Ondo Finance
Findings & Analysis Report

2023-10-12

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit outlined in this document, C4 conducted an analysis of the Ondo Finance smart contract system written in Solidity. The audit took place between September 1 — September 7, 2023.

Wardens

71 Wardens contributed reports to the Ondo Finance:

1. adriro
2. gkrastenov
3. lsaudit
4. 0xpiken
5. nirlin
6. ast3ros
7. ladboy233
8. pontifex
9. 0xAsen

10. 0xStalin
11. Inspecktor
12. Arz
13. BenRai
14. merlin
15. Delvir0
16. Udsen
17. kaveyjoe
18. kutugu
19. bin2chen
20. pep7siup
21. 0xDING99YA
22. SpicyMeatball
23. bowtiedvirus
24. catellatech
25. hunter_w3b
26. JayShreeRAM
27. DedOhWale
28. c3phas
29. seerether
30. K42
31. castle_chain
32. naman1778
33. 0xmystery
34. matrix_0wl
35. Breeje
36. peanuts
37. sandy
38. hals

39. m4ttm
40. mahdikarimi
41. Bube
42. Krace
43. 0xE1D
44. MohammedRizwan
45. wahedtalash77
46. petrichor
47. zabihullahazadzoi
48. SY_S
49. SAQ
50. SAAJ
51. 0xhex
52. 0xta
53. jeffy
54. 0x11singh99
55. albahaca
56. Eurovickk
57. ybansal2403
58. Raihan
59. klau5
60. dev0cloo
61. Aymen0909
62. Lalanda
63. SovaSlava
64. ohm (0x*kmr* and 0x4ka5h)
65. 0xpanicError
66. codegpt
67. 0x6980

68. Stormreckson
69. pipidu83
70. 0xanmol

This audit was judged by kirk-baird.

Final report assembled by thebrittfactor.

Summary

The C4 analysis yielded an aggregated total of 4 unique vulnerabilities, all with a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 34 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 21 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Ondo Finance repository, and is composed of 6 smart contracts written in the Solidity programming language and includes 962 lines of Solidity code.

In addition to the known issues identified by the project team, a Code4rena bot race was conducted at the start of the audit. The winning bot, **henry** from warden hihen, generated the Automated Findings report and all findings therein were classified as out of scope.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling

- Escalation of privileges
- Arithmetic
- Gas use

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website, specifically our section on Severity Categorization.

Medium Risk Findings (4)

[M-01] Chain support cannot be removed or cleared in bridge contracts

Submitted by adriro, also found by gkrastenov and lsaudit

Due to how addresses are handled and stored in the configuration settings, it is not possible to remove chain support in both source and destination bridge contracts.

The Axelar bridge service uses strings to represent addresses: messages sent to another chain need to specify its destination contract as a string. The protocol decided to follow the same representation and contract store addresses as strings as part of their configuration.

Chain support in the bridge contracts is then represented by associating the chain name with the address of the contract, stored as a string. This can be seen in the implementation of `setDestinationChainContractAddress()` for the SourceBridge contract, which stores the string, and the implementation of `addChainSupport()` in the DestinationBridge contract, which stores the hash of the string:

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/bridge/SourceBridge.sol#L121-L129

```
121:   function setDestinationChainContractAddress(
122:     string memory destinationChain,
123:     address contractAddress
124:   ) external onlyOwner {
125:     destChainToContractAddr[destinationChain] = AddressToSt
126:       contractAddress
127:     );
128:     emit DestinationChainContractAddressSet(destinationChai
129:   }
```

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/bridge/DestinationBridge.sol#L234-L240

```
234:   function addChainSupport(
235:     string calldata srcChain,
236:     string calldata srcContractAddress
237:   ) external onlyOwner {
238:     chainToApprovedSender[srcChain] = keccak256(abi.encode(
239:     emit ChainIdSupported(srcChain, srcContractAddress);
240:   }
```

This also means that checks need to be done based on the stored representation. SourceBridge checks the length of the string, while DestinationBridge checks for a `bytes32(0)` value:

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/bridge/SourceBridge.sol#L68-L71

```
68:     if (bytes(destContract).length == 0) {
69:       revert DestinationNotSupported();
70:     }
71:
```

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/bridge/DestinationBridge.sol#L96-L98

```
96:     if (chainToApprovedSender[srcChain] == bytes32(0)) {
97:       revert ChainNotSupported();
98:     }
```

Note that this implies that there is no way to clear these settings and remove chain support. In the case of SourceBridge, any address sent to `setDestinationChainContractAddress()` will be converted to their string representation, which will always have a length greater than zero. For the DestinationBridge, `addChainSupport()` will hash the address parameter and it will be impossible for that hash value to be zero (since it will imply knowing the preimage of zero).

Proof of Concept

1. Admin configs the destination address in SourceBridge by calling `setDestinationChainContractAddress("optimism", destinationAddress)`.
2. Admin decides to remove support for Optimism.
3. Admin calls `setDestinationChainContractAddress("optimism", address(0))`, but this will actually store the string for the zero address `0x0000....000`.
4. The check `bytes(destContract).length == 0` will fail and messages will still be routed.

Recommended Mitigation Steps

Provide functions in both contracts to allow the owner to clear the settings by resetting their configuration to the default value.

```
function removeDestinationChainContractAddress(
  string memory destinationChain
) external onlyOwner {
  delete destChainToContractAddr[destinationChain];
  emit DestinationChainContractAddressRemoved(destinationChain);
}
```

```
function removeChainSupport(
  string calldata srcChain
) external onlyOwner {
  delete chainToApprovedSender[srcChain];
  emit ChainIdRemoved(srcChain);
}
```

tom2o17 (Ondo) commented:

> @kirk-baird - Can I not set acceptable `srcAddr` to `address(0)` and then, given `address(0)` cannot have a contract deployed to it, this value should not be within the possible ranges of `srcAddr` within the `_execute` function?

> Granted not super ideal given it will hit a different error message, but functionally, the result is the same.

[kirk-baird (judge) commented](#):

> @tom2o17 - My concern here is that there's no way to stop calling `burnAndCallAxelar()`, which would burn tokens on the source chain. These would not be minted on the destination chain and are essentially lost.

> E.g. If you call `setDestinationChainContractAddress(dstChain, address(0))`, then [Address.toString(address(0))](#) will encode that to 42 hex characters.

> So [this check](#) in `burnAndCallAxelar()` will always pass, since `bytes(destContract).length` is 42 bytes.

[tom2o17 (Ondo) commented](#):

> @kirk-baird - Ah I see. I am inclined to accept this issue. Apologies, I thought this was referencing the `dstBridge` not the `srcBridge`. As mitigation we would be able to pause the contract, and prevent this functionality, but agree with the auditor that this is not ideal.

[tom2o17 (Ondo) acknowledged](#)

[M-02] All bridged funds will be lost for the users using the account abstraction wallet

Submitted by [nirlin](#), also found by [pontifex](#), [ladboy233](#), [ast3ros](#), and [0xpiken](#)

Lines of code

[https://github.com/code-423n4/2023-09-ondo/blob/47d34d6d4a5303af5f46e907ac2292e6a7745f6c/contracts/bridge/Sou](#)

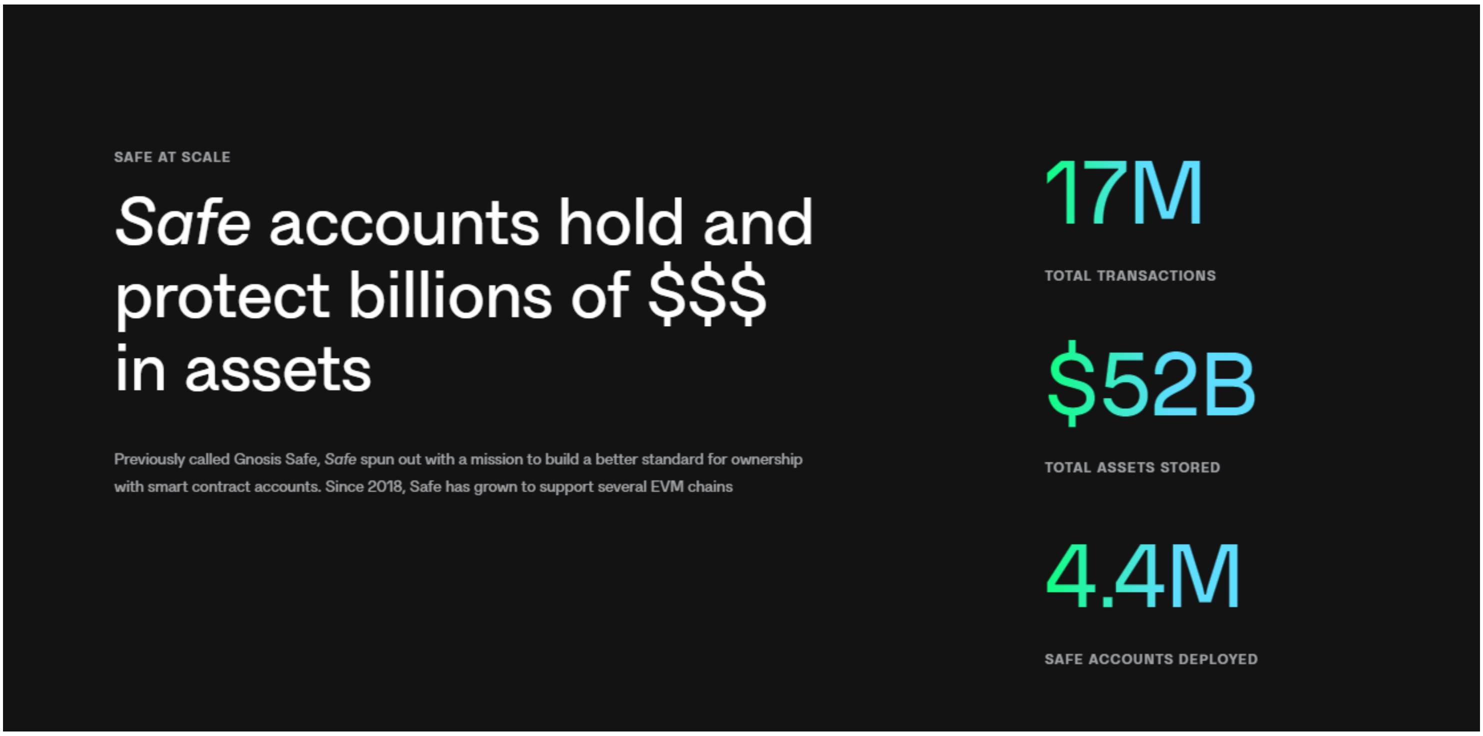
rceBridge.sol#L61-L82
https://github.com/code-423n4/2023-09-ondo/blob/47d34d6d4a5303af5f46e907ac2292e6a7745f6c/contracts/bridge/DestinationBridge.sol#L85-L114

Impact

Users with account abstraction wallets have a different address across different chains for same account, so if someone using an account abstraction wallet bridge the asset, assets will be minted to wrong address and lost permanently.

Proof of Concept

Account abstraction wallets have been on the rise for quite a time now and have a lot of users. See the below image for the figures by safe wallet (one of the account abstraction wallets):



SAFE AT SCALE

Safe accounts hold and protect billions of $$$ in assets

Previously called Gnosis Safe, *Safe* spun out with a mission to build a better standard for ownership with smart contract accounts. Since 2018, Safe has grown to support several EVM chains

17M

TOTAL TRANSACTIONS

$52B

TOTAL ASSETS STORED

4.4M

SAFE ACCOUNTS DEPLOYED

With 4.4 million users and 5.4 billion assets, there is very high risk that safe wallet users will try to bridge the assets and loose them.

Now, look at the codebase and understand how the assets will be lost.

In source bridge in `burnAndCallAxelar` we construct the payload as follow :

Here, we can see the payload passes `msg.sender` as receiving an address on the other chain, assuming that the user has the same address across all the EVM chains;

which is not the case if user is using the account abstraction wallet.

```
  bytes memory payload = abi.encode(VERSION, msg.sender, amoun
```

Then, it calls the following function passing the payload, which then calls the `callContract` function passing the payload to Axelar Network.

```
function _payGasAndCallCotract(
  string calldata destinationChain,
  string memory destContract,
  bytes memory payload
) private {
  GAS_RECEIVER.payNativeGasForContractCall{value: msg.value}(
    address(this),
    destinationChain,
    destContract,
    payload,
    msg.sender
  );

  // Send all information to AxelarGateway contract.
  AXELAR_GATEWAY.callContract(destinationChain, destContract, 
}
```

Then, on the destination, any Axelar node can call the `execute()` function, passing in the payload. The tokens will be minted to an account abstraction wallet address of the source chain, but on destination, the same person will not be the owner of that address; hence, tokens are permanently lost.

```
function _execute(
  string calldata srcChain,
  string calldata srcAddr,
  bytes calldata payload
) internal override whenNotPaused {

  (bytes32 version, address srcSender, uint256 amt, uint256 no
    .decode(payload, (bytes32, address, uint256, uint256));

  if (version != VERSION) {
    revert InvalidVersion();
  }
  if (chainToApprovedSender[srcChain] == bytes32(0)) {
    revert ChainNotSupported();
  }
  // each chain have only one approved sender that is the sour
  if (chainToApprovedSender[srcChain] != keccak256(abi.encode(
    revert SourceNotSupported();
  }
  if (isSpentNonce[chainToApprovedSender[srcChain]][nonce]) {
    revert NonceSpent();
  }

  isSpentNonce[chainToApprovedSender[srcChain]][nonce] = true;

  // same payload would have the same txhash
  bytes32 txnHash = keccak256(payload);
  txnHashToTransaction[txnHash] = Transaction(srcSender, amt);
  _attachThreshold(amt, txnHash, srcChain);
  _approve(txnHash);
  _mintIfThresholdMet(txnHash);
  emit MessageReceived(srcChain, srcSender, amt, nonce);
}
```

Recommended Mitigation Steps

Give the user the option to pass in the address the tokens should be minted to on the destination bridge. Pass in the warning for account abstraction wallet holders to not to pass the same wallet. Some wallets may follow a deterministic deployment approach to have same address, but as safe explains, that grants nothing, as each chain has its own different state and opcode differences; so even a deterministic approach may generate different addresses.

Assessed type

Invalid Validation

tom2o17 (Ondo) acknowledged, but disagreed with severity

kirk-baird (judge) decreased severity to Medium and commented:

> Downgrading this to medium severity as users would know ahead of time that the receiving address is the same as the sending address.

[M-03] Two different transactions can result in the same `txnHash` value, thus breaking the approval process of transaction minting

Submitted by Udsen*, also found by* adriro*,* pep7siup*,* kutugu*,* ast3ros*,* bin2chen*,* 0xDING99YA*,* SpicyMeatball*, 0xpiken (*1*,* 2*),* bowtiedvirus*,* Inspecktor*, and* seerether

The `DestinationBridge._execute` is an internal function that is executed when contract is called by Axelar Gateway. The `_execute` function stores the `Transaction` struct in the `txnHashToTransaction` mapping as shown below:

```
    bytes32 txnHash = keccak256(payload);
    txnHashToTransaction[txnHash] = Transaction(srcSender, amt);
```

The transaction hash `txnHash` is calculated by `keccak256(payload)` and the `payload` is an `abi encoded` value consisting of following variables.

```
    bytes32 version, address srcSender, uint256 amt, uint256 nonce
```

The issue here is that the two different `srcChains` with two different `srcAddr` contracts can end up providing the same `txnHash` if the above mentioned `version`, `srcSender`, `amt` and `nonce` are the same. The `_execute` function only restricts the same `srcAddr` to not to use the same `nonce` as shown below:

```
    if (isSpentNonce[chainToApprovedSender[srcChain]][nonce]) {
      revert NonceSpent();
    }
```

But the problem is if there are different `srcAddr`s providing the same `payload` it will result into the same `txnHash`.

Hence, there could be two transactions with the same transaction hash (`txnHash`). The later transaction will override the `txnToThresholdSet[txnHash]` of the former transaction. As a result, the approval process for transaction minting will be broken.

Proof of Concept

```
        bytes32 txnHash = keccak256(payload);
        txnHashToTransaction[txnHash] = Transaction(srcSender, amt);
```

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/bridge/DestinationBridge.sol#L108-L109

```
            txnToThresholdSet[txnHash] = TxnThreshold(
              t.numberOfApprovalsNeeded,
              new address[](0)
            );
```

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/bridge/DestinationBridge.sol#L137-L140

```
        (bytes32 version, address srcSender, uint256 amt, uint256 no
          .decode(payload, (bytes32, address, uint256, uint256));
```

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/bridge/DestinationBridge.sol#L90-L91

Tools Used

VSCode

Recommended Mitigation Steps

It is recommended to include the `srcChain` and the `srcAddr` in the payload as well, which is getting hashed to calculate the `txnHash` . By doing so, different transactions coming from different `srcChains` and `srcAddr` will not result into the same txnHash . The approval process for the transaction minting via the bridge will successfully execute.

kirk-baird (judge) decreased severity to Medium

ali2251 (Ondo) confirmed

[M-04] Admin can't burn tokens from blocklisted addresses because of a check in _beforeTokenTransfer

Submitted by OxAsen, *also found by* Arz, Inspecktor, merlin, DelvirO, BenRai, *and* OxStalin

The function burn is made so the admin can burn rUSDY tokens from **any account** (this is stated in the comments). However, the admin can't burn tokens if the account from which they're trying to burn tokens is blocklisted/sanctioned/not on the allow-list.

Proof of Concept

Let's check the burn function which calls the internal _burnShares function:

```
function burn(
    address _account,
    uint256 _amount
  ) external onlyRole(BURNER_ROLE) {
    uint256 sharesAmount = getSharesByRUSDY(_amount);

    _burnShares(_account, sharesAmount);

    usdy.transfer(msg.sender, sharesAmount / BPS_DENOMINATOR);

    emit TokensBurnt(_account, _amount);
  }

  function _burnShares(
    address _account,
    uint256 _sharesAmount
  ) internal whenNotPaused returns (uint256) {
    require(_account != address(0), "BURN_FROM_THE_ZERO_ADDRESS"
    _beforeTokenTransfer(_account, address(0), _sharesAmount); <-

    uint256 accountShares = shares[_account];
    require(_sharesAmount <= accountShares, "BURN_AMOUNT_EXCEEDS_

    uint256 preRebaseTokenAmount = getRUSDYByShares(_sharesAmoun

    totalShares -= _sharesAmount;

    shares[_account] = accountShares - _sharesAmount;

    uint256 postRebaseTokenAmount = getRUSDYByShares(_sharesAmou

    return totalShares;
```

We can see that it calls `_beforeTokenTransfer(_account, address(0), _sharesAmount)`.

Here is the code of `_beforeTokenTransfer`:

```
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256
      ) internal view {
        // Check constraints when `transferFrom` is called to facili
        // a transfer between two parties that are not `from` or `to
        if (from != msg.sender && to != msg.sender) {
          require(!_isBlocked(msg.sender), "rUSDY: 'sender' address b
          require(!_isSanctioned(msg.sender), "rUSDY: 'sender' addre
          require(
            _isAllowed(msg.sender),
            "rUSDY: 'sender' address not on allowlist"
          );
        }

        if (from != address(0)) { <--
          // If not minting
          require(!_isBlocked(from), "rUSDY: 'from' address blocked"
          require(!_isSanctioned(from), "rUSDY: 'from' address sanct
          require(_isAllowed(from), "rUSDY: 'from' address not on al
        }
        if (to != address(0)) {
          // If not burning
          require(!_isBlocked(to), "rUSDY: 'to' address blocked");
          require(!_isSanctioned(to), "rUSDY: 'to' address sanctione
          require(_isAllowed(to), "rUSDY: 'to' address not on allowl
        }
      }
```

In our case, the `form` would be the account from which we're burning tokens, so it'll enter in the 2nd if: `if (from != address(0))` . But given that the account is blocked/sanctioned/not on the allow-list, the transaction will revert and the tokens won't be burned.

Given that there are separate roles for burning and managing the block/sanctions/allowed lists (`BURNER_ROLE` and `LIST_CONFIGURER_ROLE`), it is very possible that such a scenario would occur.

In that case, the Burner would have to ask the List Configurer to update the lists, so the Burner can burn the tokens, and then the List Configurer should update the lists again. However, in that case, you're risking that the blocked user manages to transfer their funds while performing these operations.

Recommended Mitigation Steps

Organize the logic of the function better. For example, you can make the 2nd if to be: `if (from != address(0) && to != address(0))` . That way, we'll not enter the `if` when burning tokens, and we'll be able to burn tokens from blocked accounts.

Assessed type

Invalid Validation

tom2o17 (Ondo) disputed and commented via duplicate issue #120:

> Can I not assume that the guardian can batch execute transactions? Given that the guardian will also have the ability to add/remove from blocklist, can I not assume that the guardian can batch:

```
    blocklist.removeFromBlocklist()
    rUSDY.burn()
    blocklist.addToBlocklist()
```

> IFL this is not an issue considering the guardian address can execute any peripheral txns in an atomic fashion.

kirk-baird (judge) commented via duplicate issue #120:

> This is an interesting edge case. While it may be possible for guardian to bypass this issue, if it is a smart contract that can batch transactions, I see this as a potential issue.

> Going to downgrade issue #120 to medium severity, as there are some theoretical workarounds to this problem.

tom2o17 (Ondo) commented via duplicate issue #120:

> Not to impact judging,

> But to your point @kirk-baird, we are using a gnosis safe contract as the guardian, and plan to do a similar setup for the majority of tokens going forward. Perhaps we should have made note of that in the `ReadME.md`.

kirk-baird (judge) commented via duplicate issue #120:

> @tom2o17 - Okay thanks for clarifying - that does resolve the issue. Though, for the judging the wardens weren't aware of this so I'll consider it a valid issue.

ali2251 (Ondo) acknowledged

Low Risk and Non-Critical Issues

For this audit, 34 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **adriro** received the top score from the judge.

The following wardens also submitted reports: gkrastenov, Arz, castle_chain, 0xmystery, matrix_0wl, Raihan, Breeje, klau5, nirlin, MohammedRizwan, Udsen, dev0cloo, Aymen0909, peanuts, catellatech, Lalanda, SovaSlava, wahedtalash77, kutugu, ladboy233, ohm, 0xpanicError, codegpt, BenRai, ast3ros, sandy, bin2chen, 0x6980, Stormreckson, pipidu83, hals, 0xanmol, *and* 0xStalin.

Low Issues Summary

ID	Issue
[L-01]	Confusing semantics for bridged messages approvals
[L-02]	No default threshold configuration
[L-03]	Validate number of approvers in `setThresholds()`
[L-04]	Missing safe wrapper for ERC20 transfer in `rescueTokens()`
[L-05]	Account may get blacklisted during the bridge process
[L-06]	Missing validation for index parameter in `overrideRange()`
[L-07]	Oracle assumes asset has 18 decimals
[L-08]	Wrong argument in `Transfer` event of `wrap()` function
[L-09]	Wrong argument in `TransferShares` event of `wrap()` function
[L-10]	Contracts can be re-deployed in rUSDY factory

Non-Critical Issues Summary

ID	Issue
[N-01]	Missing calls to base initializers in `rUSDY`
[N-02]	Missing event to notify oracle has changed in `rUSDY`

[L-01] Confusing semantics for bridged messages approvals

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/bridge/DestinationBridge.sol#L111

Bridged messages get an “automatic” approval while the message is being executed in the destination bridge contract, which means that all messages effectively get one approval as soon as they are bridged.

This implies that "real" approvals would require setting a threshold with a `numberOfApprovalsNeeded` of at least two.

This brings a lot of unnecessary confusing semantics to the approvals scheme and could potentially lead to errors in configuration settings that would allow messages to go through automatically when actually they are expected to be approved.

It is recommended to remove this automatic approval and treat each approval as an explicit approval action of the set of enabled approvers.

[L-02] No default threshold configuration

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/bridge/DestinationBridge.sol#L128

The implementation of `_attachThreshold()` loops through each of the configured set of thresholds searching for the element that adjusts to the bridged amount.

If no configuration is found the function will revert, the message will be lost and it will require manual handling.

It is recommended setting a default value (a high value number of approvals) to be used when no threshold configuration matches.

[L-03] Validate number of approvers in `setThresholds()`

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/bridge/DestinationBridge.sol#L255

The `setThresholds()` function should validate that elements in the `numOfApprovers` array are greater than zero.

[L-04] Missing safe wrapper for ERC20 transfer in `rescueTokens()`

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/bridge/DestinationBridge.sol#L322

Use a “safe” wrapper to execute ERC20 transfer for better compatibility.

[L-05] Account may get blacklisted during the bridge process

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/bridge/DestinationBridge.sol#L349

Accounts that bridge tokens may get blacklisted in the source chain after the bridging process had been initiated. This could lead to different issues:

- If the account is not blacklisted in the destination chain, the user may still operate their funds.
- If the account is blacklisted, then the function will revert and could be treated as a failure that would require manual intervention.

[L-06] Missing validation for index parameter in `overrideRange()`

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/rwaOracles/RWADynamicOracle.sol#L186

The `overrideRange()` function should validate that `indexToModify` is within bounds, i.e. `require(indexToModify < rangeLength)`.

[L-07] Oracle assumes asset has 18 decimals

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/rwaOracles/RWADynamicOracle.sol#L282

The `roundUpTo8()` implementation assumes the asset has 18 decimals, as the rounding is done using `1e10` (since `18 - 8 = 10`). It is recommended to use a constant to properly state this dependency and for a better understanding.

[L-08] Wrong argument in `Transfer` event of `wrap()` function

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/usdy/rUSDY.sol#L438

The third argument to the `Transfer` event is expecting the amount of `rUSDY` tokens but it is wrongly called with `getRUSDYByShares(_USDYAmount)`, which lacks the proper scaling to convert the `USDY` amount to shares.

The correct version should be:

```
emit Transfer(address(0), msg.sender, getRUSDYByShares(_USDYAmou
```

[L-09] Wrong argument in `TransferShares` event of `wrap()` function

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/usdy/rUSDY.sol#L439

The third argument to the `TransferShares` event is expecting the amount of shares but it is wrongly called with `_USDYAmount`, which is the amount of `USDY` tokens, not the shares.

The correct version should be:

```
emit TransferShares(address(0), msg.sender, _USDYAmount * BPS_DE
```

[L-10] Contracts can be re-deployed in `rUSDY` factory

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/usdy/rUSDYFactory.sol#L75

The `rUSDY` contract can be redeployed by the guardian by calling `deployrUSDY()` in the `rUSDYFactory` contract. This will deploy a new set of contracts and overwrite the storage variables that link to the contract.

It is recommended to add a check to avoid redeployment once contracts have been deployed.

[N-01] Missing calls to base initializers in `rUSDY`

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/usdy/rUSDY.sol#L120

The `__rUSDY_init()` function doesn’t call the initializers for some of the base contracts:

- `Initializable`
- `ContextUpgradeable`
- `PausableUpgradeable`
- `AccessControlEnumerableUpgradeable`

[N-02] Missing event to notify oracle has changed in `rUSDY`

https://github.com/code-423n4/2023-09-ondo/blob/main/contracts/usdy/rUSDY.sol#L662

The `setOracle()` function should emit an event to signal off-chain actors that the oracle has been updated.

kirk-baird (judge) commented:

> The issues raised in this report are all valid and contain quality analysis. They correctly determine the severity except L-01, which I would consider a non-critical severity, rather than low.

ali2251 (Ondo) confirmed

Gas Optimizations

For this audit, 21 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **c3phas** received the top score from the judge.

The following wardens also submitted reports: adriro, naman1778, K42, kaveyjoe, petrichor, zabihullahazadzoi, MohammedRizwan, SY_S, SAQ, SAAJ, wahedtalash77, 0xhex, 0xta, jeffy, 0x11singh99, albahaca, Eurovickk, ybansal2403, castle_chain, *and* matrix_0wl.

Auditor’s Disclaimer

While we try our best to maintain readability in the provided code snippets, some functions have been truncated to highlight the affected portions.

It’s important to note that during the implementation of these suggested changes, developers must exercise caution to avoid introducing vulnerabilities. Although the optimizations have been tested prior to these recommendations, it is the responsibility of the developers to conduct thorough testing again.

Code reviews and additional testing are strongly advised to minimize any potential risks associated with the refactoring process.

Codebase impressions

The developers have done an excellent job of identifying and implementing some of the most evident optimizations.

However, we have identified additional areas where optimization is possible, some of which may not be immediately apparent.

Note on Gas estimates

We’ve tried to give the exact amount of gas being saved from running the included tests. Whenever the function is within the test coverage, the average gas before and after will be included, and often a diff of the code will also accompany this.

Some functions are not covered by the test cases or are internal/private functions. In this case, the gas can be estimated by looking at the opcodes involved.

Pack structs by putting data types that can fit together next to each other

As the solidity EVM works with 32 bytes, variables less than 32 bytes should be packed inside a struct so that they can be stored in the same slot. This saves gas when writing to storage ~20000 gas

https://github.com/code-423n4/2023-09-ondo/blob/02ebedeeb85b708345a4deb53a2b543ecae160d0/contracts/rwaOracl es/RWADynamicOracle.sol#L295-L300

We can pack start and end together by reducing their size to `uint48`, since they are just timestamps (save 1 SLOT: 2K Gas)

```
File: /contracts/rwaOracles/RWADynamicOracle.sol
295:  struct Range {
296:    uint256 start;
297:    uint256 end;
298:    uint256 dailyInterestRate;
299:    uint256 prevRangeClosePrice;
300:  }
```

```
diff --git a/contracts/rwaOracles/RWADynamicOracle.sol b/contrac
index 03aa7d6..6b8ff93 100644
--- a/contracts/rwaOracles/RWADynamicOracle.sol
+++ b/contracts/rwaOracles/RWADynamicOracle.sol
@@ -293,8 +293,8 @@ contract RWADynamicOracle is IRWAOracle, Acc
   ////////////////////////////////////////////////////////////////

   struct Range {
-    uint256 start;
-    uint256 end;
+    uint48 start;
+    uint48 end;
     uint256 dailyInterestRate;
     uint256 prevRangeClosePrice;
   }
```

https://github.com/code-423n4/2023-09-ondo/blob/02ebedeeb85b708345a4deb53a2b543ecae160d0/contracts/bridge/SourceBridge.sol#L61-L74

The function should first verify if `msg.value > 0` before performing any other operation (Saves 7146 Gas)

In case of a revert on `msg.value == 0` we save 7146 gas on average from the protocol gas tests (to simulate the sad path ie `msg.value == 0` see the test modification).

	Min	Average	Median	Max
Before	9815	9815	9815	9815
After	2669	2669	2669	2669

```
File: /contracts/bridge/SourceBridge.sol
function burnAndCallAxelar(
  uint256 amount,
  string calldata destinationChain
) external payable whenNotPaused {
  // check destinationChain is correct
  string memory destContract = destChainToContractAddr[dest

  if (bytes(destContract).length == 0) {
    revert DestinationNotSupported();
  }

  if (msg.value == 0) {
    revert GasFeeTooLow();
  }
```

```
diff --git a/contracts/bridge/SourceBridge.sol b/contracts/bridg
index 457bf41..b436ff7 100644
--- a/contracts/bridge/SourceBridge.sol
+++ b/contracts/bridge/SourceBridge.sol
@@ -62,6 +62,10 @@ contract SourceBridge is Ownable, Pausable, II
     uint256 amount,
     string calldata destinationChain
   ) external payable whenNotPaused {
+
+    if (msg.value == 0) {
+      revert GasFeeTooLow();
+    }
     // check destinationChain is correct
     string memory destContract = destChainToContractAddr[destin

@@ -69,10 +73,6 @@ contract SourceBridge is Ownable, Pausable, II
       revert DestinationNotSupported();
     }

-    if (msg.value == 0) {
-      revert GasFeeTooLow();
-    }
```

To get the perfect representation of the gas being saved, I modified the test function `test_bridge()` as follows:

```
   function test_bridge() public initializeAlice {
     uint256 gasReceiverBalanceBefore = AXELAR_GAS_SERVICE.balan

     // Expect event emit from gas receiver
     bytes memory payload = abi.encode(bytes32("1.0"), alice, 10
     bytes32 payloadHash = keccak256(payload);
-    vm.expectEmit(true, true, true, true);
+    //vm.expectEmit(true, true, true, true);
     emit NativeGasPaidForContractCall(
       address(usdyBridge),
       "optimism",
@@ -160,7 +160,7 @@ contract Test_SourceBridge is USDY_BasicDepl
     );

     // Expect event emit from gateway
-    vm.expectEmit(true, true, true, true);
+    //vm.expectEmit(true, true, true, true);
     emit ContractCall(
       address(usdyBridge),
       "optimism",
@@ -171,7 +171,7 @@ contract Test_SourceBridge is USDY_BasicDepl

     // Bridge Tokens
     vm.prank(alice);
-    usdyBridge.burnAndCallAxelar{value: 0.01 ether}(100e18, "op
+    usdyBridge.burnAndCallAxelar{value: 0.00 ether}(100e18, "op

     assertEq(usdy.balanceOf(address(usdyBridge)), 0);
     assertEq(usdy.balanceOf(alice), 0);
```

Note, using the `vm.expectRevert` also produces the same benchmarks.

Optimize the function `_mintIfThresholdMet()`

https://github.com/code-423n4/2023-09-ondo/blob/47d34d6d4a5303af5f46e907ac2292e6a7745f6c/contracts/bridge/Des tinationBridge.sol#L337-L353

Only read state if we are going to execute the entire logic (Saves ~2000 gas for the state read)

```
File: /contracts/bridge/DestinationBridge.sol
337:  function _mintIfThresholdMet(bytes32 txnHash) internal {
338:    bool thresholdMet = _checkThresholdMet(txnHash);
339:    Transaction memory txn = txnHashToTransaction[txnHash];
340:    if (thresholdMet) {
341:      _checkAndUpdateInstantMintLimit(txn.amount);
342:      if (!ALLOWLIST.isAllowed(txn.sender)) {
343:        ALLOWLIST.setAccountStatus(
344:          txn.sender,
345:          ALLOWLIST.getValidTermIndexes()[0],
346:          true
347:        );
348:      }
349:      TOKEN.mint(txn.sender, txn.amount);
350:      delete txnHashToTransaction[txnHash];
351:      emit BridgeCompleted(txn.sender, txn.amount);
352:    }
353:  }
```

The function above, only mints if the threshold is met. Before checking the status of whether the threshold is met or not, we making a state read(`Transaction memory txn = txnHashToTransaction[txnHash];`), which is quite expensive. We then proceed to check if the threshold is met and execute the function if the threshold is indeed met. If not met, we would not execute anything, but we would still have made the state read, as it’s being read before the check.

We can refactor the function to have the check before to avoid wasting gas in case threshold is not met:

```
diff --git a/contracts/bridge/DestinationBridge.sol b/contracts/
index 8ad410c..13cc444 100644
--- a/contracts/bridge/DestinationBridge.sol
+++ b/contracts/bridge/DestinationBridge.sol
@@ -336,8 +336,8 @@ contract DestinationBridge is
    */
   function _mintIfThresholdMet(bytes32 txnHash) internal {
     bool thresholdMet = _checkThresholdMet(txnHash);
-     Transaction memory txn = txnHashToTransaction[txnHash];
     if (thresholdMet) {
+       Transaction memory txn = txnHashToTransaction[txnHash];
       _checkAndUpdateInstantMintLimit(txn.amount);
       if (!ALLOWLIST.isAllowed(txn.sender)) {
         ALLOWLIST.setAccountStatus(
```

Optimize check order: avoid making any state reads/writes if we have to validate some function parameters

https://github.com/code-423n4/2023-09-ondo/blob/47d34d6d4a5303af5f46e907ac2292e6a7745f6c/contracts/rwaOracles/RWADynamicOracle.sol#L30-L46

```
File: /contracts/rwaOracles/RWADynamicOracle.sol
30:   constructor(
31:     address admin,
32:     address setter,
33:     address pauser,
34:     uint256 firstRangeStart,
35:     uint256 firstRangeEnd,
36:     uint256 dailyIR,
37:     uint256 startPrice
38:   ) {
39:     _grantRole(DEFAULT_ADMIN_ROLE, admin);
40:     _grantRole(PAUSER_ROLE, pauser);
41:     _grantRole(SETTER_ROLE, setter);

43:     if (firstRangeStart >= firstRangeEnd) revert InvalidRange
44:     uint256 trueStart = (startPrice * ONE) / dailyIR;
45:     ranges.push(Range(firstRangeStart, firstRangeEnd, dailyIR
46:   }
```

We have a check for function parameters which reverts the whole transaction if `firstRangeStart >= firstRangeEnd`. In case of a revert, all the gas consumed so far will not be refunded. If we look at our constructor, the operations that happen before this check consume so much gas, as they involve reading and writing states. The `_grantRole()` is inherited and the function has the following implementation:

https://github.com/code-423n4/2023-09-ondo/blob/47d34d6d4a5303af5f46e907ac2292e6a7745f6c/contracts/external/openzeppelin/contracts/access/AccessControl.sol#L238-L243

```
File: /contracts/external/openzeppelin/contracts/access/AccessCo
238:  function _grantRole(bytes32 role, address account) interna
239:    if (!hasRole(role, account)) {
240:      _roles[role].members[account] = true;
241:      emit RoleGranted(role, account, _msgSender());
242:    }
243:  }
```

Now the above function is called 3 times in our constructor. Most of these reads/writes will be cold, thus the cost would be higher.

```
diff --git a/contracts/rwaOracles/RWADynamicOracle.sol b/contrac
index 03aa7d6..fd07948 100644
--- a/contracts/rwaOracles/RWADynamicOracle.sol
+++ b/contracts/rwaOracles/RWADynamicOracle.sol
@@ -36,11 +36,11 @@ contract RWADynamicOracle is IRWAOracle, Acc
     uint256 dailyIR,
     uint256 startPrice
   ) {
+        if (firstRangeStart >= firstRangeEnd) revert InvalidRan
     _grantRole(DEFAULT_ADMIN_ROLE, admin);
     _grantRole(PAUSER_ROLE, pauser);
     _grantRole(SETTER_ROLE, setter);

-    if (firstRangeStart >= firstRangeEnd) revert InvalidRange()
     uint256 trueStart = (startPrice * ONE) / dailyIR;
     ranges.push(Range(firstRangeStart, firstRangeEnd, dailyIR,
   }
```

Don’t cache calls that are only used once

https://github.com/code-423n4/2023-09-ondo/blob/02ebedeeb85b708345a4deb53a2b543ecae160d0/contracts/bridge/DestinationBridge.sol#L322-L325

`DestinationBridge.sol.rescueTokens()` : Saves 8 gas

	Min	Average	Median	Max
Before	2608	16302	16302	29996
After	2608	16294	16294	29980

```
File: /contracts/bridge/DestinationBridge.sol
function rescueTokens(address _token) external onlyOwner {
  uint256 balance = IRWALike(_token).balanceOf(address(thi
  IRWALike(_token).transfer(owner(), balance);
}
```

```diff
diff --git a/contracts/bridge/DestinationBridge.sol b/contracts/
index 8ad410c..376f1c3 100644
--- a/contracts/bridge/DestinationBridge.sol
+++ b/contracts/bridge/DestinationBridge.sol
@@ -320,8 +320,7 @@ contract DestinationBridge is
    * @param _token The address of the token to rescue
    */
   function rescueTokens(address _token) external onlyOwner {
-    uint256 balance = IRWALike(_token).balanceOf(address(this))
-    IRWALike(_token).transfer(owner(), balance);
+    IRWALike(_token).transfer(owner(), IRWALike(_token).balance
   }
```

https://github.com/code-423n4/2023-09-ondo/blob/02ebedeeb85b708345a4deb53a2b543ecae160d0/contracts/bridge/DestinationBridge.sol#L337-L353

`DestinationBridge.sol.\_mintIfThresholdMet()` : Saves 10 gas

Benchmarks are based on the function `approve()`, which calls our internal function.

	Min	Average	Median	Max
Before	1801	62199	27240	159159
After	1801	62189	27224	159146

```
File: /contracts/bridge/DestinationBridge.sol
function _mintIfThresholdMet(bytes32 txnHash) internal {
```

```
338:     bool thresholdMet = _checkThresholdMet(txnHash);
339:     Transaction memory txn = txnHashToTransaction[txnHash];
340:     if (thresholdMet) {
341:       _checkAndUpdateInstantMintLimit(txn.amount);
```

```diff
diff --git a/contracts/bridge/DestinationBridge.sol b/contracts/
index 8ad410c..4562f20 100644
--- a/contracts/bridge/DestinationBridge.sol
+++ b/contracts/bridge/DestinationBridge.sol
@@ -335,9 +335,8 @@ contract DestinationBridge is
     * @param txnHash The hash of the transaction we wish to mint
     */
    function _mintIfThresholdMet(bytes32 txnHash) internal {
-     bool thresholdMet = _checkThresholdMet(txnHash);
      Transaction memory txn = txnHashToTransaction[txnHash];
-     if (thresholdMet) {
+     if (_checkThresholdMet(txnHash)) {
        _checkAndUpdateInstantMintLimit(txn.amount);
        if (!ALLOWLIST.isAllowed(txn.sender)) {
          ALLOWLIST.setAccountStatus(
```

Refactor the assembly code

https://github.com/code-423n4/2023-09-ondo/blob/02ebedeeb85b708345a4deb53a2b543ecae160d0/contracts/rwaOracles/RWADynamicOracle.sol#L369-L374

	Min	Average	Median	Max
Before	4021	7231	7834	17031
After	4017	7217	7822	17015

The `div` opcode uses 5 gas while the `shr` opcode uses 3 gas. Since the denominator is `known(2)`, we can simply shift right by 1 which is equivalent to division by 2.

```
File: /contracts/rwaOracles/RWADynamicOracle.sol
369:         let half := div(base, 2) // for rounding.
370:         for {
371:           n := div(n, 2)
```

```
        } n {
          n := div(n, 2)
        } {
```

```
diff --git a/contracts/rwaOracles/RWADynamicOracle.sol b/contrac
index 03aa7d6..fa175e0 100644
--- a/contracts/rwaOracles/RWADynamicOracle.sol
+++ b/contracts/rwaOracles/RWADynamicOracle.sol
@@ -366,11 +366,11 @@ contract RWADynamicOracle is IRWAOracle, A
         default {
           z := x
         }
-        let half := div(base, 2) // for rounding.
+        let half := shr(1, base) // for rounding.
         for {
-          n := div(n, 2)
+          n := shr(1, n)
         } n {
-          n := div(n, 2)
+          n := shr(1, n)
         } {
```

Conclusion

It is important to emphasize that the provided recommendations aim to enhance the efficiency of the code without compromising its readability. We understand the value of maintainable and easily understandable code to both developers and auditors.

As you proceed with implementing the suggested optimizations, please exercise caution and be diligent in conducting thorough testing. It is crucial to ensure that the changes are not introducing any new vulnerabilities and that the desired performance improvements are achieved. Review code changes, and perform thorough testing to validate the effectiveness and security of the refactored code.

Should you have any questions or need further assistance, please don’t hesitate to reach out.

tom2o17 (Ondo) confirmed

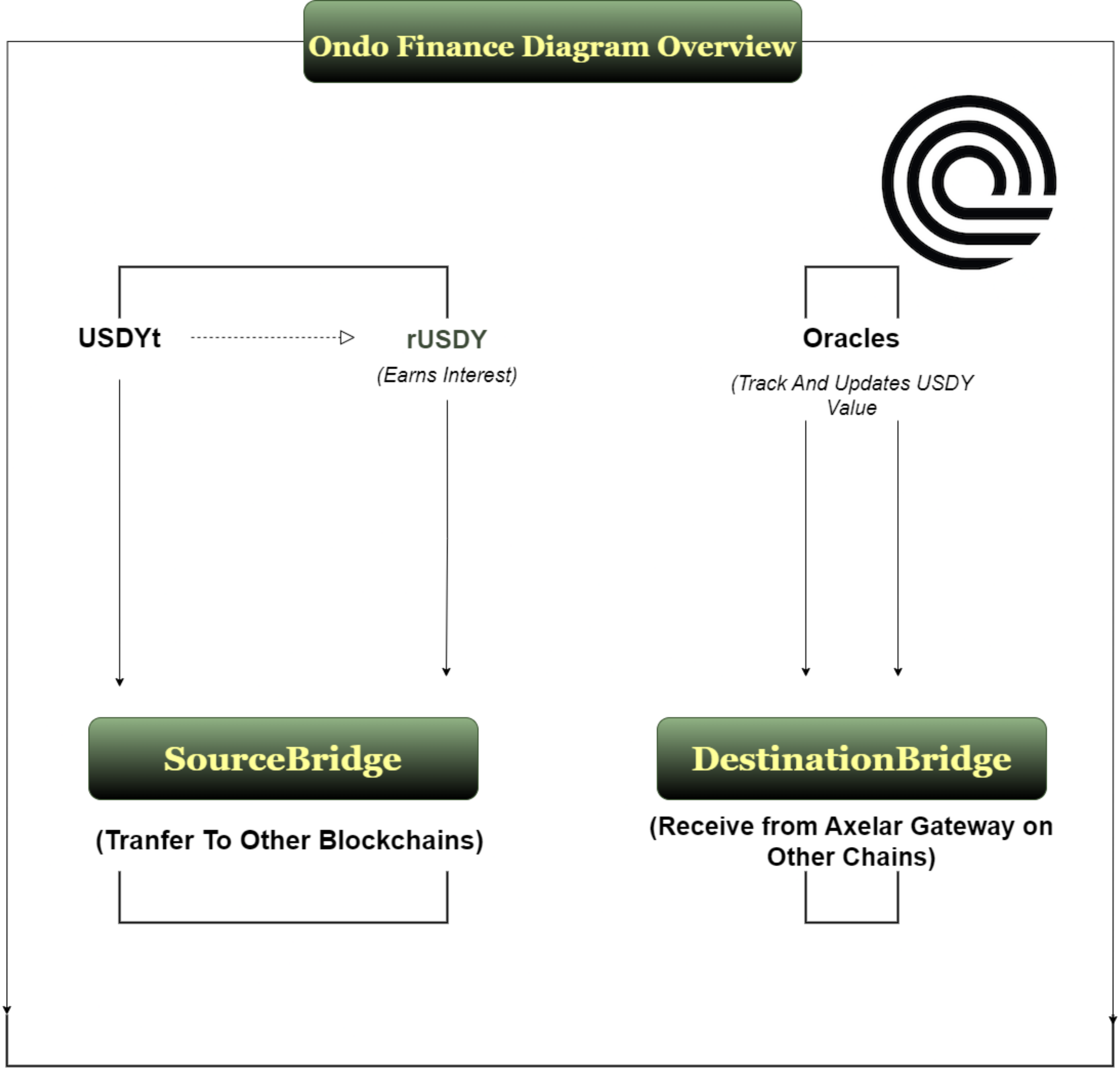
Audit Analysis

For this audit, 20 analysis reports were submitted by wardens. An analysis report examines the codebase as a whole, providing observations and advice on such topics as architecture, mechanism, or approach. The report highlighted below by **catellatech** received the top score from the judge.

The following wardens also submitted reports: hunter_w3b, JayShreeRAM, kaveyjoe, DedOhWale, 0xStalin, nirlin, peanuts, m4ttm, Breeje, 0xmystery, mahdikarimi, Bube, K42, sandy, 0xAsen, Krace, castle_chain, hals, *and* 0xE1D.

Description overview of Ondo Finance

Ondo Finance is a blockchain platform that offers investment opportunities with cryptocurrencies backed by U.S. dollar bank deposits `(USDY)`. `USDY` earns interest over time, increasing its value. Ondo Finance introduces `rUSDY`, a variant of `USDY` that automatically adjusts in quantity to reflect interest but maintains a nominal value of 1 dollar per token. `Oracles` track the value of `USDY`. Ondo Finance enables asset transfers between blockchains through `bridge` contracts.


Ondo Finance Diagram Overview
USDYt
rUSDY
(Earns Interest)
Oracles
(Track And Updates USDY Value
SourceBridge
(Tranfer To Other Blockchains)
DestinationBridge
(Receive from Axelar Gateway on Other Chains)

System Overview

Scope

- SourceBridge.sol - This contract facilitates the transfer of tokens between different blockchains using the Axelar protocol, ensuring that tokens burned on one chain are minted on the corresponding destination chain, provided certain conditions are met, and the necessary gas is paid. Additionally, it provides administrative functionalities and a pause mechanism when needed.
- DestinationBridge.sol - This contract functions as the destination chain bridge for `USDY` tokens. Its primary purpose is to facilitate the seamless transfer of `USDY` tokens from a source blockchain to the destination blockchain where this contract resides. This bridge contract plays a pivotal role in enabling cross-chain

interoperability and ensuring that `USDY` tokens can be securely and efficiently moved between different blockchain networks.

- rUSDY.sol - This contract serves as the backbone for an interest-bearing token where users can `wrap and unwrap` their `USDY` tokens to earn interest. It also includes features for access control and address management.
- rUSDYFactory.sol - This is a factory contract that allows the deployment of upgradable instances of the `rUSDY token` contract. It is managed by a guardian address, and the deployment process involves creating an implementation contract, a proxy admin contract, and a proxy contract for the token. The code is designed to facilitate the upgradeability of the `rUSDY` token and includes functions for batched external calls.
- RWADynamicOracle.sol - This contract is a dynamic Oracle that provides the price of `USDY` based on configured time ranges and daily interest rates. It also includes mechanisms for pausing and access control to manage the ranges and contract operation.
- IRWADynamicOracle - This is an interface that sets a standard for any contract wishing to provide information about the price of `RWA` (Asset-Backed Real World Assets) or similar assets. Contracts that implement this interface must provide a `getPrice()` function that returns the current price of `RWA`. The interface does not contain implementation logic but simply establishes a common structure for communication with dynamic asset price oracles.

Privileged Roles

The `SourceBridge` contract uses the `onlyOwner` modifier to restrict access to important administrative functions such as configuring destination chain-to-contract, pausing / unpausing the contract, and executing batch calls to other contracts.

`onlyOwner` is also present in these functions of the contract `DestinationBridge`: addApprover, removeApprover, addChainSupport, setThresholds, setMintLimit, setMintLimitDuration, pause, unpause, and rescueTokens.

Only the address that deploys the contract and possesses the owner role can execute these functions. The owner has full control over the contract and its administrative functions.

In the `rUSDY contract`, we found some roles, such as:

```
 /// @dev Role based access control roles
bytes32 public constant USDY_MANAGER_ROLE = keccak256("ADMIN_R
bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE")
bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE")
bytes32 public constant BURNER_ROLE = keccak256("BURN_ROLE");
bytes32 public constant LIST_CONFIGURER_ROLE =
  keccak256("LIST_CONFIGURER_ROLE");
```

- `USDY_MANAGER_ROLE`: This role has access to functions related to the management of the `USDY` token, such as changing the oracle that updates the `USDY` price.
- `MINTER_ROLE`: This role has the ability to create new `rUSDY` tokens by minting them. It is responsible for the `wrap` function, which allows users to convert `USDY` into `rUSDY`.
- `PAUSER_ROLE`: This role can `pause` and `unpause` the operations of the `rUSDY` contract. When the contract is paused, transfers and other operations are not available.
- `BURNER_ROLE`: This role allows administrators to burn (delete) `rUSDY` from a specific account using the `burn` function.
- `LIST_CONFIGURER_ROLE`: This role is used to configure the blocklist, allowlist, and sanctions list. Administrators with this role can change the addresses in these lists.

The `rUSDYFactory` contract defines the following privilege roles:

- `DEFAULT_ADMIN_ROLE`: This role represents the default administrator role. It has access to certain administrative functions in the contract.

```
bytes32 public constant DEFAULT_ADMIN_ROLE = bytes32(0);
```

- `onlyGuardian`: This modifier ensures that only the guardian address has access to certain functions of the contract, such as `deployrUSDY`.

```
    modifier onlyGuardian() {
      require(msg.sender == guardian, "rUSDYFactory: You are not t
      _;
    }
```

The `RWADynamicOracle` contract defines the following roles of privileges:

```
    bytes32 public constant SETTER_ROLE = keccak256("SETTER_ROLE")
    bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE")
```

- `SETTER_ROLE` : This role allows certain addresses to set a price range for `USDY` .
- `PAUSER_ROLE` : This role allows certain addresses to `pause` and `unpause` the operation of the oracle.

We asked the sponsors about who will be responsible for these roles, and this was their response:

```
  ali2251 — 09/5/2023 at 10:22
  it will be managed by multisigs controlled by the protocol
```

Given the level of control that these roles possess within the system, users must place full trust in the fact that the entities overseeing these roles will always act correctly and in the best interest of the system and its users.

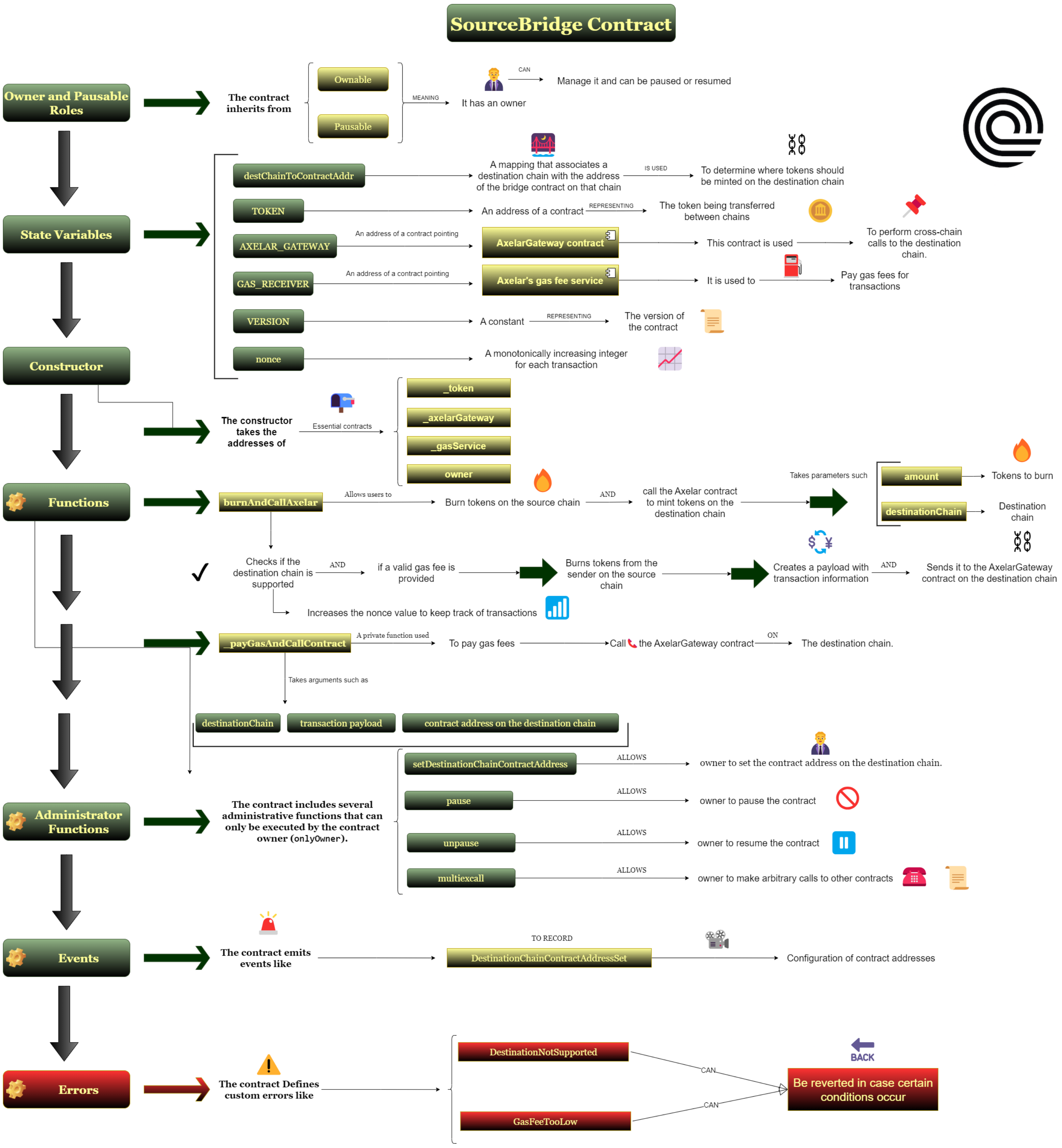
Codebase analysis through diagrams

We have decided to create diagrams detailing each part of the functions of the smart contract provided by the Ondo Finance protocol and to create a summary of the functionality of each of the contracts. This approach proves to be effective for us as it allows us to thoroughly understand all the functionalities of each contract and visually document what we have comprehended through diagrams. Furthermore, we believe that this approach can be useful in enhancing the understanding of the contracts among developers, auditors, and users.

In this audit, the protocol provided **5 contracts** and 1 **Interface.** Here's a detailed diagrams of the essential components of each one:

Contracts:

SourceBridge:



SourceBridge Contract

Owner and Pausable Roles → The contract inherits from: Ownable, Pausable —MEANING→ It has an owner —CAN→ Manage it and can be paused or resumed

State Variables →
- destChainToContractAddr → A mapping that associates a destination chain with the address of the bridge contract on that chain —IS USED→ To determine where tokens should be minted on the destination chain
- TOKEN → An address of a contract —REPRESENTING→ The token being transferred between chains
- AXELAR_GATEWAY —An address of a contract pointing→ AxelarGateway contract → This contract is used → To perform cross-chain calls to the destination chain.
- GAS_RECEIVER —An address of a contract pointing→ Axelar's gas fee service → It is used to → Pay gas fees for transactions
- VERSION → A constant —REPRESENTING→ The version of the contract
- nonce → A monotonically increasing integer for each transaction

Constructor → The constructor takes the addresses of —Essential contracts→ _token, _axelarGateway, _gasService, owner

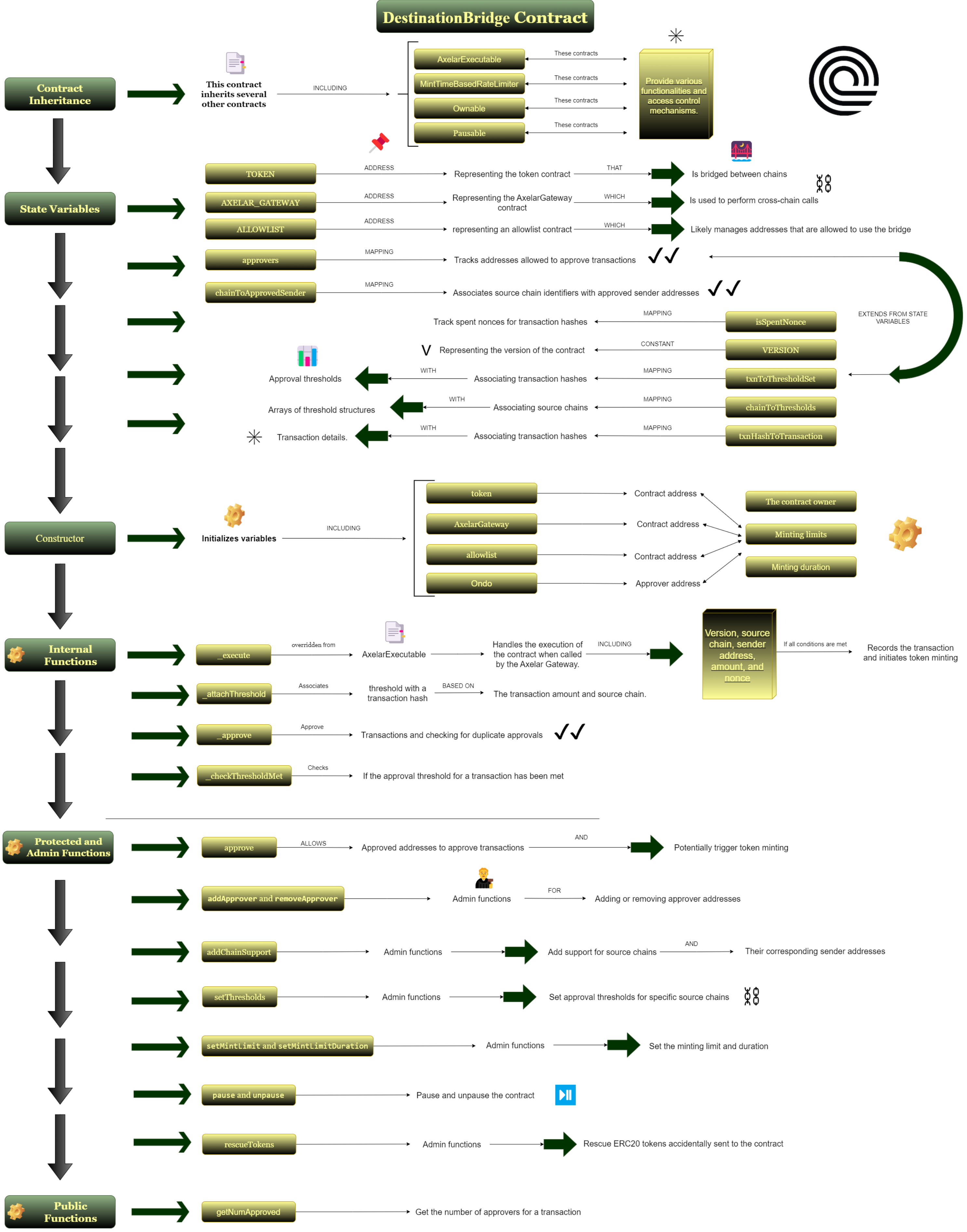
Functions →
- burnAndCallAxelar —Allows users to→ Burn tokens on the source chain —AND→ call the Axelar contract to mint tokens on the destination chain → Takes parameters such: amount → Tokens to burn; destinationChain → Destination chain
  - ✓ Checks if the destination chain is supported —AND→ if a valid gas fee is provided → Burns tokens from the sender on the source chain → Creates a payload with transaction information —AND→ Sends it to the AxelarGateway contract on the destination chain
  - Increases the nonce value to keep track of transactions
- _payGasAndCallContract —A private function used→ To pay gas fees → Call the AxelarGateway contract —ON→ The destination chain.
  - Takes arguments such as: destinationChain, transaction payload, contract address on the destination chain

Administrator Functions → The contract includes several administrative functions that can only be executed by the contract owner (`onlyOwner`).
- setDestinationChainContractAddress —ALLOWS→ owner to set the contract address on the destination chain.
- pause —ALLOWS→ owner to pause the contract
- unpause —ALLOWS→ owner to resume the contract
- multiexcall —ALLOWS→ owner to make arbitrary calls to other contracts

Events → The contract emits events like → DestinationChainContractAddressSet —TO RECORD→ Configuration of contract addresses

Errors → The contract Defines custom errors like → DestinationNotSupported —CAN→, GasFeeTooLow —CAN→ Be reverted in case certain conditions occur

DestinationBridge:



DestinationBridge Contract

Contract Inheritance → This contract inherits several other contracts INCLUDING → AxelarExecutable, MintTimeBasedRateLimiter, Ownable, Pausable → These contracts → Provide various functionalities and access control mechanisms.

State Variables

- TOKEN — ADDRESS → Representing the token contract — THAT → Is bridged between chains
- AXELAR_GATEWAY — ADDRESS → Representing the AxelarGateway contract — WHICH → Is used to perform cross-chain calls
- ALLOWLIST — ADDRESS → representing an allowlist contract — WHICH → Likely manages addresses that are allowed to use the bridge
- approvers — MAPPING → Tracks addresses allowed to approve transactions ✓✓
- chainToApprovedSender — MAPPING → Associates source chain identifiers with approved sender addresses ✓✓

EXTENDS FROM STATE VARIABLES

- isSpentNonce — MAPPING → Track spent nonces for transaction hashes
- VERSION — CONSTANT → Representing the version of the contract
- txnToThresholdSet — MAPPING → Associating transaction hashes — WITH → Approval thresholds
- chainToThresholds — MAPPING → Associating source chains — WITH → Arrays of threshold structures
- txnHashToTransaction — MAPPING → Associating transaction hashes — WITH → Transaction details.

Constructor → Initializes variables INCLUDING →

- token → Contract address
- AxelarGateway → Contract address
- allowlist → Contract address
- Ondo → Approver address

The contract owner, Minting limits, Minting duration

Internal Functions

- _execute — overridden from → AxelarExecutable → Handles the execution of the contract when called by the Axelar Gateway. — INCLUDING → Version, source chain, sender address, amount, and nonce — If all conditions are met → Records the transaction and initiates token minting
- _attachThreshold — Associates → threshold with a transaction hash — BASED ON → The transaction amount and source chain.
- _approve — Approve → Transactions and checking for duplicate approvals ✓✓
- _checkThresholdMet — Checks → If the approval threshold for a transaction has been met

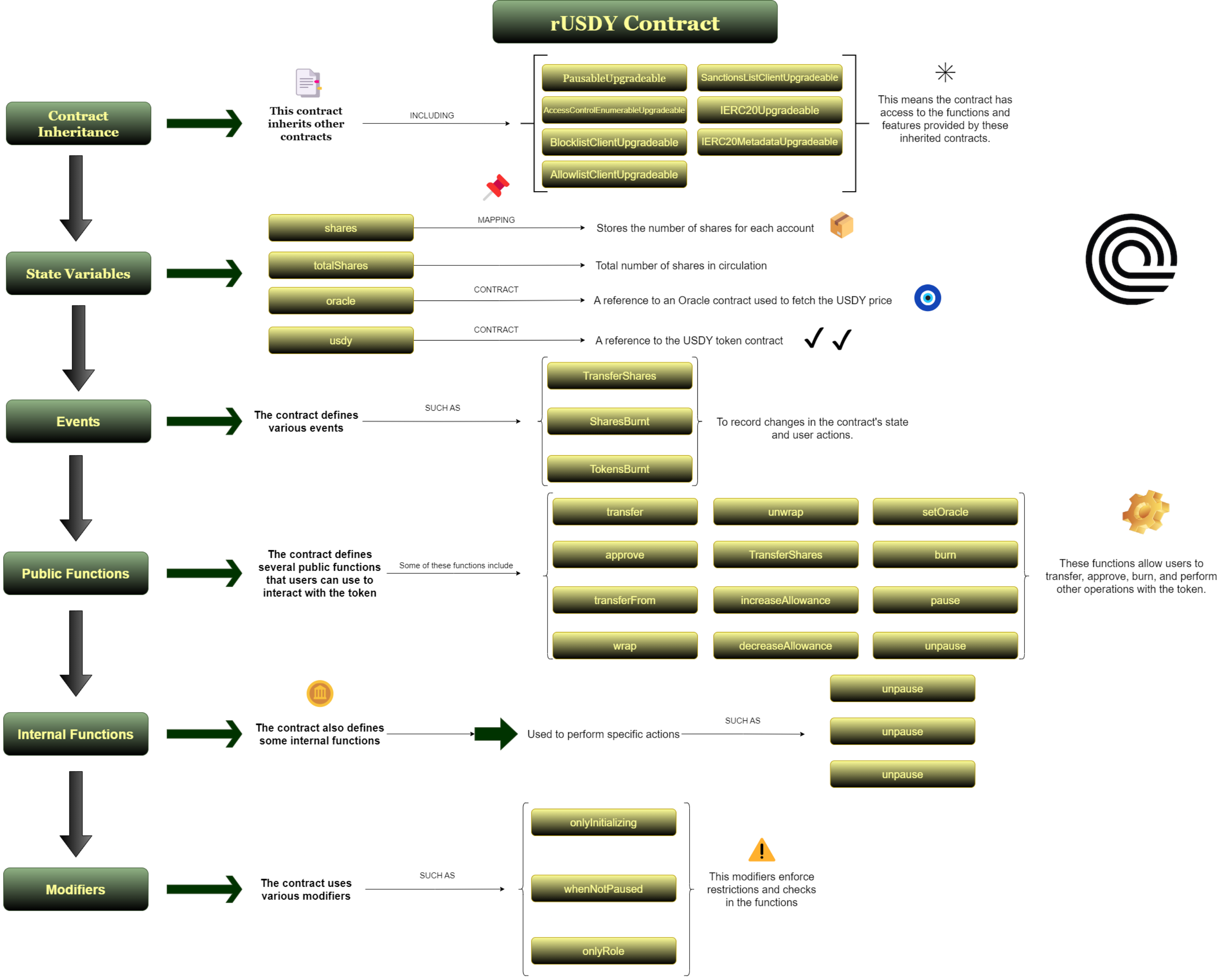
Protected and Admin Functions

- approve — ALLOWS → Approved addresses to approve transactions — AND → Potentially trigger token minting
- addApprover and removeApprover → Admin functions — FOR → Adding or removing approver addresses
- addChainSupport → Admin functions → Add support for source chains — AND → Their corresponding sender addresses
- setThresholds → Admin functions → Set approval thresholds for specific source chains
- setMintLimit and setMintLimitDuration → Admin functions → Set the minting limit and duration
- pause and unpause → Pause and unpause the contract
- rescueTokens → Admin functions → Rescue ERC20 tokens accidentally sent to the contract

Public Functions

- getNumApproved → Get the number of approvers for a transaction

rUSDY:



rUSDY Contract

Contract Inheritance

This contract inherits other contracts

INCLUDING

PausableUpgradeable

SanctionsListClientUpgradeable

AccessControlEnumerableUpgradeable

IERC20Upgradeable

BlocklistClientUpgradeable

IERC20MetadataUpgradeable

AllowlistClientUpgradeable

This means the contract has access to the functions and features provided by these inherited contracts.

State Variables

shares — MAPPING — Stores the number of shares for each account

totalShares — Total number of shares in circulation

oracle — CONTRACT — A reference to an Oracle contract used to fetch the USDY price

usdy — CONTRACT — A reference to the USDY token contract ✓ ✓

Events

The contract defines various events

SUCH AS

TransferShares

SharesBurnt

TokensBurnt

To record changes in the contract's state and user actions.

Public Functions

The contract defines several public functions that users can use to interact with the token

Some of these functions include

transfer	unwrap	setOracle
approve	TransferShares	burn
transferFrom	increaseAllowance	pause
wrap	decreaseAllowance	unpause

These functions allow users to transfer, approve, burn, and perform other operations with the token.

Internal Functions

The contract also defines some internal functions

Used to perform specific actions

SUCH AS

unpause

unpause

unpause

Modifiers

The contract uses various modifiers

SUCH AS

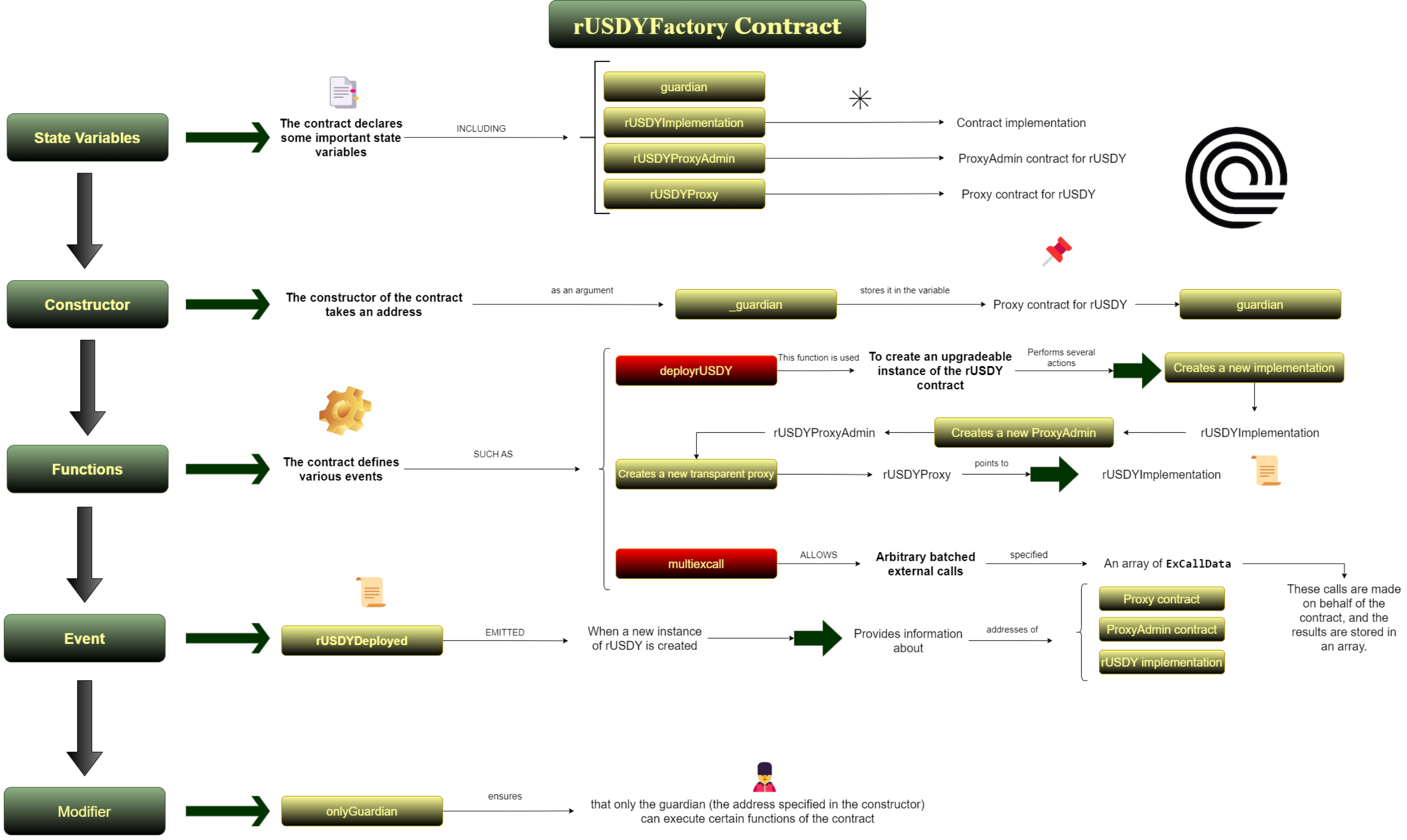
onlyInitializing

whenNotPaused

onlyRole

This modifiers enforce restrictions and checks in the functions

rUSDYFactory:



rUSDYFactory Contract

State Variables

The contract declares some important state variables

INCLUDING

- guardian
- rUSDYImplementation → Contract implementation
- rUSDYProxyAdmin → ProxyAdmin contract for rUSDY
- rUSDYProxy → Proxy contract for rUSDY

Constructor

The constructor of the contract takes an address → as an argument → _guardian → stores it in the variable → Proxy contract for rUSDY → guardian

Functions

The contract defines various events → SUCH AS

deployrUSDY → This function is used → **To create an upgradeable instance of the rUSDY contract** → Performs several actions → Creates a new implementation → rUSDYImplementation → Creates a new ProxyAdmin → rUSDYProxyAdmin → Creates a new transparent proxy → rUSDYProxy → points to → rUSDYImplementation

multiexcall → ALLOWS → **Arbitrary batched external calls** → specified → An array of `ExCallData` → These calls are made on behalf of the contract, and the results are stored in an array.

Event

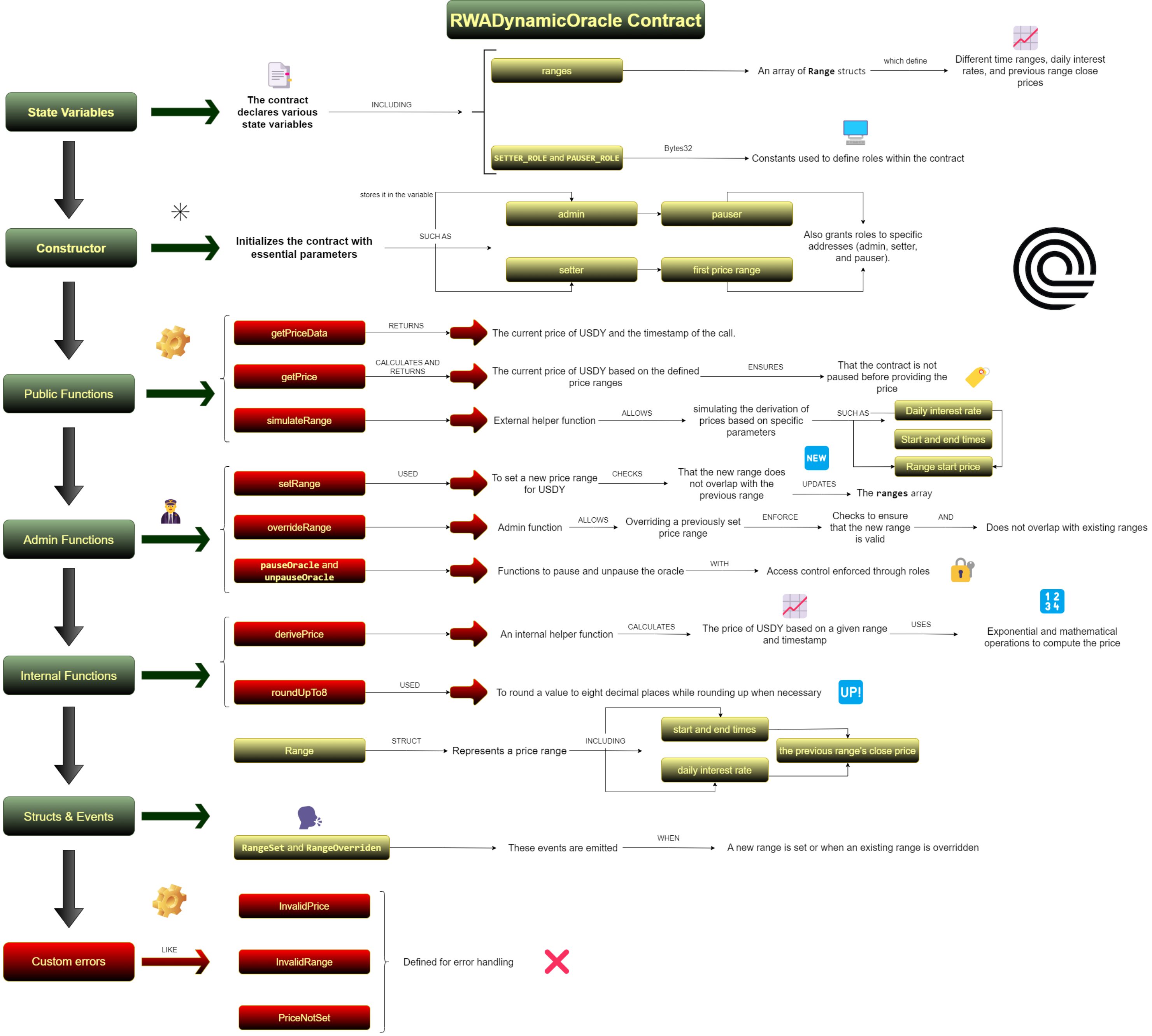
rUSDYDeployed → EMITTED → When a new instance of rUSDY is created → Provides information about → addresses of

- Proxy contract
- ProxyAdmin contract
- rUSDY implementation

Modifier

onlyGuardian → ensures → that only the guardian (the address specified in the constructor) can execute certain functions of the contract

RWADynamicOracle:


RWADynamicOracle Contract
State Variables
The contract declares various state variables
INCLUDING
ranges
An array of Range structs
which define
Different time ranges, daily interest rates, and previous range close prices
SETTER_ROLE and PAUSER_ROLE
Bytes32
Constants used to define roles within the contract
Constructor
Initializes the contract with essential parameters
SUCH AS
stores it in the variable
admin
pauser
setter
first price range
Also grants roles to specific addresses (admin, setter, and pauser).
Public Functions
getPriceData
RETURNS
The current price of USDY and the timestamp of the call.
getPrice
CALCULATES AND RETURNS
The current price of USDY based on the defined price ranges
ENSURES
That the contract is not paused before providing the price
simulateRange
External helper function
ALLOWS
simulating the derivation of prices based on specific parameters
SUCH AS
Daily interest rate
Start and end times
Range start price
Admin Functions
setRange
USED
To set a new price range for USDY
CHECKS
That the new range does not overlap with the previous range
UPDATES
The ranges array
NEW
overrideRange
Admin function
ALLOWS
Overriding a previously set price range
ENFORCE
Checks to ensure that the new range is valid
AND
Does not overlap with existing ranges
pauseOracle and unpauseOracle
Functions to pause and unpause the oracle
WITH
Access control enforced through roles
Internal Functions
derivePrice
An internal helper function
CALCULATES
The price of USDY based on a given range and timestamp
USES
Exponential and mathematical operations to compute the price
1 2 3 4
roundUpTo8
USED
To round a value to eight decimal places while rounding up when necessary
UP!
Structs & Events
Range
STRUCT
Represents a price range
INCLUDING
start and end times
the previous range's close price
daily interest rate
RangeSet and RangeOverriden
These events are emitted
WHEN
A new range is set or when an existing range is overridden
Custom errors
LIKE
InvalidPrice
InvalidRange
PriceNotSet
Defined for error handling

Architecture Feedback

The architecture of the Ondo Finance project seems solid in general, but there is always room for improvements and adjustments. Here are some areas that could be improved:

- **Upgradeability:** Since the contracts `rUSDY` and `rUSDYFactory` are upgradeable, having a clear and secure process for performing upgrades is important. It may also be useful to implement time limits for upgrades or allow a majority of users to approve upgrades.
- **Governance:** Consider incorporating a governance system to allow users to vote on key decisions, such as changes in interest rates or adjustments to system parameters.

- **Oracles and Data Consistency:** The consistency and accuracy of oracle data are crucial. You can explore the possibility of using multiple oracles or implementing consensus mechanisms to ensure data reliability.
- **Testing and Simulations:** Even though the project implements several tests, upon reviewing the codebase, there are several crucial functions that don't, such as `rUSDY:transferFrom`. Conduct thorough testing of all contracts and functions and simulations to understand how they will behave under various market conditions. This can help anticipate potential issues.

Documents

The documentation of the Ondo Finance project is quite comprehensive and detailed, providing a solid overview of how Ondo is structured and how its various aspects function. However, we have noticed that there is room for additional details, such as diagrams, to gain a deeper understanding of how different contracts interact and the functions they implement. With considerable enthusiasm, we have dedicated some days to creating diagrams for each of the contracts.

We are confident that these diagrams will bring significant value to the protocol, as they can be seamlessly integrated into the existing documentation, enriching it and providing a more comprehensive and detailed understanding for users, developers and auditors.

Systemic & Centralization Risks

Here's an analysis of potential systemic and centralization risks in the provided contracts:

- **Loss of Funds Risk:** If the contract stores or manages digital assets (such as tokens), there is an inherent risk of funds being lost due to errors in the contract's logic or malicious attacks.
- The protocol uses a `proxy`, and according to the Documentation, the type of `proxy` is:
    - rUSDYFactory.sol

```
 * 3) TransparentUpgradeableProxy - OZ, proxy contract. Admin is set to
```

```
     *                                               `_logic' is set to `address(
```

If the logic controlling the `proxy` is not implemented correctly, there could be vulnerabilities that allow an attacker to modify the underlying contract, or the `proxy` itself, in an unintended way.

- **Price Manipulation Risk:** Since these contracts are designed to calculate the price of `USDY` based on interest rates and price ranges, there is a risk that parameters may be manipulated or that input data may be compromised, potentially leading to incorrect prices.
- **Centralized Administration Risk:** If the contract has administrator roles that can modify settings or pause the contract, there is a risk that these capabilities may be used improperly or become targets for attacks.
- **Third-Party Dependency Risk:** Contracts rely on external data sources, such as @openzeppelin/contracts, and there is a risk that if any issues are found with these dependencies in your contracts, the Ondo finance protocol could also be affected.

    - We observed that old versions of OpenZeppelin are used in the project, and these should be updated to the latest version:

```
29: "@openzeppelin/contracts": "^4.8.3",
30: "@openzeppelin/hardhat-upgrades": "1.22.1",
```

The latest version is `4.9.3` (as of July 28, 2023), while the project uses version `4.8.3`.

That's why we strongly recommend that once the issues in the codebase identified by C4 Wardens are known, the Ondo Finance team takes action to implement the mitigations and make their protocol a robust financial ecosystem for all participants.

Monitoring Recommendations

While audits help in identifying code-level issues in the current implementation and potentially the code deployed in production, the Ondo Finance team is encouraged to consider incorporating monitoring activities in the production environment. Ongoing monitoring of deployed contracts helps identify potential threats and issues affecting production environments. With the goal of providing a complete security assessment, the monitoring recommendations section raises several actions addressing trust assumptions and out-of-scope components that can benefit from on-chain monitoring.

Financial Activity

Consider monitoring the token transfers over the bridge to identify:

- Transfers during normal operations to establish a baseline of healthy properties. Any large deviation, such as an unexpectedly large withdrawal, may indicate unusual behavior of the contracts or an ongoing attack.
- Transactions that revert.

These may indicate a user interface bug, an ongoing attack or other unexpected edge cases.

Time Spent

A total of 3 days (15 hours) were dedicated to completing this audit, distributed as follows:

1. 1st Day: Devoted to comprehending the protocol's functionalities and implementation.
2. 2nd Day: Initiated the analysis process, leveraging the documentation offered by Ondo Finance.
3. 3rd Day: Focused on conducting a thorough analysis, incorporating meticulously crafted diagrams derived from the contracts and information provided by the protocol.

tom2o17 (Ondo) acknowledged

Disclosures

C4 is an open organization governed by participants in the community.

C4 Audits incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Audit submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top

An open organization | Twitter | Discord | GitHub | Medium | Newsletter | Media kit | Careers |
code4rena.eth