Cache Gold]`: Since this is a specific design decision that is well documented with clear logic and reasons supporting the decision we do not acknowledge this as an audit issue. If we had chosen to return _balances[account] to balanceOf() rather than the current spendable balance, the send all/send max feature of standard ERC-20 wallets would be broken. The balanceOfNoFees() function is provided to return _balances[account].

## CGC-27 | Unclear Function Naming `calcSendAllBalance`

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Style | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 731~732 | ✓ Resolved |

### Description

The name `calcSendAllBalance` doesn't clearly describe the functionality of the function. The comments in the code state that the function "calculate the amount that would clear the balance from the address", which is not obvious from the function name.

### Recommendation

We recommend the team to use a clearer name such as "maximumTransferAmount" for example.

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash 2d7d2a2aff5fad06efde453cd6caa1f9d381ddc1.

## CGC-28 | Order Of Layout

| Category | Severity | Location | Status |
|---|---|---|---|
| Language Specific, Coding Style | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 131 ~134 | Resolved |

### Description

The `CacheGoldChild` contract does not conform to the order of layout recommended by the solidity documentation.

### Recommendation

The solidity documentation recommends that inside each contract, library or interface, to use the following order: Type declarations, State variables, Events, Modifiers, Functions. And that functions be grouped according to their visibility: constructor, receive function (if exists), fall-back function (if exists), external, public, internal, private.

### Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commit hash f8f81cc0319bb19121049befba76d7af5e3df50a.

## CGC-29 | Outside Dependency For Redemption Fee

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 83~84 | ⓘ Acknowledged |

### Description

The address is serving as the underlying entity to interact with contract a redemption fee. The scope of the audit treats this address and fee out of scope and as black boxes and assumes their functional correctness.

However, in the real world, the redemption fee can be anything and can be compromised.

### Recommendation

We advice the team to maintain and document clearly how the fee works.

### Alleviation

`[Cache Gold]`: All aspects of redemption including fees are clearly documented on the CACHE Gold website.

## L2P-02 | Spelling Mistakes

| Category | Severity | Location | Status |
|---|---|---|---|
| Coding Style | ● Informational | contracts/L2_polygon/tokens/CacheGoldChild.sol: 569, 578, 587, 1148~1149; contracts/L2_polygon/tunnel/FxBaseChildTunnel.sol: 14 | ✓ Resolved |

### Description

There are some spelling mistakes in contract: `Calcuate` should be `Calculate`, `Extempt` should be `Exempt`, `Maping` should be `Mapping`.

### Recommendation

We recommend correcting these spelling mistakes to improve code readability.

### Alleviation

`[Cache Gold]`: Fixed in commit: f4fcf2bbda6b71643742081781170f040daa0cda.

# Optimizations

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| CGC-04 | Variables That Could Be Declared As Immutable | Gas Optimization | ● Optimization | ⓘ Acknowledged |
| CGC-05 | Redundant Function | Gas Optimization, Coding Style | ● Optimization | ✓ Resolved |
| L2P-01 | Function Visibility Optimization | Gas Optimization | ● Optimization | ✓ Resolved |

## CGC-04 | Variables That Could Be Declared As Immutable

| Category | Severity | Location | Status |
|---|---|---|---|
| Gas Optimization | ● Optimization | contracts/L2_polygon/tokens/CacheGoldChild.sol: 9~10, 16, 19, 22, 25, 28, 31, 34, 40 | ⓘ Acknowledged |

### Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

### Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

### Alleviation

`[Cache Gold]`: Since both constants are immmutable and do not get stored in storage, we would like to keep it consistent with the mainnet contract regardless of optimization benefits.

## CGC-05 | Redundant Function

| Category | Severity | Location | Status |
|---|---|---|---|
| Gas Optimization, Coding Style | ● Optimization | contracts/L2_polygon/tokens/CacheGoldChild.sol: 436, 445~447, 555, 839, 1073~1100 | ✓ Resolved |

### Description

The function `simulateTransfer` calls the function `_simulateTransfer` which is only called once and returns a value in the `simulateTransfer` function. Calling a separate function is redundant as you could just put the logic of `_simulateTransfer` into the body of the `simulateTransfer` function.

In the function `_transfer` the first call it makes is to the `_transferRestrictions(to,from)` function. Having `_transferRestrictions` as a seperate function is redundant as it is called only once in the whole contract (in the `_transfer` function), therefore you could just put the logic that is within `_transferRestrictions` into the `_transfer` function and save gas.

Having two separate functions to `setTransferFeeExempt` and to `setStorageFeeExempt` seems redundant as there isn't a clear role that would be exempt from transfer fees, but would have to pay storage fees, and vice versa. In addition there is already a function called `setFeeExempt` which fulfils the logic of both `setTransferFeeExempt` and `setStorageFeeExempt` in one function call, and is used internally throughout the contract.

### Recommendation

You can refactor the `simulateTransfer` function as so:

```
function simulateTransfer(
        address from,
        address to,
        uint256 value
    ) external view returns (uint256[5] memory) {
        uint256[5] memory ret;
        // Return value slots
        // 0 - fees `from`
        // 1 - fees `to`
        // 2 - transfer fee `from`
        // 3 - final `from` balance
        // 4 - final `to` balance
        ret[0] = calcOwedFees(from);
        ret[1] = 0;
        ret[2] = 0;
        // Don't double charge storage fee sending to self
        if (from != to) {
            ret[1] = calcOwedFees(to);
            ret[2] = calcTransferFee(from, value);
            ret[3] = (((_balances[from] - value) - ret[0]) - ret[2]);
            ret[4] = ((_balances[to] + value) - ret[1]);
        } else {
            ret[3] = _balances[from] - (ret[0]);
            ret[4] = ret[3];
        }
        return ret;
    }
```

And remove `_simulateTransfer` from the code.

We recommend refactoring the beginning of the `_transfer` function like so:

```
function _transfer(address from, address to, uint256 value) internal {
// redeem address can only call burn
require(from != _redeemAddress, "Redeem address can only transfer to mainnet by
burning");
require(from != address(0));
require(to != address(0));
require(to != address(this), "Cannot transfer tokens to the contract");
...
...
}
```

We recommend removing unneeded functions from the code.

## Alleviation

`[Cache Gold]`: Issue acknowledged. Changes have been reflected in the commits 71c5c01597a842cb9778bada5042428366fdb9ef and 613455330cd7e84c680337e65d18088bc107cd71.

## L2P-01 | Function Visibility Optimization

| Category | Severity | Location | Status |
|---|---|---|---|
| Gas Optimization | ● Optimization | contracts/L2_polygon/cgt/erc20-transfer/FxERC20ChildTunnel.sol: 30; contracts/L2_polygon/tokens/CacheGoldChild.sol: 117~118, 137, 146, 157, 184, 208, 226, 249, 477, 488, 600 | ✓ Resolved |

### Description

The following functions are declared as `public`, and are not invoked in any of the contracts contained within the project's scope. The functions that are never called internally within the contract should have external visibility.

### Recommendation

We advise that the functions' visibility specifiers are set to `external`

### Alleviation

`[Cache Gold]`: Fixed in commit: f4fcf2bbda6b71643742081781170f040daa0cda.

# Appendix

## Finding Categories

### Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

### Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

### Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

### Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

### Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

### Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

### Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

### Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

### Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

### Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

### Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

## Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

## Disclaimer

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

## About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we’re able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.