The CACHE team asked us to review the CACHE Gold contracts. We looked at the code and now publish our results.

The audited commit is ab745ff0279829adeb07bb387bba74d9c419a9e6 and the files included in scope were CacheGold.sol and LockedGoldOracle.sol.

This audit was a full audit covering the entire project. This audit encompassed a deep dive into the system as a whole and verified compliance with the ERC20 specification. Additionally, the auditors reviewed many different cases that may have led to exploits or bugs in the code.
Here are our assessment and recommendations, in order of importance.

Issues

Critical

None

High

None

Medium

None

Low

[L01] Documentation about inactive threshold is inconsistent with the code

The comments in the code state that an account can only be inactive if “it’s been over INACTIVE_THRESHOLD_DAYS since last activity”.

The implementation, however, allows for inactivity if the number of days is either equal to or over the threshold. Similarly, in the `calcStorageFee` function, `daysInactive` is checked for equality against the threshold as well. Consider updating the code to reflect the comment.

Additionally, the token specification states "If an address has not interacted with the contract (originated a transaction to the contract) for 3 years, the contract is allowed to mark the account "inactive"." Depending on the action taken to resolve this issue, this may need to be updated as well.

Update: The documentation has been updated.

Notes

- Consider adding the `index` keyword to various parameters in the events for easier off-chain analysis.

  Update: This has been resolved in 3b02840

- The constructor is missing docstrings for `feeAddress` and `redeemAddress`. Consider adding these for completion.

  Update: This has been resolved in 0a6a742

- The code seems to use both `uint` and `uint256`, even in parameters within the same function. `uint` is and alias for `uint256`, so they behave identically. Consider using one or the other throughout the code to help readability.

  Update: This has been resolved in e03f894

- In the creation of the contract, the constructor sets the `feeEnforcer` address to be exempt from fees. The `feeEnforcer` address can be updated in the `setFeeEnforcer` function, but this new address is not set as fee exempt. Consider atomically setting the new `feeEnforcer` address as exempt from fees when updating the address in the `setFeeEnforcer` function.

  Update: This has been resolved in 8bdfe41

- There is a redundant declaration of using `SafeMath`, since `uint` is an alias for `uint256`. Consider removing one of these declarations.

  Update: This has been resolved in 6e5be06

- Consider renaming `_inactiveFee` to `_calculateInactiveFee` in order to be consistent and clear with the intentions of the code.

  Update: This has been resolved in 4a507d5

- This comment states “Round up by 1 token”, but there does not appear to be any rounding being done. Consider removing this comment if there is no rounding being done or move it to the correct location in the code if it is being done.

  Update: This has been resolved in 42dfb0b

- As it stands, `balanceOf()` returns the total number of tokens that *can* be sent from an account, after fees. This value may be less than the true full balance of an account, which is returned from `balanceOfNoFees()`.
  While this does abide by the ERC20 standard, it will result in balance inconsistencies between event totals and `balanceOf()` calls. With this architecture, `balanceOf()` may return a different balance than an account's balance that has been calculated by adding and subtracting each transfer event. Services that use a block explorer to display and use token values will display and use the account's true balance (the equivalent of `balanceOfNoFees()`). If the service in question uses this value to perform actions, such as "send all tokens", the transaction will fail and will cause confusion for holders of the token.

  Additionally some services may use `balanceOf()` to display account balances while some use past events. A user will seemingly have two different account balances on these two different services, which may also cause confusion.

- Newly set internal account addresses are set as fee exempt atomically within the setting function. The old addresses, however are never unset from the fee exempt status.

  If there is not a case where internal accounts can be the same address, consider atomically unsetting the old addresses in the same call that the new addresses are set.

- The arithmetic on line 728 in the `calcSendAllBalance` function can be simplified to:

  ```
  uint256 transFee =
  sendAllAmount.mul(_transferFeeBasisPoints).div(BASIS_POINTS
  _MULTIPLIER);
  ```

  Update: This has been resolved in c75e195

- The arithmetic on line 751 in the `calcTransferFee` function can be simplified to:

```
    return
value.mul(_transferFeeBasisPoints).div(BASIS_POINTS_MULTIPLIER)
```

Update: This has been resolved in c75e195

- There are a number of instances in the contract where magic constants are used. All of these instances should be declared as a variable, and this variable should be used throughout the code. Some locations where this applies:
    - The use of 40000000000 in the fee calculation
    - The use of 20000000000 in the inactive fee calculation
    - The use of 200 to represent the max number of years to be collected
    - The use of 146000to represent the minimum fee
    - The use of 10 to represent the max number of basis points
    - Various uses of the number 365 throughout the code
    - Various uses of the number 86400 throughout the code

  Update: This has been resolved in 361ce6c

- There are a few misspellings throughout the code. The following are some locations where this was noticed and should be fixed:
    - *Can only unlock amount of gold if **it's** would leave the*
    - *Cache has the ability to force collecting storage and inactivity fees, but in the event **and** address was missed, can we still detect if the account was inactive when they next transact*
    - *we can start deducting chunks off the address so that full balance can be **recooped** after 200 years. This is likely*
    - *User now owes storage fees on entire balance, as if they held tokens for 1 **years**, instead of resetting clock to now.*
    - ***]4]** final to balance*

  Update: This has been resolved in d3d216c

Conclusion

No critical, high, or medium issues were found. Some changes were proposed to follow best practices and reduce the potential attack surface. Overall, the code was well written and void of any serious flaws.

Auditors:

Chris Whinfrey

[illegible]

September 3, 2019

Shane Fontaine

[illegible]

September 3, 2019