Celsius - WrappedToken

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **November 18th, 2021 - December 3rd, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 4

CONTACTS 4

1 EXECUTIVE OVERVIEW 5

1.1 INTRODUCTION 6

1.2 AUDIT SUMMARY 6

1.3 TEST APPROACH & METHODOLOGY 6

RISK METHODOLOGY 7

1.4 SCOPE 9

2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 12

3 FINDINGS & TECH DETAILS 13

3.1 (HAL-01) NON-STANDARD INITIALIZATION - MEDIUM 15

Description 15

Code Location 15

Risk Level 15

Recommendation 16

Remediation Plan 16

3.2 (HAL-02) UNEXPECTED FLASH-LOAN FLOW - LOW 17

Description 17

Code Location 18

Risk Level 19

Recommendation 19

Remediation Plan 19

3.3 (HAL-03) RE-ENTRANCY ALLOWED BY FLASH LOANS - LOW 21

Description 21

Proof of Concept 21

Risk Level 21

Recommendation 22

Remediation Plan 22

3.4 (HAL-04) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL 23

Description 23

Code Location 23

Risk Level 25

Recommendation 25

Remediation Plan 25

3.5 (HAL-05) UNUSED IMPORTS - INFORMATIONAL 26

Description 26

Code Location 26

Risk Level 26

Recommendation 26

Remediation Plan 27

4 MANUAL TESTING 28

4.1 Capabilities 29

Blacklistable 29

Mintable 29

Revocable 31

RevocableToAddress 31

4.2 WrappedTokenV1 31

5 Inheritance initialization and call checks 33

5.1 Initializers 35

5.2 Sub-inheritance 36

5.3 Call Graph 40

6 AUTOMATED TESTING 41

6.1 STATIC ANALYSIS REPORT 42

Description 42

Slither results 42

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	11/20/2021	Ferran Celades
0.2	Document Edits	12/01/2021	Ferran Celades
0.3	Draft Review	12/01/2021	Gabi Urrutia
1.0	Remediation Plan	12/16/2021	Ferran Celades
1.1	Remediation Plan Review	12/16/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Ferran Celades	Halborn	Ferran.Celades@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Celsius engaged Halborn to conduct a security audit on their smart contracts beginning on November 18th, 2021 and ending on December 3rd, 2021. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were mostly addressed by the Celsius team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy with regards to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process,and implementation, automated testing techniques help enhance coverage of the bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Static Analysis of security for scoped contract, and imported functions (Slither)
- Testnet deployment (Brownie, Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.

1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:
The security assessment was scoped to the following smart contracts:

- ERC1404.sol
- Migrations.sol
- Proxy.sol
- WrappedTokenV1.sol
- capabilities/Blacklistable.sol
- capabilities/Burnable.sol
- capabilities/Mintable.sol
- capabilities/Pausable.sol
- capabilities/Proxiable.sol
- capabilities/Revocable.sol
- capabilities/RevocableToAddress.sol
- capabilities/Whitelistable.sol
- roles/BlacklisterRole.sol
- roles/BurnerRole.sol
- roles/MinterRole.sol
- roles/OwnerRole.sol
- roles/PauserRole.sol
- roles/RevokerRole.sol
- roles/WhitelisterRole.sol
- All contracts inherited by these contracts

Those are the checksums (SHA-256) for the audited files **before** the remediation:

Listing 1

```
b8789d8d6e6b7825346ecb66088e862d89e8219290f03d59bdb1e51c815fd8c6
    contracts/ERC1404.sol
d47816a279f9ad0ebad434e8306bab82feaf134d316441b5165cb226f708cf00
    contracts/Migrations.sol
4fcb7cf2c06195f0ac2547ec44af3ace7cd437628b7f752356353da2883ab355
    contracts/Proxy.sol
b9808a73dcbb83902683c04b7dd9695657877c99eec853689863523bfbabf456
```

```
    contracts/WrappedTokenV1.sol
5592236267fc2596fa33b0ba1aa5ce3e10708e86554023b245119d6ea577f44b
    contracts/capabilities/Blacklistable.sol
8d53e471adf280732c626ba8ebadd7ee50574cdd0ec1eae3980840c00f51108b
    contracts/capabilities/Burnable.sol
fc6b790a942f7f512370f65b845d7ea52736b7bdfcc3d75409b5df853c4ea6fa
    contracts/capabilities/Mintable.sol
2c02b47100cd7b793435bb2ca9ed53674f8062f1e0ac3cbb87f2f14e101d3fa2
    contracts/capabilities/Pausable.sol
6284c50dbdc0f5a4c00787f4332b53f6a4d2b8a2ac2a076b3c4eaabd3b17417a
    contracts/capabilities/Proxiable.sol
4e62f221fd511ddab1f56c29d015598497fccd4773896bffc29c38e793fd0536
    contracts/capabilities/Revocable.sol
89da6b042991cec8732cf43aaaf9c75ef911a6ed47852290e38eb1d5dfc8134a
    contracts/capabilities/RevocableToAddress.sol
d97701eeb162cef1f565d0553ec6fc954d556b72b9acd33d498cdfa8aa030c95
    contracts/capabilities/Whitelistable.sol
d529e568ebc52509a548bd83f0f57f939406994486df3dd06646385001d5f806
    contracts/roles/BlacklisterRole.sol
f07c855f01e8d31a7e8c750c9457839d6c0d0f1b390538cc45b4ca44c37a0927
    contracts/roles/BurnerRole.sol
323774121ffe2af163c08d1799d8ad78c9375ccafc884a6e613a0138182cabc0
    contracts/roles/MinterRole.sol
4aa0fe0a6e494cd48f5677302b5bb0a4dbcb55c49056d0f59aef7072ff8c63a2
    contracts/roles/OwnerRole.sol
0c42af32e9363d8b0c8e6537ecfb9c4e75d1918c5d37300d8f3550bb369dea27
    contracts/roles/PauserRole.sol
c913aa3964c27bbd37f525903f3568fe4833cdce7f8ea36cb242182d1e8f351e
    contracts/roles/RevokerRole.sol
f9d6b9d1e6ce67866edafbef5d92dfd6588dcfafb70bac06cd4a9ac7f34b40be
    contracts/roles/WhitelisterRole.sol
```

Those are the checksums (SHA-256) for the audited files **after** the remediation:

Listing 2

```
c8479966ee562ffb6a41ca8a4a3af1561054ef952352a994ccf8e7ed470aee70
    contracts/ERC1404.sol
d47816a279f9ad0ebad434e8306bab82feaf134d316441b5165cb226f708cf00
    contracts/Migrations.sol
4fcb7cf2c06195f0ac2547ec44af3ace7cd437628b7f752356353da2883ab355
    contracts/Proxy.sol
```

```
e1190d300a5f6aedbe2d6c23060d869972b01758de1b2faaa0555c391b24d863
    contracts/WrappedTokenV1.sol
4eb6d858f4924faa1843667c3d9b9658ca66591cd5957b477220e7ef58818995
    contracts/capabilities/Blacklistable.sol
580bb595abf51da30cc3331f9841c8ad38941b7b55137a32510e7359d744ac62
    contracts/capabilities/Burnable.sol
78ef2a4dff2cc1b1cac043268ecc3f504a6e25e998cc5caa42d337471334a4ff
    contracts/capabilities/Mintable.sol
a5912ecf64cf66a90968adf480287cb41a790ced022216d5b4ce4a1b682d53d6
    contracts/capabilities/Pausable.sol
a1586fc27a8c3e23e341272e2989e64112654f64edf223ca5cc7509bee80a679
    contracts/capabilities/Proxiable.sol
9a4c541cc581c61d8189ed27c0cd778a83f6bd222449f0f81b2db5d892c493a1
    contracts/capabilities/Revocable.sol
c310be4cc37fa4a6a740f22f58118bb1d9eaffd710259a09424cb15304b9ab82
    contracts/capabilities/RevocableToAddress.sol
81ac290aef919d20ef67bff6546127f53c0684a35cb523b81dd6b3928a4019a3
    contracts/capabilities/Whitelistable.sol
884627ed63c739e97912c44f2f7122240015140452409781808124 3a3accede7
    contracts/roles/BlacklisterRole.sol
48cefe7ef2031444cf5c28b6de72fe8d98f2ed83f4f9309fc7bf2bd36b552acd
    contracts/roles/BurnerRole.sol
2190c1ef96f4f5bdfa2818c0addce3dac784f9adcadca732fe94b40c21cbb9a6
    contracts/roles/MinterRole.sol
03b4f83676693e1a9a93446a669108c3932853add95813b3e783465ee5ca0a9a
    contracts/roles/OwnerRole.sol
04e61578afa0d0defd7ecdb487e08968000beea2f13ac5c01f8778dfc46cd60f
    contracts/roles/PauserRole.sol
93df5a9e06ba478d4438f27b3e9377cc2f999b34e40731df1ca01534ecca988a
    contracts/roles/RevokerRole.sol
22a57ddefcb5037efabf06b4170c02b5636651df90e7594a859408e8561b6db2
    contracts/roles/WhitelisterRole.sol
```

OUT-OF-SCOPE:

Other smart contracts in the repository, external libraries and economical attacks. The versions for those external libraries can be found below:

Listing 3

```
"@chainlink/contracts": "0.1.9"
"@openzeppelin/contracts-upgradeable": "4.4.0"
```

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	2	2

LIKELIHOOD

IMPACT	🟨 (HAL-01)	🟨	🟥	🟥	🟪
	🟩	🟨	🟨	🟥	🟥
	🟩	🟩	🟨	🟨	🟥
	⬜	🟩	🟩 (HAL-02) (HAL-03)	🟨	🟨
	⬜ (HAL-04) (HAL-05)	⬜	🟩	🟩	🟨

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
NON-STANDARD INITIALIZATION	Medium	RISK ACCEPTED
UNEXPECTED FLASH-LOAN FLOW	Low	SOLVED - 12/07/2021
RE-ENTRANCY ALLOWED BY FLASH LOANS	Low	SOLVED - 12/07/2021
POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	ACKNOWLEDGED
UNUSED IMPORTS	Informational	SOLVED - 12/07/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) NON-STANDARD INITIALIZATION - MEDIUM

Description:

Proxy deployment is not using standard proxies such as UpgradeableProxy.sol or TransparentUpgradeableProxy.sol. Both Proxy.sol and WrappedTokenV1 are deployed independently and then linked with the initialize function on a separated transaction. This allows anyone the possibility to front-run the transaction and take control over the contract.

Code Location:

Listing 4: contracts/Proxy.sol

```
    constructor(address contractLogic) {
        // Verify a valid address was passed in
        require(contractLogic != address(0), "Contract Logic cannot be 0x0");

        // save the code address
        assembly {
            // solium-disable-line
            sstore(PROXIABLE_MEM_SLOT, contractLogic)
        }
    }
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

To prevent this, the contract initialize function should be called during the deployment/constructor of the proxy itself, binding the proxy with the implementation in a single transaction, thus preventing front-running. It is recommended to use an already tested Proxy version, such as the OpenZeppelin Proxies.

Remediation Plan:

RISK ACCEPTED: The Celsius team stated that if someone would front run they would just deploy again, so there’s no incentive to front run.

3.2 (HAL-02) UNEXPECTED FLASH-LOAN FLOW - LOW

Description:

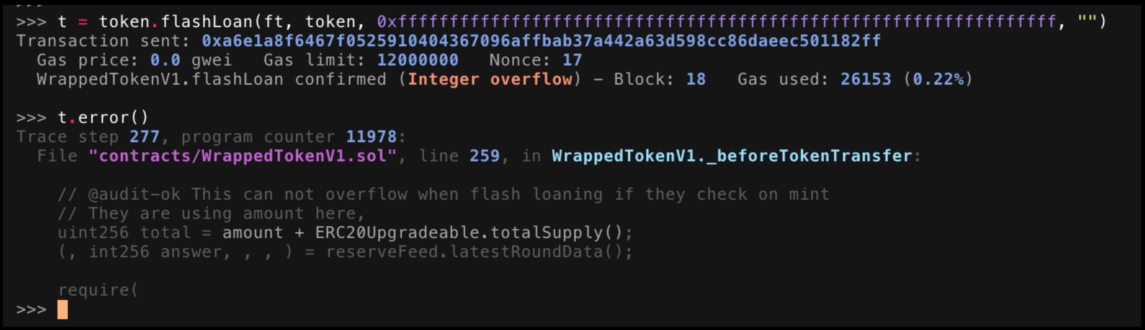
The flash loan is not allowed to pass the reserve amount; the check is performed on _beforeTokenTransfer. This means that the ERC20FlashMintUpgradeable.maxFlashLoan in mint check is unnecessary.

Furthermore, the flashLoan function does not use the Mintable._mint but the ERC20Upgradable._mint, thus making the Mintable._mint only used during WrappedTokenV1 initialization and on the exposed mint method.

This causes the flashLoan not to check for maxFlashLoan, relying uniquely on the reserve check.

Even if totalSupply were 0, the maximum accepted flash loan amount would be uint256.max. During the _beforeTokenTransfer the reserve check would always fail since > is used instead of >=. This confirms that for any case, not just the extreme, the reserve check would fail even before the maxFlashLoan would ever trigger.

Using a flash loan that surpasses the maxFlashLoan would still fail on the beforeTokenTransfer function with an overflow:



```
>>> t = token.flashLoan(ft, token, 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff, "")
Transaction sent: 0xa6e1a8f6467f0525910404367096affbab37a442a63d598cc86daeec501182ff
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 17
  WrappedTokenV1.flashLoan confirmed (Integer overflow) - Block: 18   Gas used: 26153 (0.22%)

>>> t.error()
Trace step 277, program counter 11978:
  File "contracts/WrappedTokenV1.sol", line 259, in WrappedTokenV1._beforeTokenTransfer:

    // @audit-ok This can not overflow when flash loaning if they check on mint
    // They are using amount here,
    uint256 total = amount + ERC20Upgradeable.totalSupply();
    (, int256 answer, , , ) = reserveFeed.latestRoundData();

    require(
>>> 
```

Code Location:

Listing 5: WrappedTokenV1.sol (Lines 252,257,258,259)

```
240     function _beforeTokenTransfer(
241         address from,
242         address to,
243         uint256 amount
244     ) internal override {
245         if (from != address(0)) {
246             return;
247         }
248
249         // silence warning about unused variable without the
                addition of bytecode.
250         to;
251
252         uint256 total = amount + ERC20Upgradeable.totalSupply();
253         (, int256 answer, , , ) = reserveFeed.latestRoundData();
254
255         require(
256             (answer > 0) &&
257                 ((uint256(answer) / 10**uint256(reserveFeed.
                        decimals())) *
258                     10**ERC20Upgradeable.decimals() >
259                     total),
260             "reserve must exceed the total supply"
261         );
262     }
```

Listing 6: capabilities/Mintable.sol (Lines 126)

```
117    function flashLoan(
118         IERC3156FlashBorrowerUpgradeable receiver,
119         address token,
120         uint256 amount,
121         bytes calldata data
122     ) public override returns (bool) {
123         require(isFlashMintEnabled, "flash mint is disabled");
124
125         uint256 fee = flashFee(token, amount);
126         _mint(address(receiver), amount);
```

Listing 7: capabilities/Mintable.sol (Lines 34)

```
28    function _mint(
29        address minter,
30        address to,
31        uint256 amount
32    ) internal returns (bool) {
33        require(
34            ERC20FlashMintUpgradeable.maxFlashLoan(address(this))
                 > amount,
35            "mint exceeds max allowed"
36        );
37
38        ERC20Upgradeable._mint(to, amount);
39        emit Mint(minter, to, amount);
40        return true;
41    }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

It is recommended to modify the flashLoan function to use Mintable._mint and optionally Burnable._burn. Another option would be to remove the Mintable._mint flash loan check and rely on the _beforeTokenTransfer check for normal mint and flashLoan, always keeping in mind that the security of it relies on the overflow checks introduced on solidity 0.8.0 on uint256 total = amount + ERC20Upgradeable.totalSupply();. If the latter is chosen, downgrading the pragma version would introduce a critical flaw on the system.

Remediation Plan:

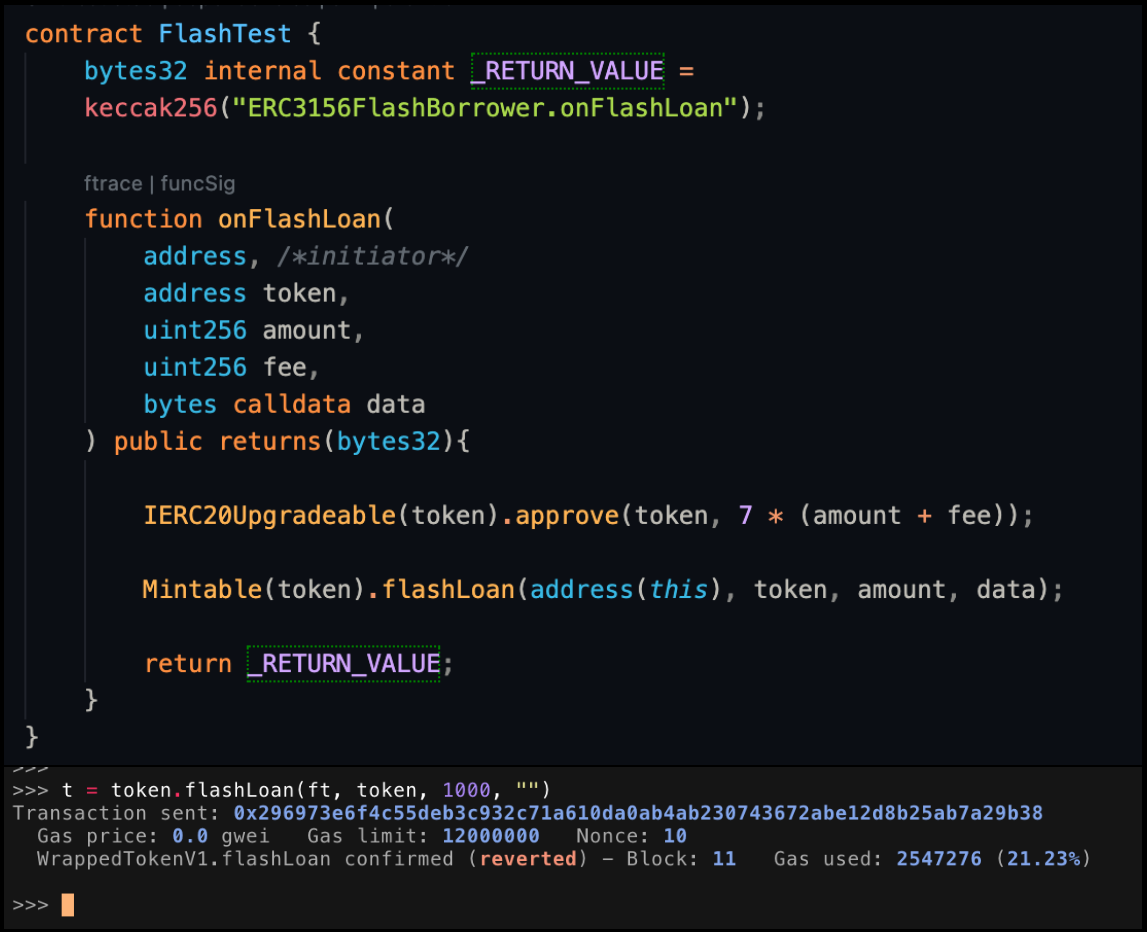
SOLVED: The mint function was declared on the WrappedToken contract to directly use Mintable.mint. Now, mint will check the reserve amount and flashLoan will only check the maxFlashLoan on _beforeTokenTransfer. Furthermore, the code now contemplates the possibility of reserveFeed decimals being different from the token decimals.

3.3 (HAL-03) RE-ENTRANCY ALLOWED BY FLASH LOANS - LOW

Description:

The flashLoan function from the capability/Mintable.sol can be recalled from the receiver contract causing the transaction to revert.

Proof of Concept:



```
contract FlashTest {
    bytes32 internal constant _RETURN_VALUE =
    keccak256("ERC3156FlashBorrower.onFlashLoan");

    ftrace | funcSig
    function onFlashLoan(
        address, /*initiator*/
        address token,
        uint256 amount,
        uint256 fee,
        bytes calldata data
    ) public returns(bytes32){

        IERC20Upgradeable(token).approve(token, 7 * (amount + fee));

        Mintable(token).flashLoan(address(this), token, amount, data);

        return _RETURN_VALUE;
    }
}
```

```
>>> t = token.flashLoan(ft, token, 1000, "")
Transaction sent: 0x296973e6f4c55deb3c932c71a610da0ab4ab230743672abe12d8b25ab7a29b38
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 10
  WrappedTokenV1.flashLoan confirmed (reverted) - Block: 11   Gas used: 2547276 (21.23%)

>>>
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

It is recommended to add a re-entrancy lock on the flashLoan function to prevent unexpected behaviors or possible denial of service.

Remediation Plan:

SOLVED: The nonReentrant lock was added into the flashLoan function, preventing re-entrancy.

3.4 (HAL-04) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In public functions, array arguments are immediately copied to memory while external functions can read directly from calldata. Reading calldata is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments to be located in memory when the compiler generates the code for an internal function.

Code Location:

Roles:

- BlacklisterRole.sol
  - addBlacklister
  - removeBlacklister
- BurnerRole.sol
  - addBurner
  - removeBurner
- MinterRole.sol
  - addMinter
  - removeMinter
- OwnerRole.sol
  - addOwner
  - removeOwner
- PauserRole.sol
  - addPauser
  - removePauser

- RevokerRole.sol
  - addRevoker
  - removeRevoker
- WhitelisterRole.sol
  - addWhitelister
  - removeWhitelister

Capabilities:

- Blacklistable.sol
  - checkBlacklistAllowed
  - setBlacklistEnabled
  - removeFromBlacklist
- Burnable.sol
  - burn
- Mintable.sol
  - setFlashMintFee
  - setFlashMintEnabled
  - setFlashMintFeeReceiver
  - flashLoan
- Pausable.sol
  - pause
  - unpause
- Revocable.sol
  - revoke
- RevocableToAddress.sol
  - revokeToAddress
- Whitelistable.sol
  - setWhitelistEnabled
  - addToWhitelist
  - removeFromWhitelist
  - updateOutboundWhitelistEnabled

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider declaring external variables instead of public variables. A best practice is to use external if expecting a function to only be called externally and public if called internally. Public functions are always accessible, but external functions are only available to external callers.

Remediation Plan:

ACKNOWLEDGED: The Celsius team acknowledged the issue.

3.5 (HAL-05) UNUSED IMPORTS - INFORMATIONAL

Description:

The OwnerRole contract does import the standard OpenZeppelin AccessControlUpgradeable library. However, the code does not make use of its functionality and only the internal RoleData struct is used.

Code Location:

Listing 8: OwnerRole.sol

```solidity
contract OwnerRole is AccessControlUpgradeable {
    event OwnerAdded(address indexed addedOwner, address indexed addedBy);
    event OwnerRemoved(address indexed removedOwner, address indexed removedBy);
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to keep the code as small and as functional as possible. This will prevent unnecessary gas costs, easier development and less inheritance complexity. In this case, copying the RoleData to the OwnerRole and removing the AccessControlUpgradeable dependency would solve the issue.

Remediation Plan:

SOLVED: The client did refactor the code to only include the RoleData struct, which was internally renamed to Role.

MANUAL TESTING

4.1 Capabilities

Blacklistable:

The checkBlacklistAllowed function state is valid:

Sender	Receiver	isBlacklistEnabled	Return
False	False	True	True
False	True	True	False
True	False	True	False
True	True	True	False
False	False	False	True
False	True	False	True
True	False	False	True
True	True	False	True

Mintable:

- The following initial values could be omitted, by default they are 0 and false(0).

Listing 9

```
uint256 public flashMintFee = 0;
bool public isFlashMintEnabled = false;
```

- By default, fees go to the owner of WrappedTokenV1
- By default, flash minting is enabled is provided on the token constructor.
- The maxFlashLoan will always be the (max(uint256)-1)- totalSupply; this is the remaining amount of tokens to overflow a full uint256.

- This means that _mint will allow minting a max amount of maxFlashLoan .

- Mint call trace:

Listing 10

```
WrappedTokenV1.mint
 Mintable.mint
    OwnerRole._has
 Mintable._mint
     ERC20FlashMintUpgradeable.maxFlashLoan
        ERC20Upgradeable.totalSupply
     ERC20Upgradeable._mint
         WrappedTokenV1._beforeTokenTransfer
             ERC20Upgradeable.totalSupply
             MockV3Aggregator.latestRoundData  [STATICCALL]
        _totalSupply += amount;
        _balances[account] += amount;

```

- The total supply is incremented after the previous call trace. That means that _beforeTokenTransfer has to take into consideration the amount, which it does, to validate the reserve.

- Flash loan call trace:

Listing 11

```
WrappedTokenV1.flashLoan
 Mintable.flashLoan
    Mintable.flashFee
    ERC20Upgradeable._mint
       WrappedTokenV1._beforeTokenTransfer
           ERC20Upgradeable.totalSupply
           MockV3Aggregator.latestRoundData  [STATICCALL]
        (_totalSupply += amount;)
        (_balances[account] += amount;)
    ERC20Upgradeable.decimals
 MockV3Aggregator.decimals  [STATICCALL]
 FlashTest.onFlashLoan  [CALL]
    WrappedTokenV1.approve  [CALL]
```

```
       ERC20Upgradeable.approve
           ContextUpgradeable._msgSender
           ERC20Upgradeable._approve
ERC20Upgradeable.allowance
ERC20Upgradeable._transfer
   WrappedTokenV1._beforeTokenTransfer
   AccessControlUpgradeable.grantRole
ERC20Upgradeable._approve
ERC20Upgradeable._burn  2414:2580
    WrappedTokenV1._beforeTokenTransfer
    AccessControlUpgradeable.grantRole
```

- onFlashLoan can call flashLoan again, causing DOS.

Revocable:

Revoker has the privilege to transfer any amount of tokens from any user on its behalf.

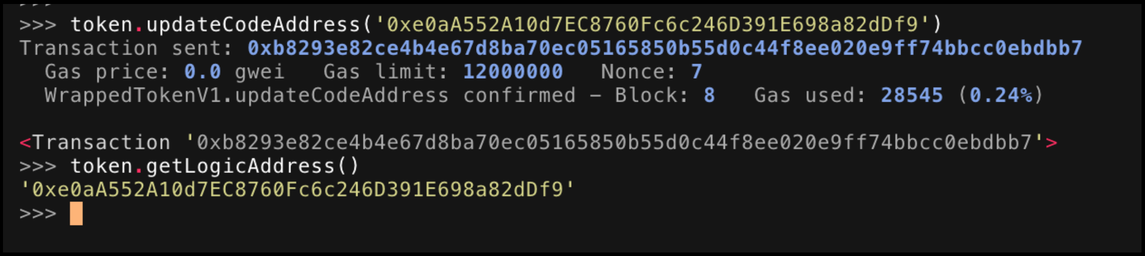
RevocableToAddress:

Does the same as Revocable but allows specifying the to parameter.

4.2 WrappedTokenV1

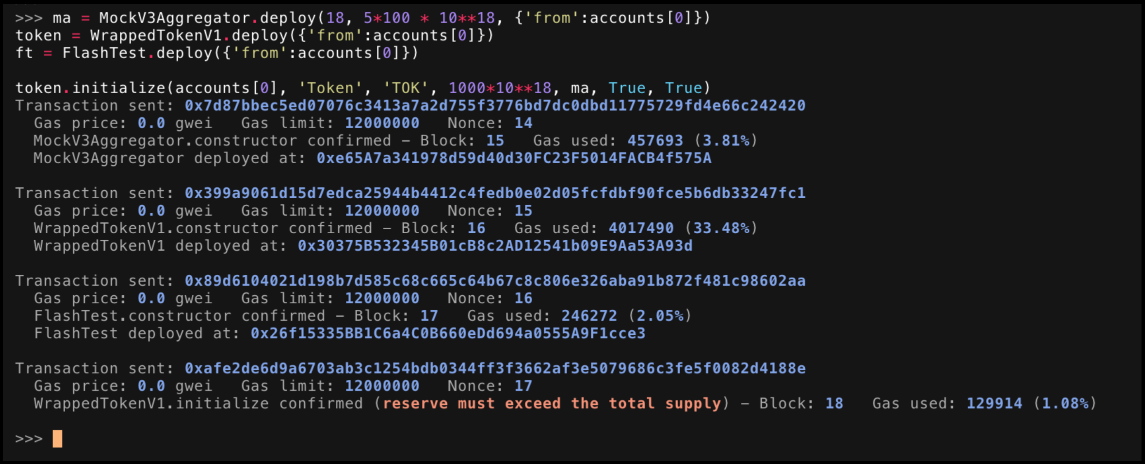
- _beforeTokenTransfer is also called when minting and burning. This could lead issues while flash-loaning.
- owner can transfer to anyone, even if not whitelisted.
- _setBlacklistEnabled is not called on the initializer. This means that checkBlacklistAllowed will always return True.

Updating Proxiable address will update the proxy storage value.



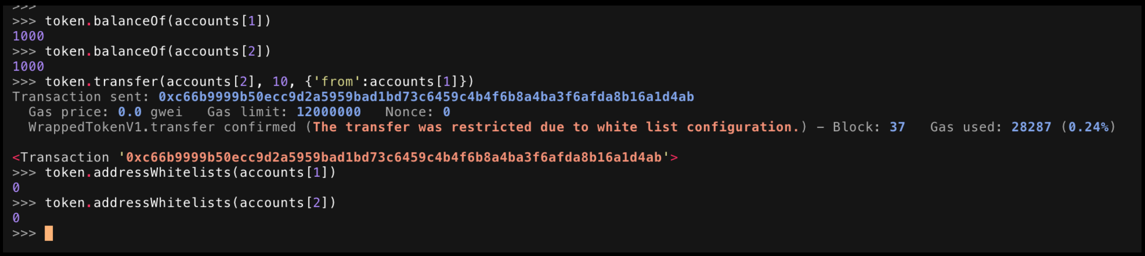
```
>>> token.updateCodeAddress('0xe0aA552A10d7EC8760Fc6c246D391E698a82dDf9')
Transaction sent: 0xb8293e82ce4b4e67d8ba70ec05165850b55d0c44f8ee020e9ff74bbcc0ebdbb7
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  WrappedTokenV1.updateCodeAddress confirmed - Block: 8   Gas used: 28545 (0.24%)

<Transaction '0xb8293e82ce4b4e67d8ba70ec05165850b55d0c44f8ee020e9ff74bbcc0ebdbb7'>
>>> token.getLogicAddress()
'0xe0aA552A10d7EC8760Fc6c246D391E698a82dDf9'
>>>
```

Initializing with less than the reserve results in an error during the _mint.



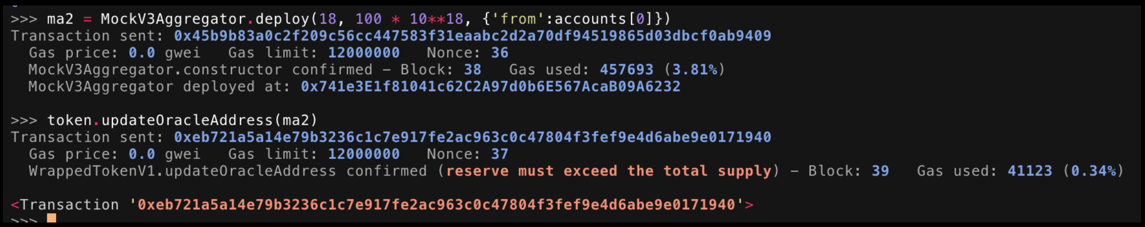
```
>>> ma = MockV3Aggregator.deploy(18, 5*100 * 10**18, {'from':accounts[0]})
token = WrappedTokenV1.deploy({'from':accounts[0]})
ft = FlashTest.deploy({'from':accounts[0]})

token.initialize(accounts[0], 'Token', 'TOK', 1000*10**18, ma, True, True)
Transaction sent: 0x7d87bbec5ed07076c3413a7a2d755f3776bd7dc0dbd11775729fd4e66c242420
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 14
  MockV3Aggregator.constructor confirmed - Block: 15   Gas used: 457693 (3.81%)
  MockV3Aggregator deployed at: 0xe65A7a341978d59d40d30FC23F5014FACB4f575A

Transaction sent: 0x399a9061d15d7edca25944b4412c4fedb0e02d05fcfdbf90fce5b6db33247fc1
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 15
  WrappedTokenV1.constructor confirmed - Block: 16   Gas used: 4017490 (33.48%)
  WrappedTokenV1 deployed at: 0x30375B532345B01cB8c2AD12541b09E9Aa53A93d

Transaction sent: 0x89d6104021d198b7d585c68c665c64b67c8c806e326aba91b872f481c98602aa
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 16
  FlashTest.constructor confirmed - Block: 17   Gas used: 246272 (2.05%)
  FlashTest deployed at: 0x26f15335BB1C6a4C0B660eDd694a0555A9F1cce3

Transaction sent: 0xafe2de6d9a6703ab3c1254bdb0344ff3f3662af3e5079686c3fe5f0082d4188e
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 17
  WrappedTokenV1.initialize confirmed (reserve must exceed the total supply) - Block: 18   Gas used: 129914 (1.08%)

>>>
```

The transfer call will fail if either the receiver or the sender does not have a whitelist. However, if the owner is calling the transfer function, no restrictions exist:



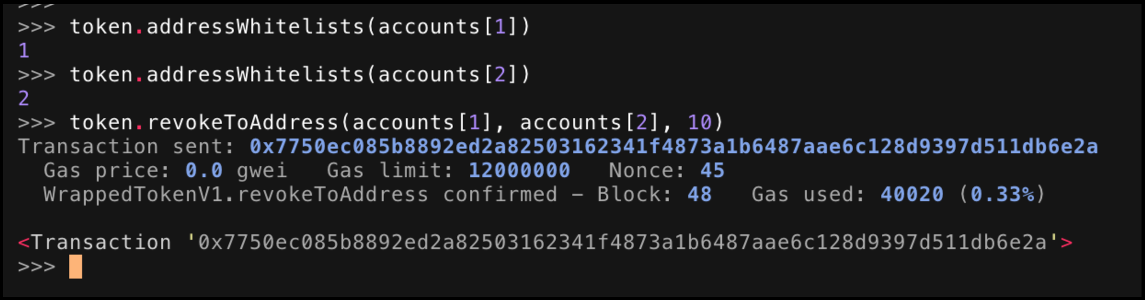
```
>>> token.balanceOf(accounts[1])
1000
>>> token.balanceOf(accounts[2])
1000
>>> token.transfer(accounts[2], 10, {'from':accounts[1]})
Transaction sent: 0xc66b9999b50ecc9d2a5959bad1bd73c6459c4b4f6b8a4ba3f6afda8b16a1d4ab
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  WrappedTokenV1.transfer confirmed (The transfer was restricted due to white list configuration.) - Block: 37   Gas used: 28287 (0.24%)

<Transaction '0xc66b9999b50ecc9d2a5959bad1bd73c6459c4b4f6b8a4ba3f6afda8b16a1d4ab'>
>>> token.addressWhitelists(accounts[1])
0
>>> token.addressWhitelists(accounts[2])
0
>>>
```

Updating the oracle address with a latestRoundData smaller than the totalSupply will result in a revert:



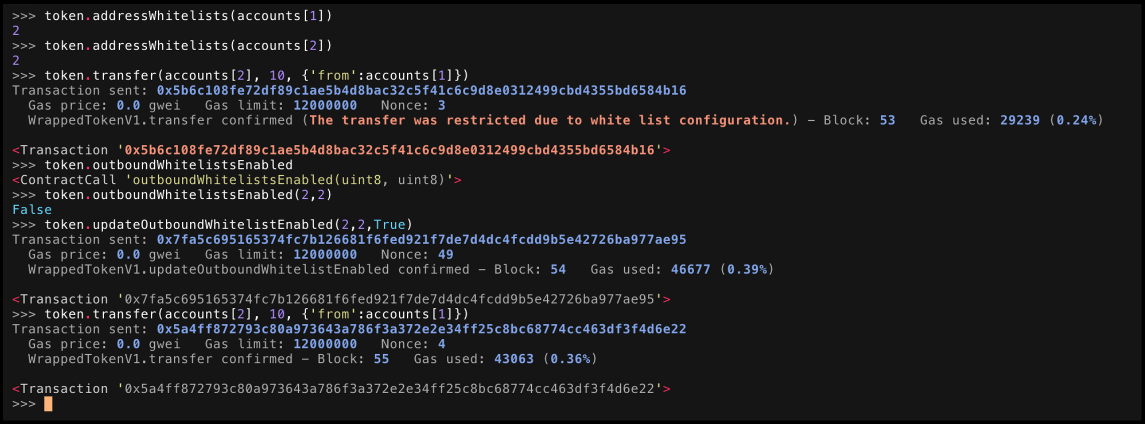
```
>>> ma2 = MockV3Aggregator.deploy(18, 100 * 10**18, {'from':accounts[0]})
Transaction sent: 0x45b9b83a0c2f209c56cc447583f31eaabc2d2a70df94519865d03dbcf0ab9409
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 36
  MockV3Aggregator.constructor confirmed - Block: 38   Gas used: 457693 (3.81%)
  MockV3Aggregator deployed at: 0x741e3E1f81041c62C2A97d0b6E567AcaB09A6232

>>> token.updateOracleAddress(ma2)
Transaction sent: 0xeb721a5a14e79b3236c1c7e917fe2ac963c0c47804f3fef9e4d6abe9e0171940
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 37
  WrappedTokenV1.updateOracleAddress confirmed (reserve must exceed the total supply) - Block: 39   Gas used: 41123 (0.34%)

<Transaction '0xeb721a5a14e79b3236c1c7e917fe2ac963c0c47804f3fef9e4d6abe9e0171940'>
>>>
```

Calling revoke or revokeToAddress (this action can only be performed by the Revocable addresses) does allow skipping the restriction checks:



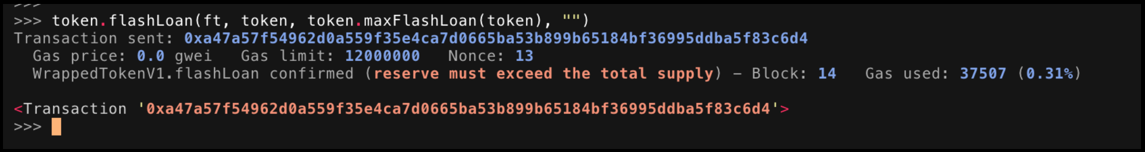
```
>>> token.addressWhitelists(accounts[1])
1
>>> token.addressWhitelists(accounts[2])
2
>>> token.revokeToAddress(accounts[1], accounts[2], 10)
Transaction sent: 0x7750ec085b8892ed2a82503162341f4873a1b6487aae6c128d9397d511db6e2a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 45
  WrappedTokenV1.revokeToAddress confirmed - Block: 48   Gas used: 40020 (0.33%)

<Transaction '0x7750ec085b8892ed2a82503162341f4873a1b6487aae6c128d9397d511db6e2a'>
>>> 
```

Transfers between the same whitelists are not allowed by default if not specified under outboundWhitelistsEnabled



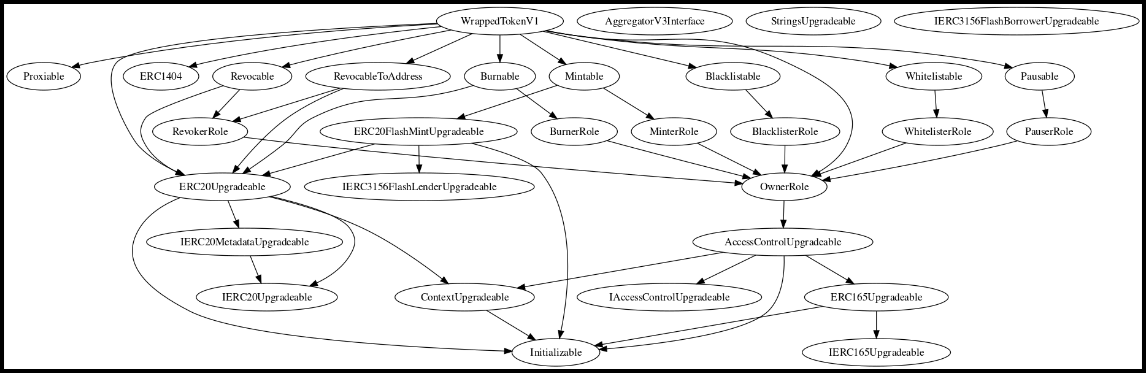
```
>>> token.addressWhitelists(accounts[1])
2
>>> token.addressWhitelists(accounts[2])
2
>>> token.transfer(accounts[2], 10, {'from':accounts[1]})
Transaction sent: 0x5b6c108fe72df89c1ae5b4d8bac32c5f41c6c9d8e0312499cbd4355bd6584b16
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  WrappedTokenV1.transfer confirmed (The transfer was restricted due to white list configuration.) - Block: 53   Gas used: 29239 (0.24%)

<Transaction '0x5b6c108fe72df89c1ae5b4d8bac32c5f41c6c9d8e0312499cbd4355bd6584b16'>
>>> token.outboundWhitelistsEnabled
<ContractCall 'outboundWhitelistsEnabled(uint8, uint8)'>
>>> token.outboundWhitelistsEnabled(2,2)
False
>>> token.updateOutboundWhitelistEnabled(2,2,True)
Transaction sent: 0x7fa5c695165374fc7b126681f6fed921f7de7d4dc4fcdd9b5e42726ba977ae95
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 49
  WrappedTokenV1.updateOutboundWhitelistEnabled confirmed - Block: 54   Gas used: 46677 (0.39%)

<Transaction '0x7fa5c695165374fc7b126681f6fed921f7de7d4dc4fcdd9b5e42726ba977ae95'>
>>> token.transfer(accounts[2], 10, {'from':accounts[1]})
Transaction sent: 0x5a4ff872793c80a973643a786f3a372e2e34ff25c8bc68774cc463df3f4d6e22
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  WrappedTokenV1.transfer confirmed - Block: 55   Gas used: 43063 (0.36%)

<Transaction '0x5a4ff872793c80a973643a786f3a372e2e34ff25c8bc68774cc463df3f4d6e22'>
>>> 
```

The flash loan is not allowed to pass the reserve amount; the check is performed on _beforeTokenTransfer. This means that the ERC20FlashMintUpgradeable.maxFlashLoan in mint check is unnecessary.



```
>>> token.flashLoan(ft, token, token.maxFlashLoan(token), "")
Transaction sent: 0xa47a57f54962d0a559f35e4ca7d0665ba53b899b65184bf36995ddba5f83c6d4
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 13
  WrappedTokenV1.flashLoan confirmed (reserve must exceed the total supply) - Block: 14   Gas used: 37507 (0.31%)

<Transaction '0xa47a57f54962d0a559f35e4ca7d0665ba53b899b65184bf36995ddba5f83c6d4'>
>>> 
```

INHERITANCE INITIALIZATION AND CALL CHECKS

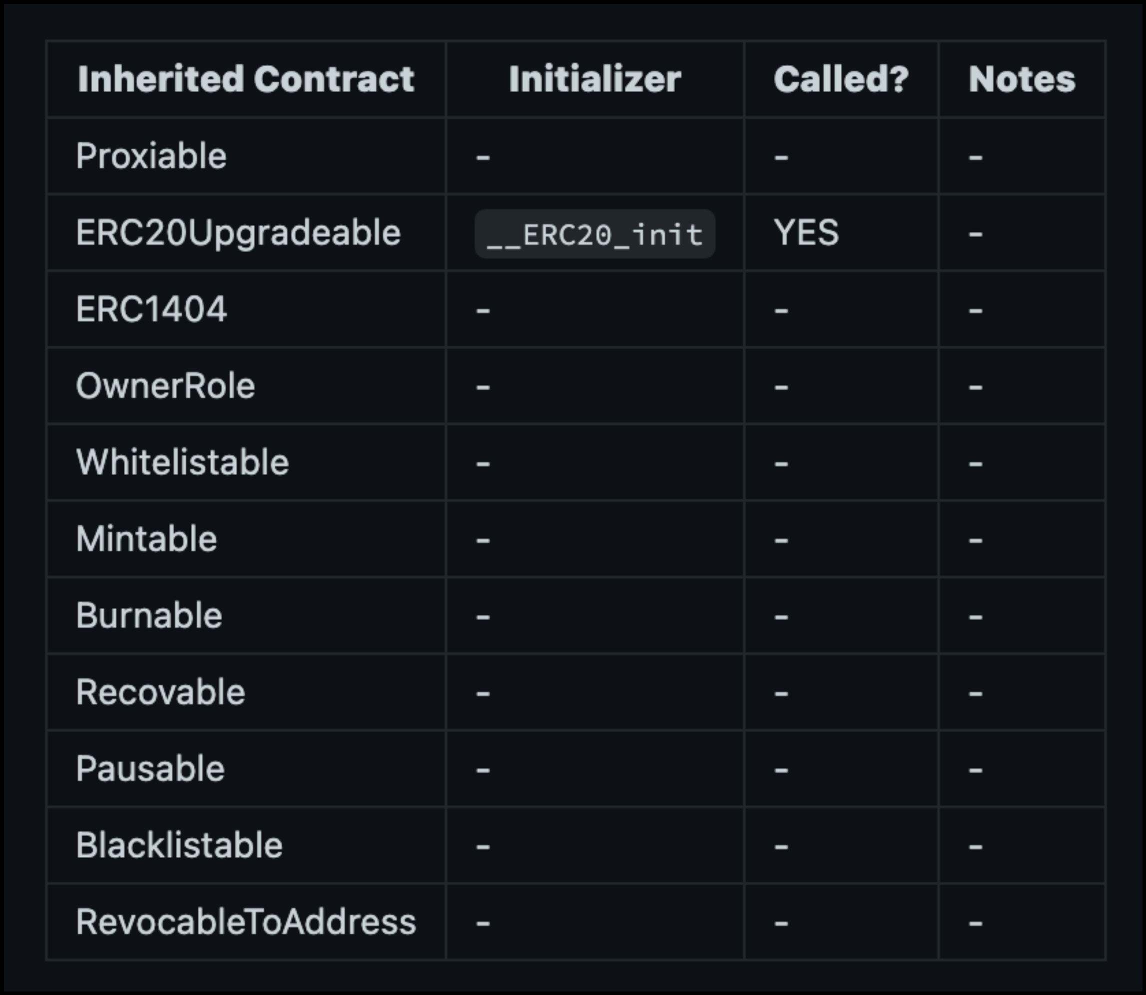


5.1 Initializers

The contract is upgradable, so all inherited contracts should be correctly initialized under the initialize function or sub-initializers:

First level inherited contracts:

- Proxiable
- ERC20Upgradeable
- ERC1404
- OwnerRole
- Whitelistable
- Mintable
- Burnable
- Revocable
- Pausable
- Blacklistable
- RevocableToAddress



Inherited Contract	Initializer	Called?	Notes
Proxiable	-	-	-
ERC20Upgradeable	`__ERC20_init`	YES	-
ERC1404	-	-	-
OwnerRole	-	-	-
Whitelistable	-	-	-
Mintable	-	-	-
Burnable	-	-	-
Recovable	-	-	-
Pausable	-	-	-
Blacklistable	-	-	-
RevocableToAddress	-	-	-

NOTE: Contracts marked with ("*") are OpenZeppelin contracts and sub-inheritance was not checked.

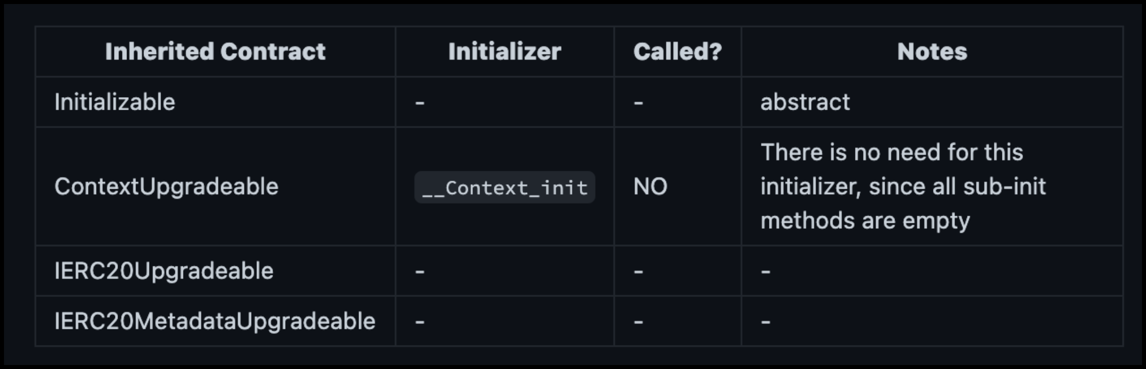
5.2 Sub-inheritance

Proxiable:

It does not have any constructor or variable state change: no need for initialization.

ERC20Upgradeable:

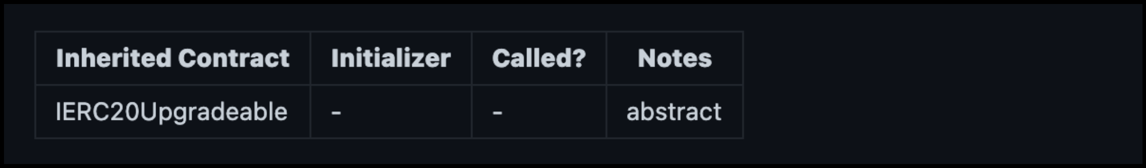
The initializer __ERC20_init or __ERC20_init_unchained should be called on the parent contract.



Inherited Contract	Initializer	Called?	Notes
Initializable	-	-	abstract
ContextUpgradeable	`__Context_init`	NO	There is no need for this initializer, since all sub-init methods are empty
IERC20Upgradeable	-	-	-
IERC20MetadataUpgradeable	-	-	-

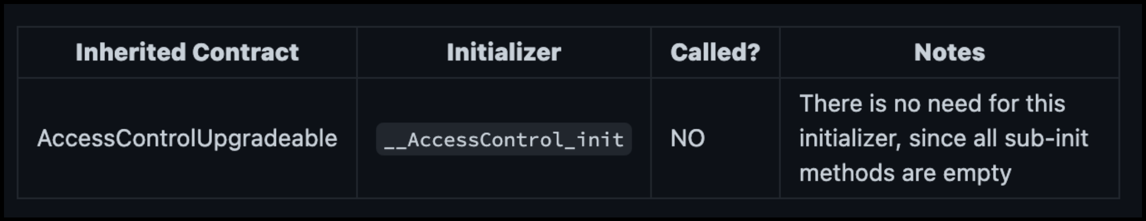
ERC1404:

It does not have any constructor or variable state change: no need for initialization.



Inherited Contract	Initializer	Called?	Notes
IERC20Upgradeable	-	-	abstract

OwnerRole:



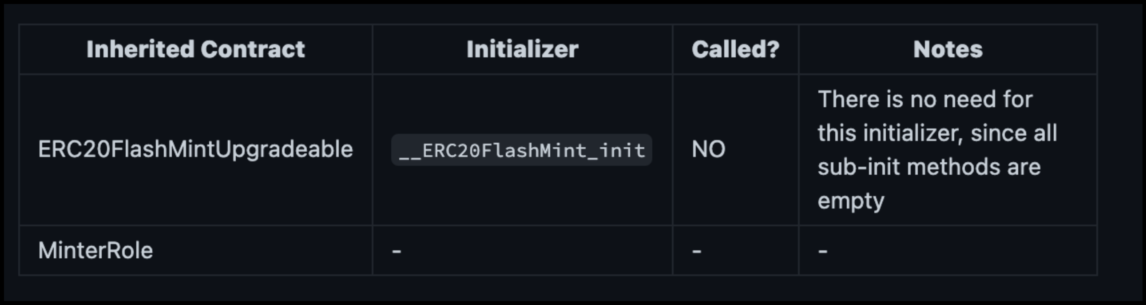
Inherited Contract	Initializer	Called?	Notes
AccessControlUpgradeable	`__AccessControl_init`	NO	There is no need for this initializer, since all sub-init methods are empty

Whitelistable:

It does not have any constructor or variable state change: no need for initialization.

Mintable:

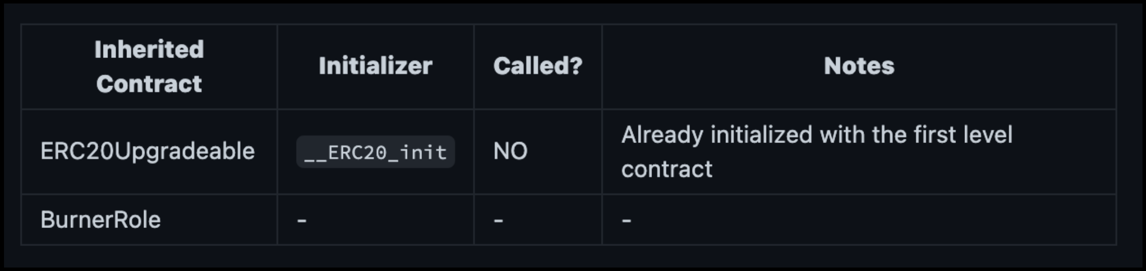
It does not have any constructor or variable state change: no need for initialization.



Inherited Contract	Initializer	Called?	Notes
ERC20FlashMintUpgradeable	`__ERC20FlashMint_init`	NO	There is no need for this initializer, since all sub-init methods are empty
MinterRole	-	-	-

Burnable:

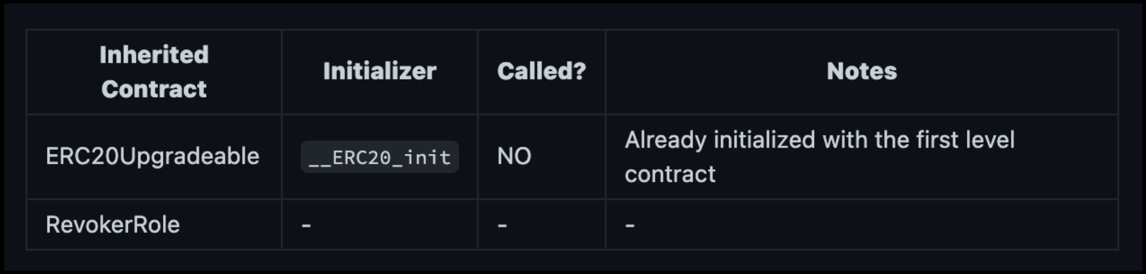
It does not have any constructor or variable state change: no need for initialization.



Inherited Contract	Initializer	Called?	Notes
ERC20Upgradeable	`__ERC20_init`	NO	Already initialized with the first level contract
BurnerRole	-	-	-

Revocable:

It does not have any constructor or variable state change: no need for initialization.



Inherited Contract	Initializer	Called?	Notes
ERC20Upgradeable	`__ERC20_init`	NO	Already initialized with the first level contract
RevokerRole	-	-	-

Pausable:

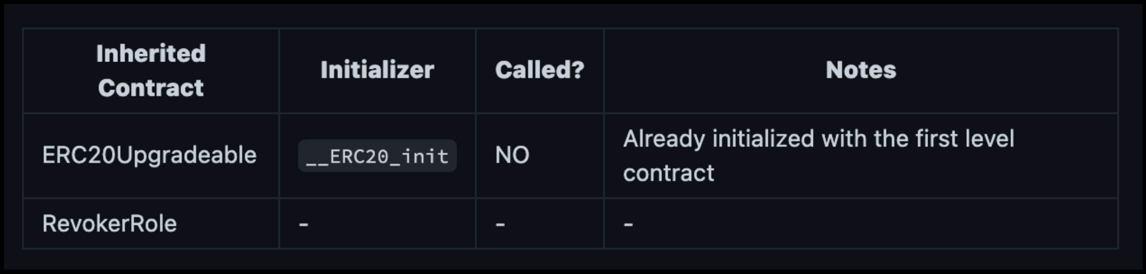
It does not have any constructor or variable state change: no need for initialization.

Blacklistable:

It does not have any constructor or variable state change: no need for initialization.

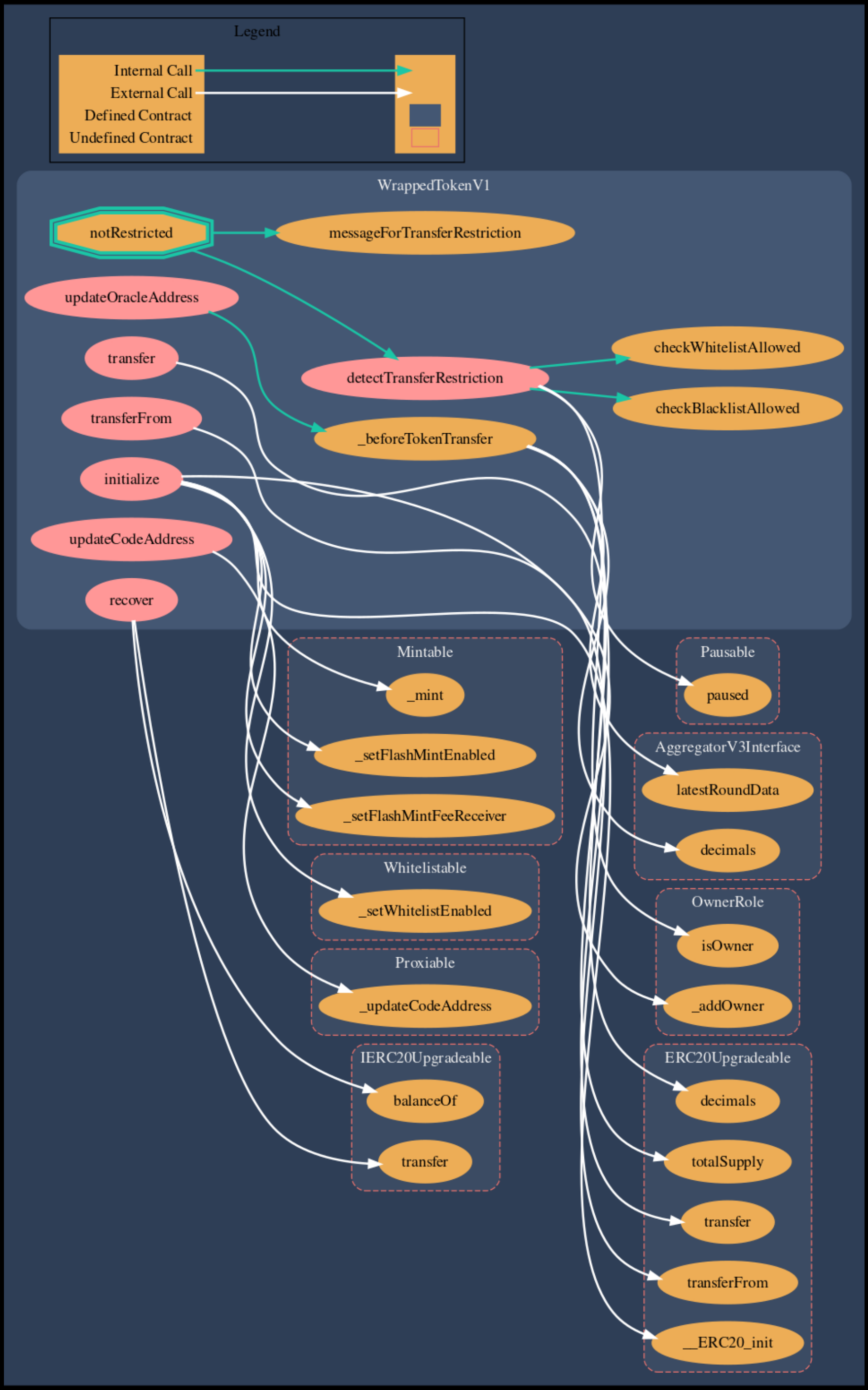
RevocableToAddress:

It does not have any constructor or variable state change: no need for initialization.



Inherited Contract	Initializer	Called?	Notes
ERC20Upgradeable	`__ERC20_init`	NO	Already initialized with the first level contract
RevokerRole	-	-	-

5.3 Call Graph



The call graph looks perfectly valid. As stated on (HAL-04) POSSIBLE MISUSE OF PUBLIC FUNCTIONS some exposed functions are not internally called, thus the recommendation.

AUTOMATED TESTING

6.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither results:

Slither did not find anything worth mentioning.

The only reported high severity issue is the shadowing on the __gap variable of the inherited contract. This is already known.

THANK YOU FOR CHOOSING

HALBORN