MONO
CEROS
ALPHA

PROJECT

Halo Rewards

CLIENT

Halo DAO

DATE

June 2021

REVIEWERS

Daniel Luca
@cleanunicorn

Andrei Simion
@andreiashu

Table of Contents

Details

- **Client** Halo DAO
- **Date** June 2021
- **Lead reviewer** Daniel Luca (@cleanunicorn)
- **Reviewers** Daniel Luca (@cleanunicorn), Andrei Simion (@andreiashu)
- **Repository**: Halo Rewards
- **Initial commit hash** `1cff704a4065256f30bb50858626aa7ef5552268`
- **Final commit hash** `04f83da3ebe42a1df6692e1e3080fab9ed051920`
- **Technologies**
  - Solidity
  - Node.JS

Issues Summary

SEVERITY	OPEN	CLOSED
Informational	0	1
Minor	0	4
Medium	0	1
Major	0	0

Executive summary

This report represents the results of the engagement with **Halo DAO** to review **Halo Rewards**.

The review was conducted over the course of **1 week** from **June 28 to July 2, 2021**. A total of **10 person-days** were spent reviewing the code.

Week 1

Before the beginning of the first week, we set up a kickoff meeting on Friday to ensure we have all the details to start the audit. This meeting was recorded and shared with all the parties involved in this review.

We started the following week by creating a code snapshot at commit hash `1cff704a4065256f30bb50858626aa7ef5552268`. We started using our arsenal of tools to make the graphs and descriptions of the contracts in scope. We proceeded to review the rest of the code manually.

Because there were no major issues, we gave access to the development team to the minor issues we found. This gave them the time and opportunity to provide fixes for them during the current week. We continued to review the code, but we found no major issues.

Towards the end of the week, we set up a meeting to present the current state of the report.

We also received a new commit hash `04f83da3ebe42a1df6692e1e3080fab9ed051920` that included fixes for the presented issues and included that in the report.

Scope

The initial review focused on the Halo Rewards repository, identified by the commit hash `1cff704a4065256f30bb50858626aa7ef5552268`.

We focused on manually reviewing the codebase, searching for security issues such as, but not limited to, re-entrancy problems, transaction ordering, block timestamp dependency, exception handling, call stack depth limitation, integer overflow/underflow, self-destructible contracts, unsecured balance, use of origin, costly gas patterns, architectural problems, code readability.

Includes:

- HaloHalo.sol
- HaloToken.sol
- AmmRewards.sol
- RewardsManager.sol

Issues

Consider checking for duplicate LP tokens when adding a new one in `AmmRewards`

Status Fixed Severity Medium

Description

An owner can add a new LP token to the list by calling `add`.

code/contracts/AmmRewards.sol#L85

```
    function add(uint256 allocPoint, IERC20 _lpToken, IRewarder _rewarder) public onlyOwner {
```

There's a comment stating that rewards will suffer if the same LP token is added multiple times in the list.

code/contracts/AmmRewards.sol#L81

```
    /// DO NOT add the same LP token more than once. Rewards will be messed up if you do.
```

Consider checking for duplicate tokens when adding a new one to make sure the system does not behave incorrectly.

Recommendation

If the number of tokens will not be over 10, consider looping over the existing tokens, making sure the new token does not match one of the existing ones.

```
contract CheckWithLoop {
    uint[] list;
    function addLoop(uint n) public {
        uint listLength = list.length;
        for (uint i; i < listLength; i++) {
            require(list[i] != n, "no duplicates");
        }
        list.push(n);
    }
}
```

If you expect to have more than 10 tokens on the list, a mapping can be used to check the existence of a token.

```
contract CheckWithMapping {
    uint[] list;
    mapping(uint => bool) listDups;

    function addNonDuplicate(uint n) public {
        require(listDups[n] == false, "no duplicates");
        list.push(n);
        listDups[n] = true;
    }
}
```

Simplify storage of `haloHaloContract` in `RewardsManager`

Status Fixed Severity Minor

Description

During the `RewardsManager` deploy a few storage variables are set.

code/contracts/RewardsManager.sol#L26-L37

```
  constructor(
    uint256 _initialVestingRatio, //in BASIS_POINTS, multiplied by 10^4
    address _rewardsContract,
    address _haloHaloContract,
    IERC20 _halo
  ) public {
    vestingRatio = _initialVestingRatio;
    rewardsContract = _rewardsContract;
    haloHaloContract = _haloHaloContract;
    halohalo = HaloHalo(haloHaloContract);
    halo = _halo;
  }
```

The variable storing the address for the HaloHalo contract is received as `_haloHaloContract` in the arguments.

code/contracts/RewardsManager.sol#L29

```
    address _haloHaloContract,
```

The contract's address is stored in `haloHaloContract`

code/contracts/RewardsManager.sol#L34

```
    haloHaloContract = _haloHaloContract;
```

And it is also stored in a separate variable as the `HaloHalo` interface

code/contracts/RewardsManager.sol#L35

```
    halohalo = HaloHalo(haloHaloContract);
```

However, both of the values are identical in the contract's storage.

Consider this simplified example

```
interface SomeContract {
    function doSomething() external;
}

contract Save {
    SomeContract someContract;
    address someContractAddress;

    constructor(address _someContractAddress) {
        someContract = SomeContract(_someContractAddress);
        someContractAddress = _someContractAddress;
    }
}
```

The contract `Save` receives an argument on deploy and proceeds to save the address both as an address (as `someContractAddress`), but also as a `SomeContract` interface (as `someContract`).

We deployed this contract locally (on a Ganache instance) and checked the both of the storage slots in the contract after a successful deploy.

We connected to the local instance with Hitomi and we're dropped in a web3 enabled local console.

```
$ hitomi http://localhost:8545
Starting Hitomi v0.3.2.
```

```
Connected to http://localhost:8545.

Node version: EthereumJS TestRPC/v2.13.2/ethereum-js
Enode path: None
Protocol version: 99
Chain ID: 1337
Block number: 0
Mining: False (0 hash rate)
Syncing: False
```

We proceeded to check the first storage slot representing `SomeContract someContract;`

```
>>> web3.eth.getStorageAt("0x6F3dC8C964d3F2ce4A1Ba8CD6Da6E7320A69CC6D", 0)
HexBytes('0x5b38da6a701c568545dcfcb03fcb875f56beddc4')
```

And the second storage slot representing `address someContractAddress;`

```
>>> web3.eth.getStorageAt("0x6F3dC8C964d3F2ce4A1Ba8CD6Da6E7320A69CC6D", 1)
HexBytes('0x5b38da6a701c568545dcfcb03fcb875f56beddc4')
```

We can see both values are identical. Solidity needs to cast an address as an interface to know which methods are available and how to use them. The generated assembly doesn't do anything specific to the initially provided address, this is just syntactic sugar.

Because both values are identical, we don't need to save both of them in separate storage slots, this makes the save (or update) and the read more expensive. Saving only one of them is better because, on read, it doesn't need to retrieve both of them and storage handing (reading and writing) is very expensive.

Recommendation

Use only one of the storage slots to save either the HaloHalo contract and cast when needed to either access the address or the interface.

Capping the minting process can be simplified

Status Fixed Severity Minor

Description

When the HaloToken is deployed, there are 2 state variables set:

code/contracts/HaloToken.sol#L24-L25

```
        canMint = true;
        isCappedFuncLocked = false;
```

The `canMint` state variable is checked when the owner tries to mint additional tokens.

code/contracts/HaloToken.sol#L40-L41

```
    function mint(address account, uint256 amount) external onlyOwner {
        require(canMint == true, "Total supply is now capped, cannot mint more");
```

The owner can renounce this power by calling `setCapped`.

code/contracts/HaloToken.sol#L28-L30

```
    /// @notice Locks the cap and disables mint func.
    /// @dev Should be called only once. Allows owner to lock the cap and disable mint function.
    function setCapped() external onlyOwner {
```

When `setCapped` is called, the state variable `isCappedFuncLocked` is checked to be false.

code/contracts/HaloToken.sol#L31

```
        require(isCappedFuncLocked == false, "Cannot execute setCapped more than once.");
```

Once the check passes, `canMint` is set to `false`, blocking future token minting.

code/contracts/HaloToken.sol#L32

```
        canMint = false;
```

And `isCappedFuncLocked` is set to `true`; this prevents a second call to `setCapped`.

However, the 2 state variables are always synchronized. They can have the value `true` or `false`. And they only exist in these 2 states:

1. Can mint tokens; can disable token minting

```
canMint = true
isCappedFuncLocked = false
```

2. Cannot mint tokens anymore; cannot re-enable token minting

```
canMint = false
isCappedFuncLocked = true
```

The code can be simplified if one of the 2 state variables is removed, the other one's value can be deduced by negating the remaining one.

Recommendation

Remove `isCappedFuncLocked` and use `canMint` to limit the `setCapped` execution.

`haloHaloAmount` should be renamed to `haloAmount`

Status Fixed Severity Minor

Description

When a user wants to unstake their tokens, they need to call `leave`.

code/contracts/HaloHalo.sol#L44-L46

```
// Claim HALOs from HALOHALOs.
// Unlocks the staked + gained Halo and burns HALOHALO
function leave(uint256 _share) public {
```

The amount of tokens to be unlocked is calculated and saved in a local variable `haloHaloAmount`:

code/contracts/HaloHalo.sol#L49-L51

```
  // Calculates the amount of Halo the HALOHALO is worth
  uint256 haloHaloAmount =
    _share.mul(halo.balanceOf(address(this))).div(totalShares);
```

This value should be named `haloAmount` because this value is then used to send the halo tokens back to the user.

code/contracts/HaloHalo.sol#L53

```
  halo.transfer(msg.sender, haloHaloAmount);
```

Recommendation

Rename the variable `haloHaloAmount` to `haloAmount`.

Can set immutable for `halo` in `HaloHalo`

Status Fixed Severity Minor

Description

The Halo token contract is set when the `HaloHalo` contract is deployed.

code/contracts/HaloHalo.sol#L16

```
  halo = _halo;
```

The `halo` variable is defined as a state variable.

code/contracts/HaloHalo.sol#L10

```
IERC20 public halo;
```

Because this state variable is never changed, it can be defined as `immutable` for a significant gas cost.

Recommendation

Set `halo` as `immutable`.

The number of minted tokens might not be the expected one

Status Acknowledged Severity Informational

Description

A user can stake HALO tokens for HALOHALO tokens by calling the `enter` method.

code/contracts/HaloHalo.sol#L23

```
function enter(uint256 _amount) public {
```

The number of HALO tokens is retrieved:

code/contracts/HaloHalo.sol#L27-L28

```
    // Gets the amount of Halo locked in the contract
    uint256 totalHalo = halo.balanceOf(address(this));
```

Next, the total number of shares is retrieved, which matches the number of minted tokens:

code/contracts/HaloHalo.sol#L29-L30

```
    // Gets the amount of HALOHALO in existence
    uint256 totalShares = totalSupply();
```

If this is the first time someone enters the stake, the conditional is true, and a ratio of 1:1 is minted based on the amount entering the contract.

code/contracts/HaloHalo.sol#L31-L33

```
    // If no HALOHALO exists, mint it 1:1 to the amount put in
    if (totalShares == 0 || totalHalo == 0) {
        _mint(msg.sender, _amount);
```

i.e., An actor entering with 100 HALO tokens will receive 100 HALOHALO tokens.

If tokens were already minted and HALO tokens exist in the contract, a formula is used to calculate how many tokens should be minted.

code/contracts/HaloHalo.sol#L35-L37

```
        // Calculate and mint the amount of HALOHALO the Halo is worth. The ratio will change over
        uint256 haloHaloAmount = _amount.mul(totalShares).div(totalHalo);
        _mint(msg.sender, haloHaloAmount);
```

Let's assume an actor is the first one to stake tokens in the contract. They send 100 HALO tokens to the contact. Because they are the first ones, the contract mints 1:1 tokens, effectively 100 HALOHALO tokens.

The second actor enters with 100 HALO too. This time, the formula is activated.

$$hh_{amount} = \frac{h_{enter} * hh_{minted}}{h_{locked}}$$

A number of $hh_{amount} = 100$ get minted.

This is because the formula uses the amount of tokens which the user wants to lock ($h_{enter} = 100$), the number of tokens already minted ($hh_{minted} = 100$, from the previous user) and the total locked tokens in the contract ($h_{locked} = 100$).

minted tokens = $\frac{100*100}{100} = 100$

If everything works well, a ratio of 1:1 will always be respected. However, if anyone sends HALO tokens to the contract, the 1:1 ratio is forever changed for all users staking tokens.

Let's assume that after the 2 users deposited 100 HALO tokens each, and they received 100 HALOHALO tokens, a malicious user sends 100 HALO tokens to the contract without calling `enter`, but by using the `transfer` method.

We should be aware of the current state of the contract right now.

$$hh_{minted} = 200$$

$$h_{locked} = 300$$

The number of h_{locked} is equal to 300 because the actual balance of the token is retrieved by using the `balanceOf` method, not an internal accounting method.

If a 3rd user wants to lock 100 HALO tokens, a different ratio of HALOHALO tokens will be minted for them. Using the formula, we obtain the number of minted tokens.

Minted tokens = $\frac{100*200}{300} = 66.66$

This allows a malicious actor to manipulate the ratio of minted tokens.

It's important to note that the "unstake" mechanism is not affected negatively because the current ratio is used to return the tokens back to the users. Also, the actor who sends the tokens doesn't seem to gain anything from sending the tokens; in fact, they lose the tokens they sent to the contract, and the users being part of the system (who used `enter`) gain the lost tokens.

Artifacts

Surya

Sūrya is a utility tool for smart contract systems. It provides a number of visual outputs and information about the structure of smart contracts. It also supports querying the function call graph in multiple ways to aid in the manual inspection and control flow analysis of contracts.

Files Description Table

File Name	SHA-1 Hash
./AmmRewards.sol	49b19808853abe8c5c8bfb439d83ee196314a616
./HaloHalo.sol	4d1f4fa884b7499e9ab33fcb423b88ea05ac2242
./HaloToken.sol	aadb215941a561bc6f3005d0f19a09a7775476bf
./RewardsManager.sol	c0ee996398307a8e777783f8b1b1ef22af2e11c1

Contracts Description Table

Contract	Type	Bases	
└	**Function Name**	**Visibility**	**Mu**
AmmRewards	Implementation	ReentrancyGuard, Ownable	
└		Public ❗️	
└	poolLength	Public ❗️	
└	add	Public ❗️	
└	set	Public ❗️	
└	setRewardTokenPerSecond	External ❗️	
└	pendingRewardToken	External ❗️	
└	massUpdatePools	External ❗️	
└	updatePool	Public ❗️	
└	deposit	Public ❗️	
└	withdraw	Public ❗️	
└	harvest	Public ❗️	
└	withdrawAndHarvest	Public ❗️	
└	emergencyWithdraw	Public ❗️	
└	setRewardsManager	Public ❗️	
HaloHalo	Implementation	ERC20	
└		Public ❗️	
└	enter	Public ❗️	
└	leave	Public ❗️	
└	getCurrentHaloHaloPrice	Public ❗️	
HaloToken	Implementation	ERC20, ERC20Burnable, Ownable	
└		Public ❗️	
└	setCapped	External ❗️	
└	mint	External ❗️	

Contract	Type	Bases
RewardsManager	Implementation	Ownable
L		Public ❗
L	releaseEpochRewards	External ❗
L	setVestingRatio	External ❗
L	setRewardsContract	External ❗
L	setHaloHaloContract	External ❗
L	getVestingRatio	External ❗
L	getRewardsContract	External ❗
L	getHaloHaloContract	External ❗
L	transferToHaloHaloContract	Internal 🔒
L	convertAndTransferToRewardsContract	Internal 🔒

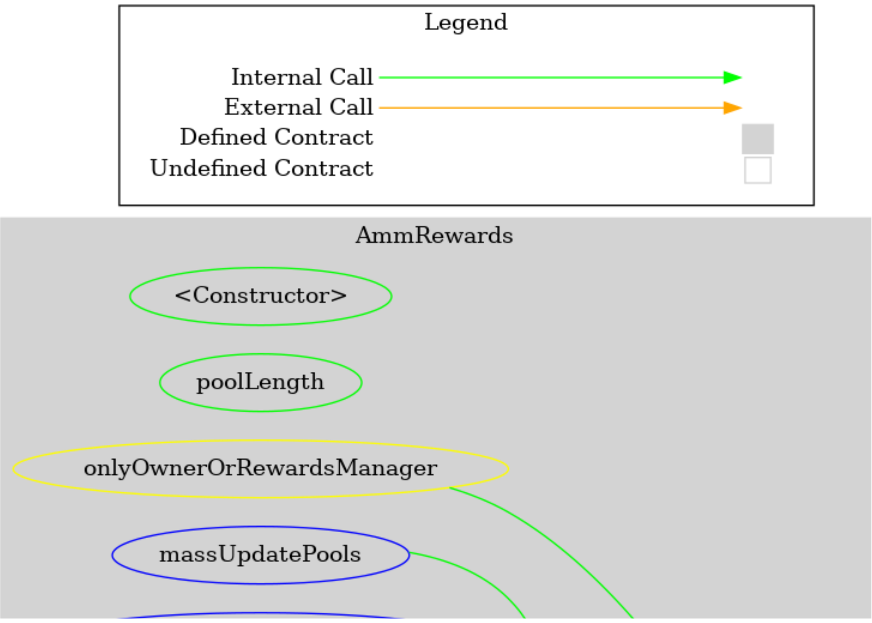
Legend

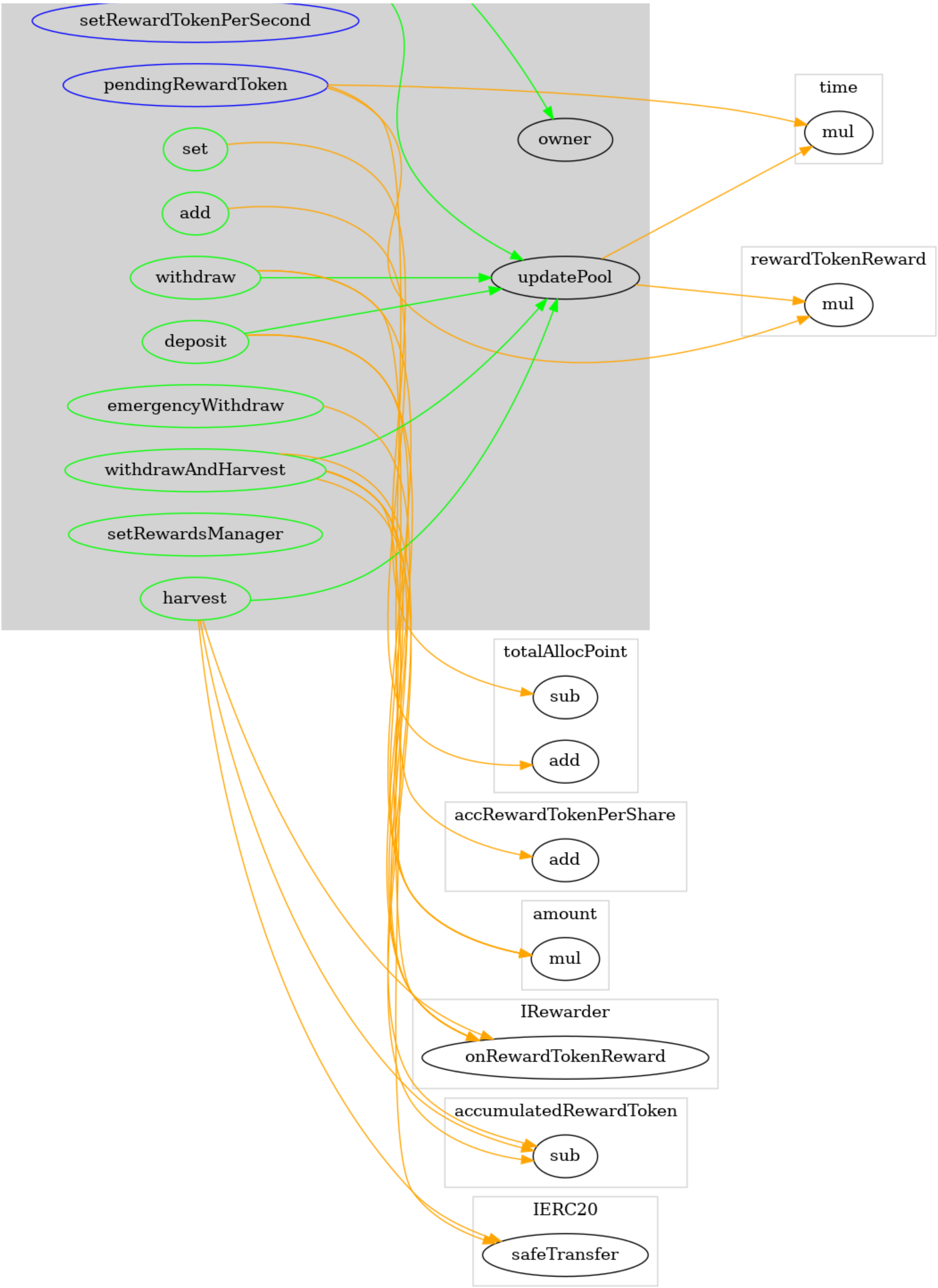
Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

Graphs

AmmRewards

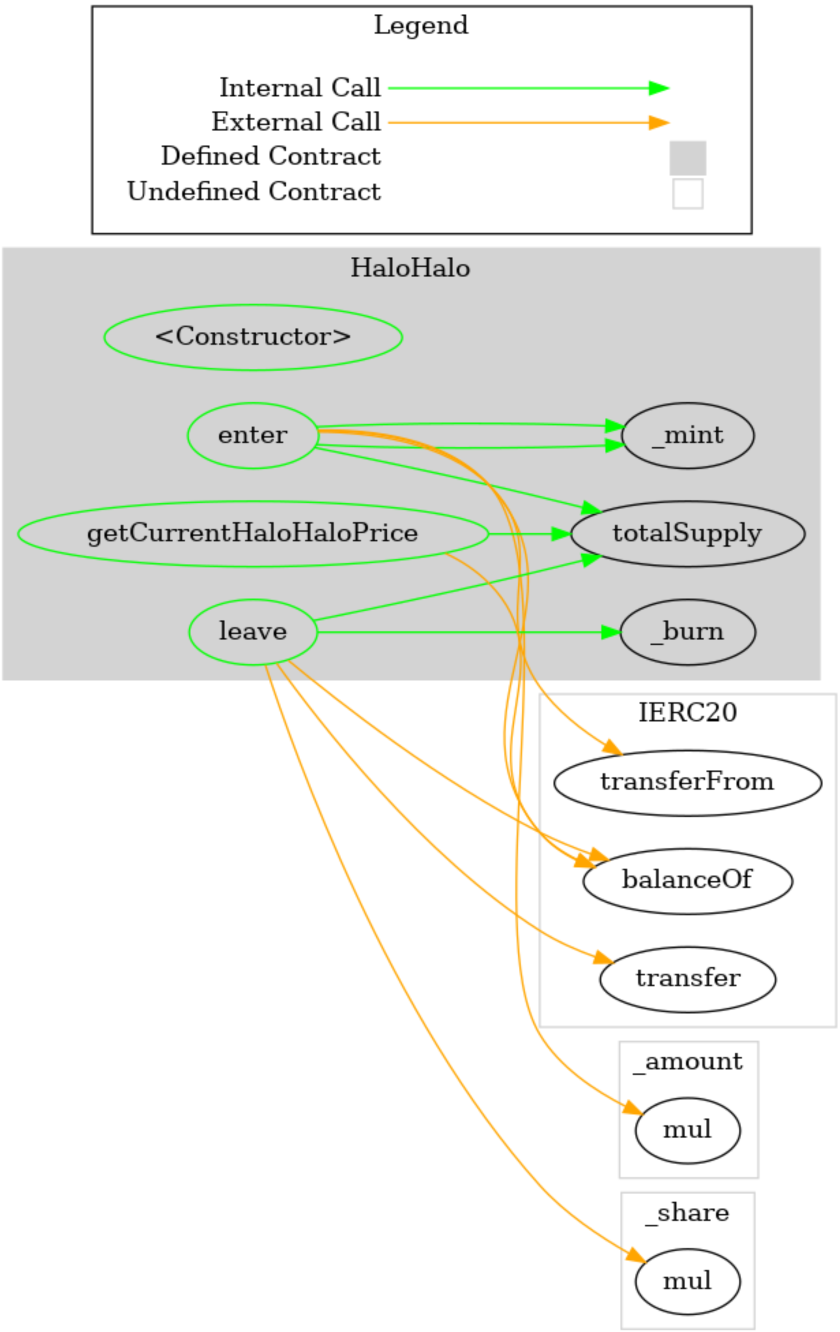
```
surya graph AmmRewards.sol | dot -Tpng > ./static/AmmRewards_graph.png
```





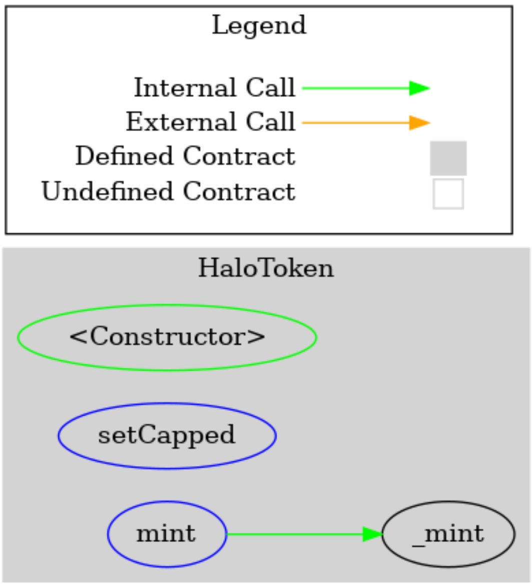
HaloHalo

```
surya graph HaloHalo.sol | dot -Tpng > ./static/HaloHalo_graph.png
```



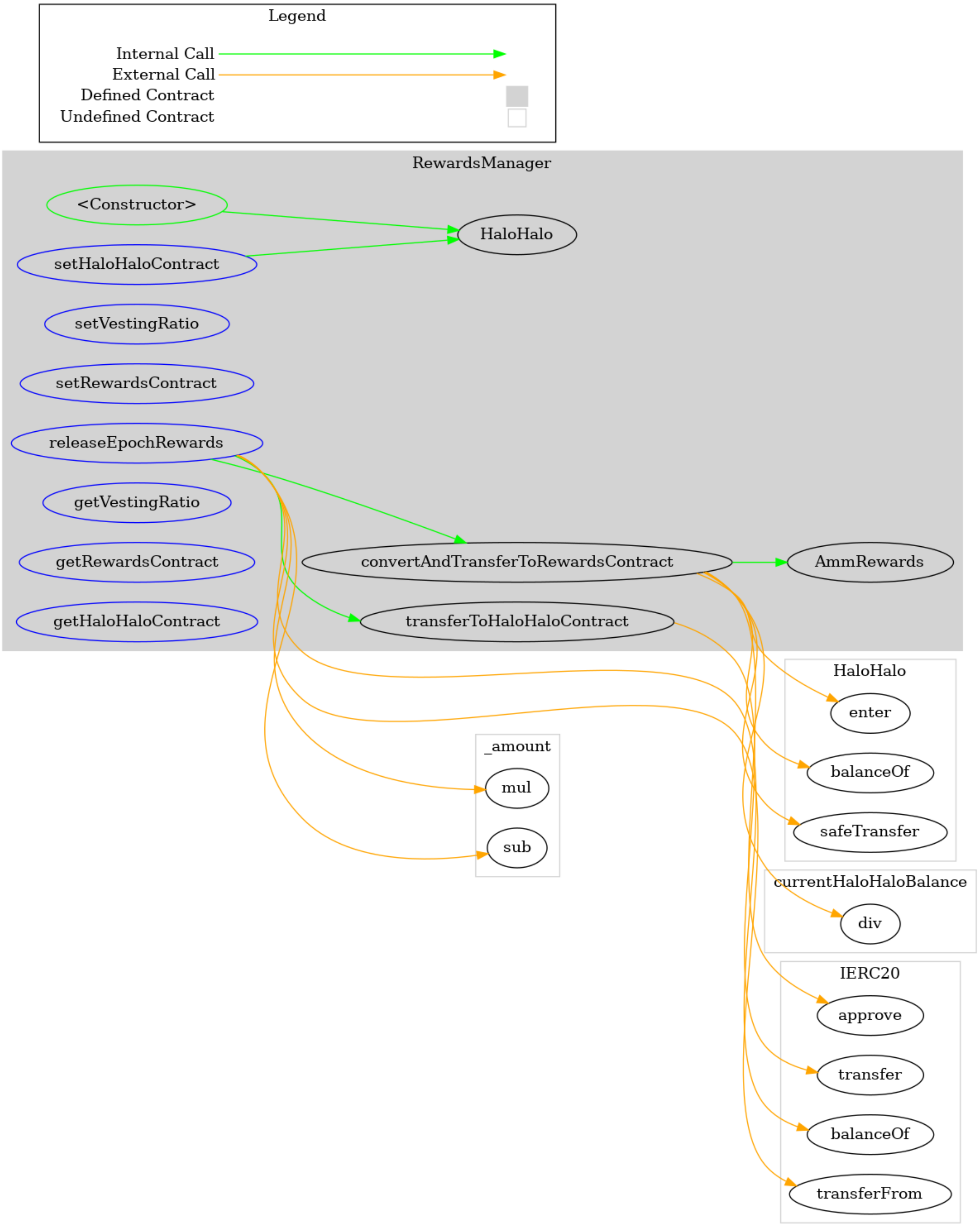
HaloToken

```
surya graph HaloToken.sol | dot -Tpng > ./static/HaloToken_graph.png
```

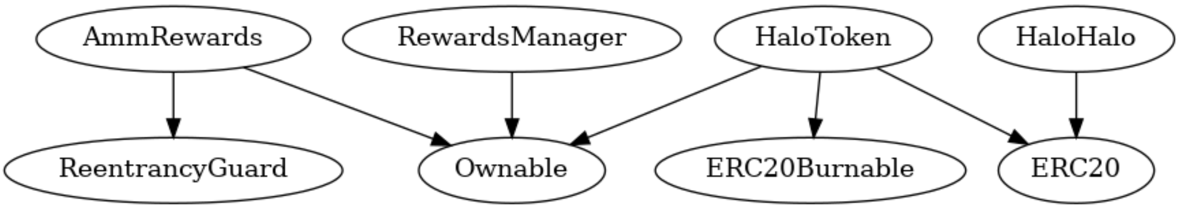


RewardsManager

```
surya graph RewardsManager.sol | dot -Tpng > ./static/RewardsManager_graph.png
```



Inheritance



Describe

```
$ npx surya describe *.sol

 +  AmmRewards (ReentrancyGuard, Ownable)
    - [Pub] <Constructor> #
    - [Pub] poolLength
    - [Pub] add #
       - modifiers: onlyOwner
    - [Pub] set #
       - modifiers: onlyOwner
    - [Ext] setRewardTokenPerSecond #
       - modifiers: onlyOwnerOrRewardsManager
    - [Ext] pendingRewardToken
    - [Ext] massUpdatePools #
    - [Pub] updatePool #
    - [Pub] deposit #
    - [Pub] withdraw #
    - [Pub] harvest #
    - [Pub] withdrawAndHarvest #
    - [Pub] emergencyWithdraw #
    - [Pub] setRewardsManager #
       - modifiers: onlyOwner

 +  HaloHalo (ERC20)
    - [Pub] <Constructor> #
    - [Pub] enter #
    - [Pub] leave #
    - [Pub] getCurrentHaloHaloPrice

 +  HaloToken (ERC20, ERC20Burnable, Ownable)
    - [Pub] <Constructor> #
       - modifiers: ERC20
    - [Ext] setCapped #
       - modifiers: onlyOwner
    - [Ext] mint #
       - modifiers: onlyOwner

 +  RewardsManager (Ownable)
    - [Pub] <Constructor> #
    - [Ext] releaseEpochRewards #
       - modifiers: onlyOwner
    - [Ext] setVestingRatio #
       - modifiers: onlyOwner
    - [Ext] setRewardsContract #
       - modifiers: onlyOwner
    - [Ext] setHaloHaloContract #
       - modifiers: onlyOwner
    - [Ext] getVestingRatio
    - [Ext] getRewardsContract
    - [Ext] getHaloHaloContract
    - [Int] transferToHaloHaloContract #
    - [Int] convertAndTransferToRewardsContract #
 ($) = payable function
 # = non-constant function
```

Coverage

Tests

```
$ yarn run test
yarn run v1.22.10
warning package.json: No license field
$ npx hardhat --network localhost test
BASIS_POINTS =  10000


  Amm Rewards
    PoolLength
      √ PoolLength should execute (187308 gas)
    Set
      √ Should emit event LogSetPool (285027 gas)
      √ Should revert if invalid pool
    Pending Reward Token
      √ Pending Reward Token should equal Expected Reward Token (408378 gas)
      √ When time is lastRewardTime (408378 gas)
    MassUpdatePools
      √ Should call updatePool (234554 gas)
      √ Updating invalid pools should fail
    Add
      √ Should add pool with reward token multiplier (187308 gas)
    UpdatePool
      √ Should emit event LogUpdatePool (234463 gas)
    Deposit
      √ Depositing 0 amount (290613 gas)
      √ Depositing into non-existent pool should fail
    Withdraw
      √ Withdraw 0 amount (255423 gas)
    Harvest
      √ Should give back the correct amount of Reward Token (509159 gas)
      √ Harvest with empty user balance (246015 gas)
    EmergencyWithdraw
      √ Should emit event EmergencyWithdraw (365663 gas)
    Admin functions
      √ Non-owner should not be able to add pool
      √ Owner should be able to add pool (187308 gas)
      √ Non-owner should not be able to set pool allocs (187308 gas)
      √ Owner should be able to set pool allocs (231511 gas)
      √ Non-owner should not be able to set rewardTokenPerSecond
      √ Owner should be able to set rewardTokenPerSecond (32504 gas)
    Set rewardTokenPerSecond
      √ Non-owner should not be able to set rewardTokenPerSecond
      √ RewardsManager should change rewardTokenPerSecond (167779 gas)
      √ Owner should be able to set rewardTokenPerSecond (32504 gas)

  Halo Token
===================Deploying Contracts=====================
halo token deployed
Minted initial HALO for owner account
Minted initial HALO for addr1 account
    Check Contract Deployment
      √ HaloToken should be deployed (54478 gas)
    I should be able to transfer HALO tokens
      √ Allow transfer (89841 gas)
    I should be able to mint HALO tokens and get the correct totalSupply
      √ Only owner should mint (72741 gas)
5e+25  HALO tokens owner balance
      √ When owner mints, the total supply should be equal to all wallet balance (74756 gas)
    I should not be allowed to mint if capped is already locked
      √ Only owner can execute setCapped (64598 gas)
      √ Should revert mint when capped is locked (27220 gas)
      √ Should revert setCapped func if it has been executed more than once (27220 gas)
    I should be able to burn HALO tokens and get the correct totalSupply
      √ Only account holder should burn (61594 gas)
      √ Only owner should burn users tokens (144521 gas)
4e+25 HALO tokens owner balance
      √ When user burns, the total supply should be equal to all wallet balance (62005 gas)
      √ Burn amount should not exceed wallet balance (34374 gas)

  HALOHALO Contract
===================Deploying Contracts=====================
halo token deployed
40000000 HALO minted to 0x959FD7Ef9089B7142B6B908Dc3A8af7Aa8ff0FA1

halohalo deployed
==========================================================


    Check Contract Deployments
      √ HaloToken should be deployed (37402 gas)
      √ Halohalo should be deployed (37402 gas)
    Earn vesting rewards by staking HALO inside halohalo
      √ Genesis is zero (37402 gas)
      √ Deposit HALO tokens to halohalo contract to receive halohalo (190239 gas)
      √ Calculates current value of HALOHALO in terms of HALO without vesting (105873 gas)
      √ Calculates current value of HALOHALO in terms of HALO after vesting (143239 gas)
      √ Claim staked HALO + bonus rewards from Halohalo and burn halohalo (73308 gas)
Minting HALO to be entered in the halohalo contract..

Minting 100 HALO to User A...
Minting 100 HALO to User B...
Minting 100 HALO to User C...
100 HALO deposited by User A to halohalo
Simulate releasing vested bonus tokens to halohalo from Rewards contract #1
100 HALO deposited by User B to halohalo
```

```
Simulate releasing vested bonus tokens to halohalo from Rewards contract #2
100 HALO deposited by User C to halohalo
Transfer to 0xB0201641d9b936eB20155a38439Ae6AB07d85Fbd approved
All users leave halohalo
Address 0 left
Address 1 left
Address 2 left
      √ HALO earned by User A > HALO earned by User B > HALO earned by User C (754048 gas)

  Rewards Manager
===================Deploying Contracts=====================
collateralERC20 deployed
halo token deployed
halohalo deployed
changedHaloHaloContract deployed
Set Rewards Manager contract.
Deployed Rewards Manager Contract address: 0x2Cc79B6860Fd7b58f0Fb56B4f448c13C7e898EC4
==========================================================


    Check Contract Deployments
      √ HaloToken should be deployed (46340 gas)
      √ Halohalo should be deployed (46340 gas)
      √ Lptoken should be deployed (46340 gas)
      √ Rewards Management Contract should be deployed (46340 gas)
    Admin functions can be set by the owner
      √ can set the vestingRatio if the caller is the owner (75061 gas)
      √ can not set the vestingRatio if the caller is not the owner (28636 gas)
      √ can not set the vesting ratio if vesting ratio is equal to zero (28636 gas)
      √ can set the rewards contract if the caller is the owner (57691 gas)
      √ can not set the rewards contract if the caller is not the owner (28636 gas)
      √ can not set the rewards contract if address parameter is address(0) (28636 gas)
      √ can set the halohalo contract if the caller is the owner (62872 gas)
      √ can not set the halohalo contract if the caller is not the owner (34236 gas)
      √ can not set the halohalo contract if the address parameter is address(0) (28636 gas)
    Released HALO will be distributed 80% to the rewards contract converted to DESRT and 20% wil
      √ Release rewards in Epoch 0, HALOHALO priced to one at the end (347535 gas)
      √ Release rewards in Epoch 1, HALOHALO priced to 1.25 at the end  (400100 gas)
      √ fails if the caller is not the owner (98301 gas)

·----------------------------------------------|----------------------------|-------------|-----
|             Solc version: 0.6.12             ·  Optimizer enabled: false  ·  Runs: 200  ·  Blo
···············································|····························|·············|·····
|  Methods
···················|···························|··············|·············|·············|·····
|  Contract        ·  Method                   ·  Min         ·  Max        ·  Avg        ·  # c
···················|···························|··············|·············|·············|·····
|  AmmRewards      ·  add                      ·      187296  ·     187308  ·     187307  ·
···················|···························|··············|·············|·············|·····
|  AmmRewards      ·  deposit                  ·       73501  ·     119032  ·     103855  ·
···················|···························|··············|·············|·············|·····
|  AmmRewards      ·  emergencyWithdraw        ·           -  ·          -  ·      29531  ·
```

```
···················|···························|··············|·············|·············|·····
|  AmmRewards      ·  harvest                  ·       58707  ·      81390  ·      70049  ·
···················|···························|··············|·············|·············|·····
|  AmmRewards      ·  massUpdatePools          ·           -  ·          -  ·      47258  ·
···················|···························|··············|·············|·············|·····
|  AmmRewards      ·  set                      ·       38843  ·      58876  ·      49000  ·
···················|···························|··············|·············|·············|·····
|  AmmRewards      ·  setRewardsManager        ·       46328  ·      46340  ·      46338  ·
···················|···························|··············|·············|·············|·····
|  AmmRewards      ·  setRewardTokenPerSecond  ·           -  ·          -  ·      32504  ·
···················|···························|··············|·············|·············|·····
|  AmmRewards      ·  updatePool               ·       47155  ·      72234  ·      63874  ·
···················|···························|··············|·············|·············|·····
|  AmmRewards      ·  withdraw                 ·       68115  ·      91625  ·      75952  ·
···················|···························|··············|·············|·············|·····
|  HaloHalo        ·  enter                    ·       51896  ·     105873  ·      84537  ·
···················|···························|··············|·············|·············|·····
|  HaloHalo        ·  leave                    ·       35930  ·      56859  ·      44302  ·
···················|···························|··············|·············|·············|·····
|  HaloToken       ·  approve                  ·       46916  ·      46964  ·      46926  ·
···················|···························|··············|·············|·············|·····
|  HaloToken       ·  burn                     ·           -  ·          -  ·      34374  ·
···················|···························|··············|·············|·············|·····
|  HaloToken       ·  burnFrom                 ·           -  ·          -  ·      27631  ·
···················|···························|··············|·············|·············|·····
|  HaloToken       ·  increaseAllowance        ·           -  ·          -  ·      47237  ·
···················|···························|··············|·············|·············|·····
|  HaloToken       ·  mint                     ·       37366  ·      71578  ·      59531  ·
···················|···························|··············|·············|·············|·····
|  HaloToken       ·  setCapped                ·           -  ·          -  ·      27220  ·
···················|···························|··············|·············|·············|·····
|  HaloToken       ·  transfer                 ·       35279  ·      35363  ·      35335  ·
···················|···························|··············|·············|·············|·····
|  LpToken         ·  approve                  ·       29792  ·      46928  ·      43974  ·
···················|···························|··············|·············|·············|·····
|  LpToken         ·  mint                     ·           -  ·          -  ·      71335  ·
···················|···························|··············|·············|·············|·····
|  LpToken         ·  transfer                 ·           -  ·          -  ·      52417  ·
···················|···························|··············|·············|·············|·····
|  RewardsManager  ·  releaseEpochRewards      ·       98301  ·     215393  ·     199781  ·
···················|···························|··············|·············|·············|·····
|  RewardsManager  ·  setHaloHaloContract      ·       28636  ·      34236  ·      29756  ·
···················|···························|··············|·············|·············|·····
|  RewardsManager  ·  setRewardsContract       ·           -  ·          -  ·      29055  ·
···················|···························|··············|·············|·············|·····
|  RewardsManager  ·  setVestingRatio          ·           -  ·          -  ·      28721  ·
···················|···························|··············|·············|·············|·····
|  Deployments                                 ·                                          ·  % o
···············································|··············|·············|·············|·····
|  AmmRewards                                  ·     3165941  ·    3165953  ·    3165952  ·
···············································|··············|·············|·············|·····
|  CollateralERC20                             ·     1750829  ·    1750913  ·    1750871  ·
```

```
···············································|··············|·············|·············|·····
|  HaloHalo                                    ·      1752202  ·      1752226  ·      1752224  ·
···············································|··············|·············|·············|·····
|  HaloToken                                   ·            -  ·            -  ·      1910204  ·
···············································|··············|·············|·············|·····
|  LpToken                                     ·            -  ·            -  ·      1750829  ·
···············································|··············|·············|·············|·····
|  RewardsManager                              ·      1651492  ·      1651516  ·      1651512  ·
·----------------------------------------------|--------------|-------------|-------------|-----


  59 passing (36s)
```

License

This report falls under the terms described in the included LICENSE.