CERTIK

RWA Ecosystem

Security Assessment

CertiK Assessed on Jan 22nd, 2025

CertiK Assessed on Jan 22nd, 2025

RWA Ecosystem

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

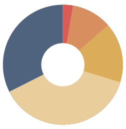
TYPES	ECOSYSTEM	METHODS
DeFi	Binance Smart Chain (BSC)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 01/22/2025	N/A

Highlighted Centralization Risks

- ⚠ Privileged role can mint tokens
- ⚠ Fees are bounded by 100%

Vulnerability Summary



Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined
37	1	0	0	36	0

■	1	Critical	1 Acknowledged	Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
■	4	Major	4 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
■	6	Medium	6 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
■	14	Minor	14 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
■	12	Informational	1 Resolved, 11 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | RWA ECOSYSTEM

Summary

Findings

AUDIT SCOPE | RWA ECOSYSTEM

8 files audited ● 8 files with Acknowledged findings

ID	Repo	File	SHA256 Checksum
● BVC	CertiKProject/certik-audit-projects	projects/audit-fed1/BondV2.sol	27bda271000131b1f2fe3cd5b4e5c939972ba5827c36d43dc4d2677b3aff90c1
● ERC	CertiKProject/certik-audit-projects	projects/audit-fed1/ERC20.sol	b289b210f8e9ce9c7eb237b4ac0c27bf12d215e4934eff5f4a6d388d21b30256
● SVC	CertiKProject/certik-audit-projects	projects/audit-fed1/StakingV2.sol	4c8cbc12ee9e725ecd824e4d74f8666715dbb8bac518e133adc6bd6a72caee9a
● SDC	CertiKProject/certik-audit-projects	projects/audit-fed2/StakingDistributor.sol	48e078371a3a43ac6fbcf372dfb94a007a51977af7d7390cc7e7ffb79256249b
● SWC	CertiKProject/certik-audit-projects	projects/audit-fed2/StakingWarmup.sol	db8a28b7127a7358c975d8b14590f891b02f5d37392582846b4450bb8b6e3b3c
● SBC	CertiKProject/certik-audit-projects	projects/audit-fed2/StandardBondingCalculator.sol	3064a9bda43d33c31d5de66be41a7328f55f0525bac38d119dda12743e6af51f
● TCK	CertiKProject/certik-audit-projects	projects/audit-fed2/Treasury.sol	44f40d02c4540402173ed1ee7895f75b671bd82780ce7dcd17fa9b7035c8cbda
● ERK	CertiKProject/certik-audit-projects	projects/audit-fed2/sERC20.sol	c7237a5a1cffda199fdba24914fe0eafd0c4998ce0697644999464bad7d128d0

APPROACH & METHODS | RWA ECOSYSTEM

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

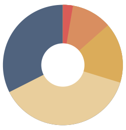
The auditing process pays special attention to the following considerations:

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

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

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

FINDINGS | RWA ECOSYSTEM



37	1	4	6	14	12
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for RWA Ecosystem. Through this audit, we have uncovered 37 issues ranging from different severity levels. Utilizing the techniques of Static Analysis & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
CKP-20	**Unlimited Mint Of `ERC20TokenX` Allows Privileged Role To Drain All User Deposits From The Treasury**	**Centralization**	**Critical**	● **Acknowledged**
CKP-02	**Centralization Risks**	**Centralization**	**Major**	● **Acknowledged**
ERC-02	**No Cap On Fees**	**Centralization**	**Major**	● **Acknowledged**
TCK-01	**Unrestricted Reward Minter Privileges And Potential Minter Role Misconfiguration**	**Centralization**	**Major**	● **Acknowledged**
TCK-02	Defects Of IncurDebt	Logical Issue	Major	● Acknowledged
BVC-02	Logic Issue In Function `deposit()`	Logical Issue	Medium	● Acknowledged
BVC-03	Inconsistent Implementation Of `terms.minimumPrice` In Function `bondPrice()` And `_bondPrice()`	Logical Issue	Medium	● Acknowledged
CKP-03	Ownership Can Be Regained After Renouncement	Logical Issue	Medium	● Acknowledged
CKP-04	Anyone Can Call `redeem()` And `claim()` For Any Arbitrary `_recipient` Address	Access Control	Medium	● Acknowledged

ID	Title	Category	Severity	Status
CKP-05	Lack Of A Permissionless Mechanism To Redeem Principal	Logical Issue, Design Issue	Medium	Acknowledged
SBC-01	Potential Flashloan Attack	Design Issue	Medium	Acknowledged
BVC-04	Inconsistent And Missing Validations In Bond Term Management Functions	Volatile Code	Minor	Acknowledged
BVC-05	Inconsistent Scaling Factors In `getNewBCV()` And `getNewPrice()` Calculations	Inconsistency	Minor	Acknowledged
BVC-06	Users Can Only Stake When They Redeem	Design Issue	Minor	Acknowledged
CKP-06	Hidden Role In The Contract May Raise Centralization Concerns	Coding Issue	Minor	Acknowledged
CKP-07	Missing Zero Address Validation	Volatile Code	Minor	Acknowledged
CKP-08	Incompatibility With Deflationary Tokens (Non-Standard ERC20 Token)	Volatile Code	Minor	Acknowledged
CKP-09	Susceptible To Signature Malleability	Volatile Code	Minor	Acknowledged
CKP-10	Possibility Of Replay Attack In `Permit`	Volatile Code	Minor	Acknowledged
CKP-11	Third-Party Dependencies	Volatile Code	Minor	Acknowledged
ERC-03	Function `_burnFrom()` Should Be `internal`	Logical Issue	Minor	Acknowledged
ERK-02	Divide By Zero	Incorrect Calculation	Minor	Acknowledged
SWC-01	Return Value Not Handled	Volatile Code	Minor	Acknowledged

ID	Title	Category	Severity	Status
TCK-03	Liquidity Token Cannot Be Withdrawn	Logical Issue	Minor	● Acknowledged
TCK-04	Missing Validation For `sOHMQueue` In `toggle` Function	Volatile Code	Minor	● Acknowledged
BVC-07	Unused Variables	Coding Issue	Informational	● Acknowledged
CKP-12	Event Not Indexed	Design Issue	Informational	● Acknowledged
CKP-13	Missing Input Validation	Logical Issue	Informational	● Acknowledged
CKP-14	Spenders With Infinite Allowance Handled Incorrectly	Coding Style	Informational	● Acknowledged
CKP-15	Wrong Address In `_mint()` Function	Logical Issue	Informational	● Acknowledged
CKP-16	Contracts With Todos	Coding Issue	Informational	● Acknowledged
CKP-17	Using Library For All Is Depreciated	Coding Style	Informational	● Acknowledged
CKP-18	Missing Error Messages	Coding Style	Informational	● Acknowledged
CKP-19	Missing Emit Events	Coding Style	Informational	● Acknowledged
ERC-01	Discussion On Design	Design Issue	Informational	● Resolved
ERK-03	Incorrect Comment	Coding Style	Informational	● Acknowledged
SVC-01	Discussion On LockBonus	Design Issue, Logical Issue	Informational	● Acknowledged

CKP-20 UNLIMITED MINT OF `ERC20TokenX` ALLOWS PRIVILEGED ROLE TO DRAIN ALL USER DEPOSITS FROM THE TREASURY

Category	Severity	Location	Status
Centralization	● **Critical**	**projects/audit-fed1/ERC20.sol: 394~397, 401~403; projects/audit-fed2/Treasury.sol: 423~436, 653~764**	● **Acknowledged**

Description

Important Note: Certain identification procedures were attempted to be applied to the project team in order to better understand the centralization situation and potential risks of the project. We strongly advise end users to conduct further research and exercise due diligence before engaging with the project given the centralization related risks. It is crucial for end users to independently verify and assess all available information

The `ERC20TokenX` contract inherits the `ERC20Token` contract, which contains a privileged mint function that allows the address with `MINT` authority to mint unlimited amount of token. The `DEFAULT_ADMIN_ROLE` has the ability to give any address the `MINT` role.

In the `Treasury` contract where all user deposits of the reserve / principle token is stored, the `withdraw()` function allows any address that is a reserve spender to burn `ERC20TokenX` token and withdraw the corresponding amount of reserve / principle token up to the `totalReserves` amount which reflects total user deposits. The `owner` of the `Treasury` contract has the ability to set any address as the reserve spender via the `queue()` and `toggle()` functions.

Combining the above, the privileged `Owner` / `DEFAULT_ADMIN_ROLE` address has the ability to mint a large amount of `ERC20TokenX` token and drain all user deposits from the `Treasury` contract by burning the `ERC20TokenX` tokens and withdrawing the reserve / principle (USDT) tokens.

Recommendation

The `Treasury` contract should be the only address that can mint the `ERC20TokenX` (OHM) token. The `DEFAULT_ADMIN_ROLE` of the `ERC20TokenX` contract should be revoked to prevent it from adding any address to have the `MINT` role after giving `Treasury` contract the ability to mint.

Alleviation

[RWA Team, 01/17/2025]: Issue acknowledged, DAO wallet will renounce `DEFAULT_ADMIN_ROLE` when the project stabilizes.

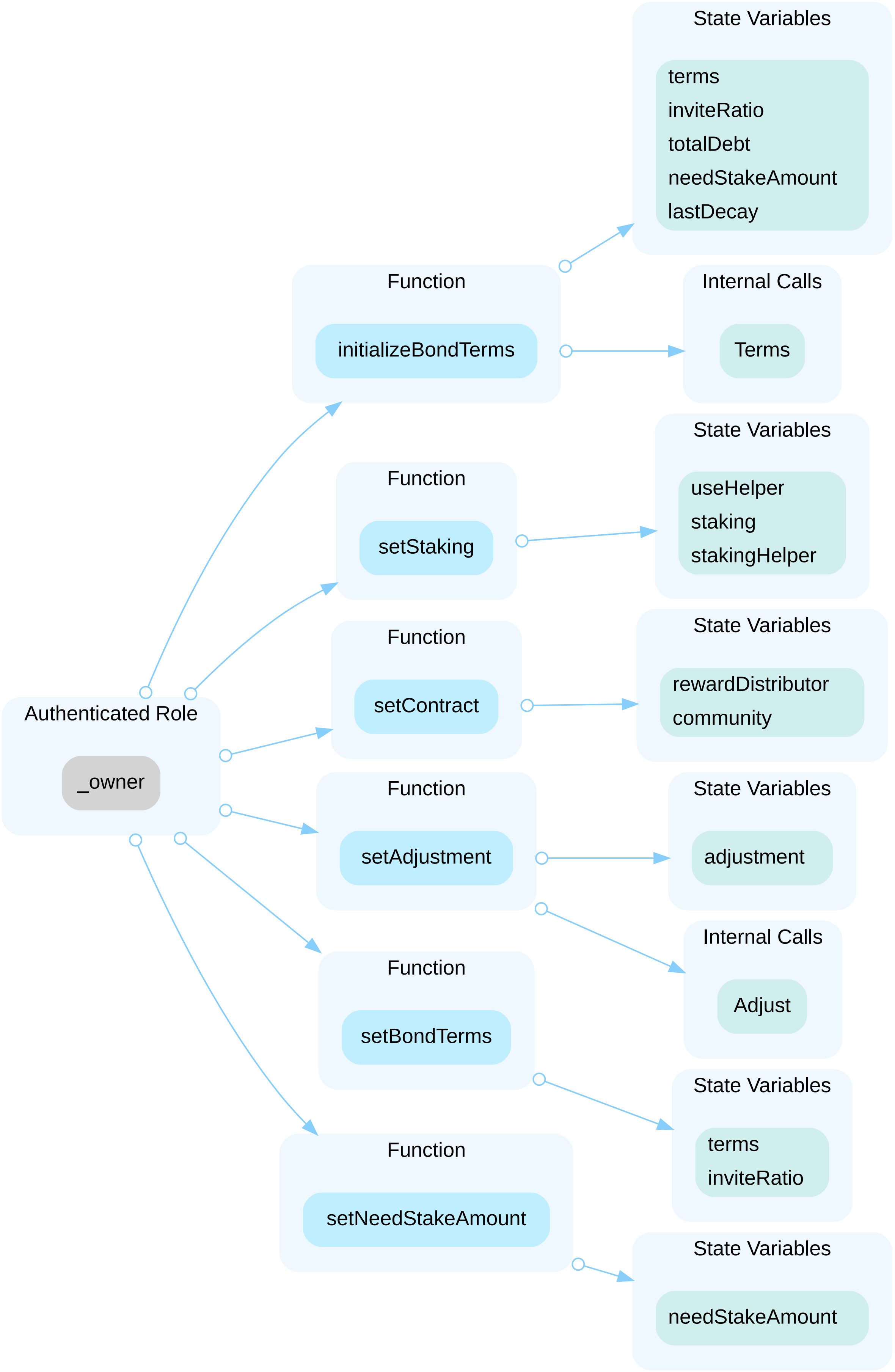
CKP-02 | CENTRALIZATION RISKS

Category	Severity	Location	Status
Centralization	● **Major**	**projects/audit-fed1/BondV2.sol: 45, 50, 58, 952, 994, 1024, 1049, 1070, 1084; projects/audit-fed1/ERC20.sol: 401, 470, 474; projects/audit-fed1/StakingV2.sol: 500, 505, 511, 756, 766, 778, 799; projects/audit-fed2/StakingDistributor.sol: 463, 476, 489; projects/audit-fed2/Treasury.sol: 393, 443, 469, 489, 505, 526, 547, 599, 653; projects/audit-fed2/sERC20.sol: 1042, 1057**	● **Acknowledged**

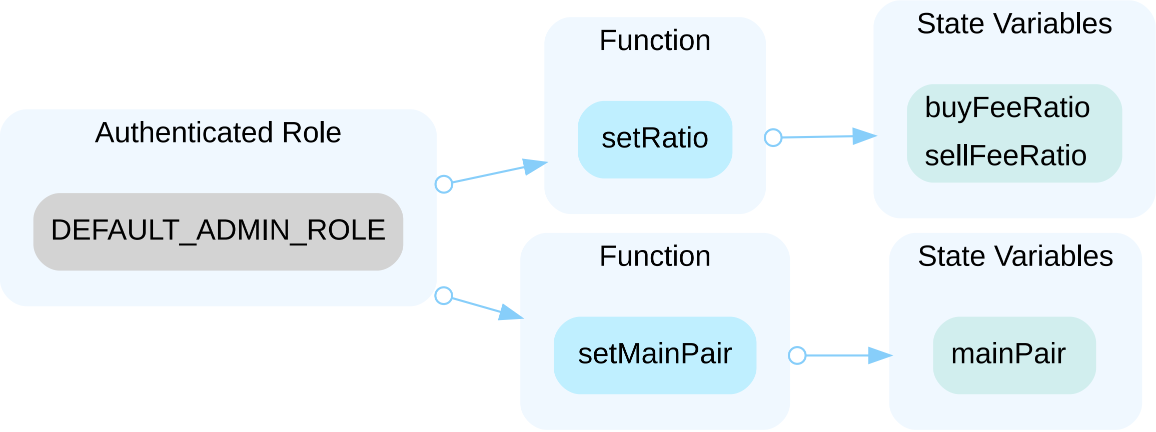
Description

Important Note: Certain identification procedures were attempted to be applied to the project team in order to better understand the centralization situation and potential risks of the project. We strongly advise end users to conduct further research and exercise due diligence before engaging with the project given the centralization related risks. It is crucial for end users to independently verify and assess all available information

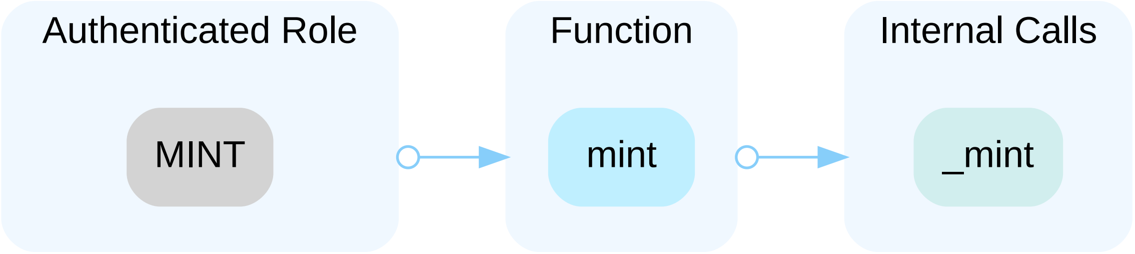
In the contract `BondDepositoryDai` , the role `_owner` / `policy` has authority over the functions shown in the diagram below. Any compromise to the `_owner` / `policy` account may allow the hacker to take advantage of this authority and initialize bond terms with given parameters, set staking address with an optional helper parameter, set contract address based on contract ID, set adjustment parameters, set bond terms based on given parameters, and set the need stake amount.


State Variables
terms
inviteRatio
totalDebt
needStakeAmount
lastDecay
Function
initializeBondTerms
Internal Calls
Terms
State Variables
useHelper
staking
stakingHelper
Function
setStaking
Function
setContract
State Variables
rewardDistributor
community
Authenticated Role
_owner
Function
setAdjustment
State Variables
adjustment
Internal Calls
Adjust
Function
setBondTerms
State Variables
terms
inviteRatio
Function
setNeedStakeAmount
State Variables
needStakeAmount

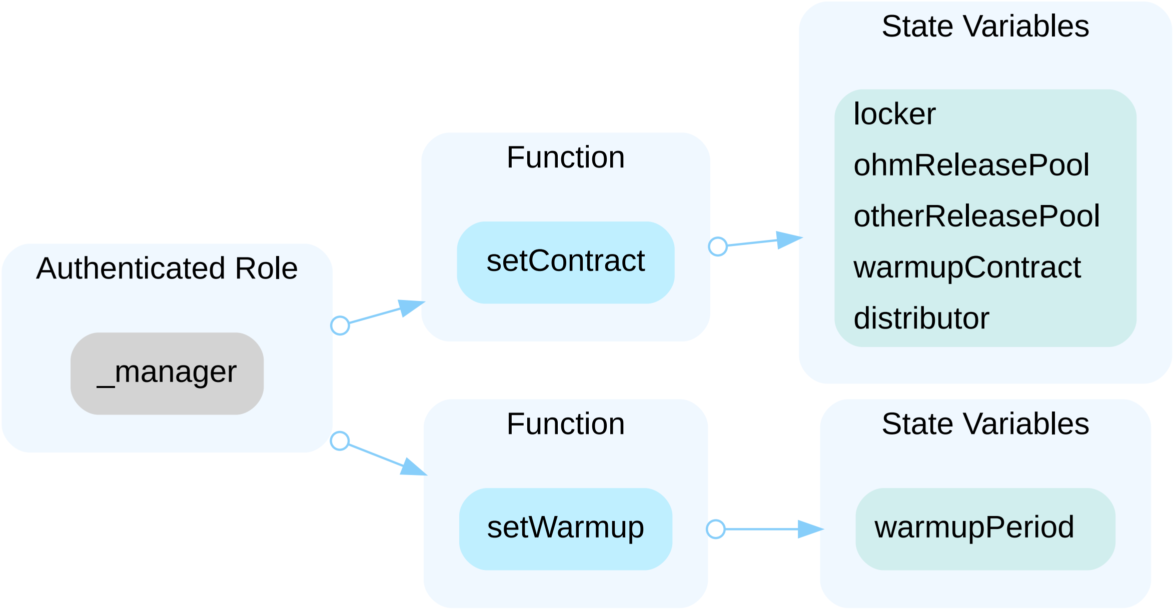
In the contract `ERC20TokenX` , the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and set the fee ratio, set and the main pair. The `DEFAULT_ADMIN_ROLE` also has the authority to grant other roles including the `MINT` role.



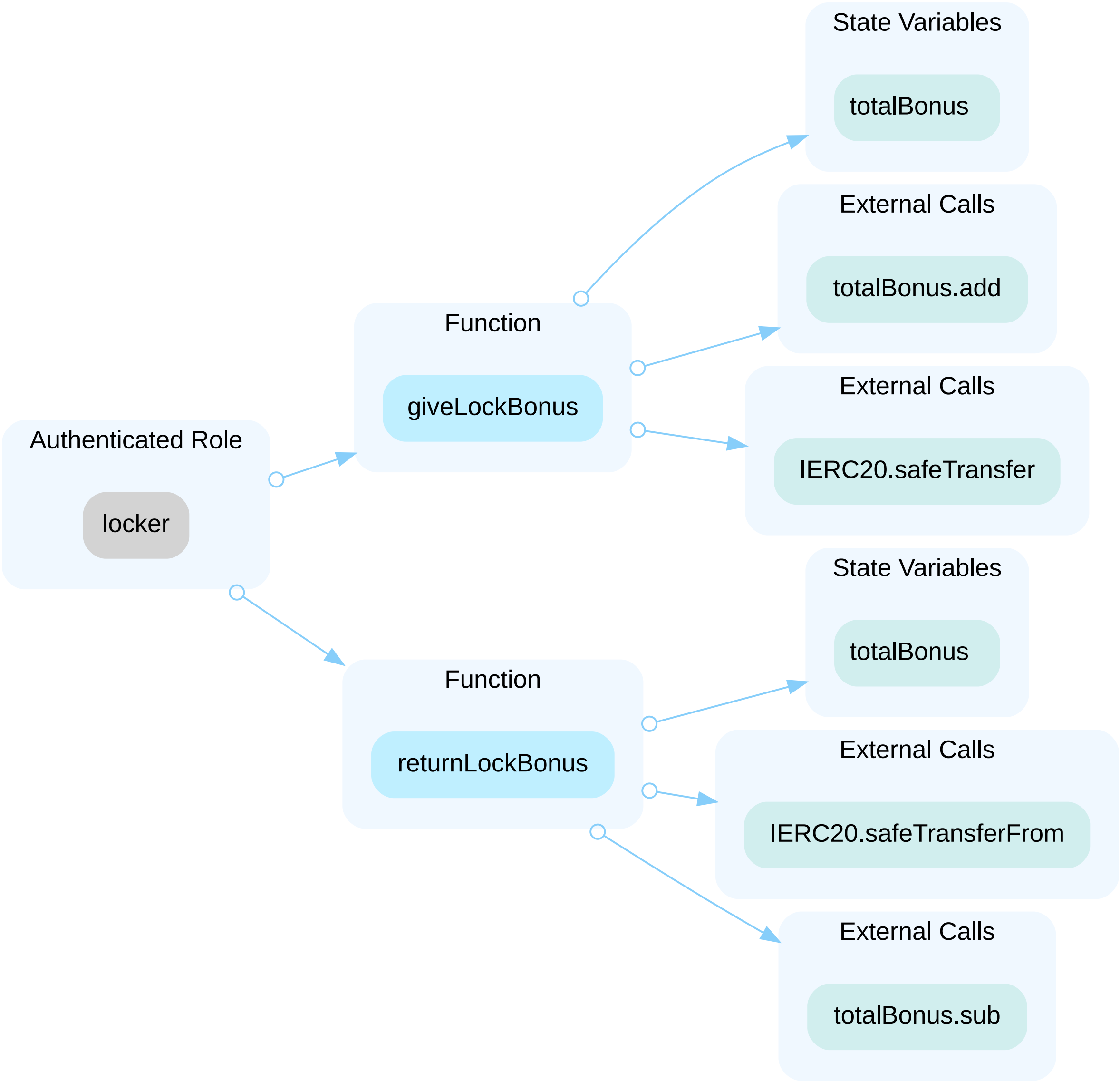
In the contract `ERC20Token` , the role `MINT` / `Vault` has authority over the functions shown in the diagram below. Any compromise to the `MINT` / `Vault` account may allow the hacker to take advantage of this authority and mint tokens to a specified account.



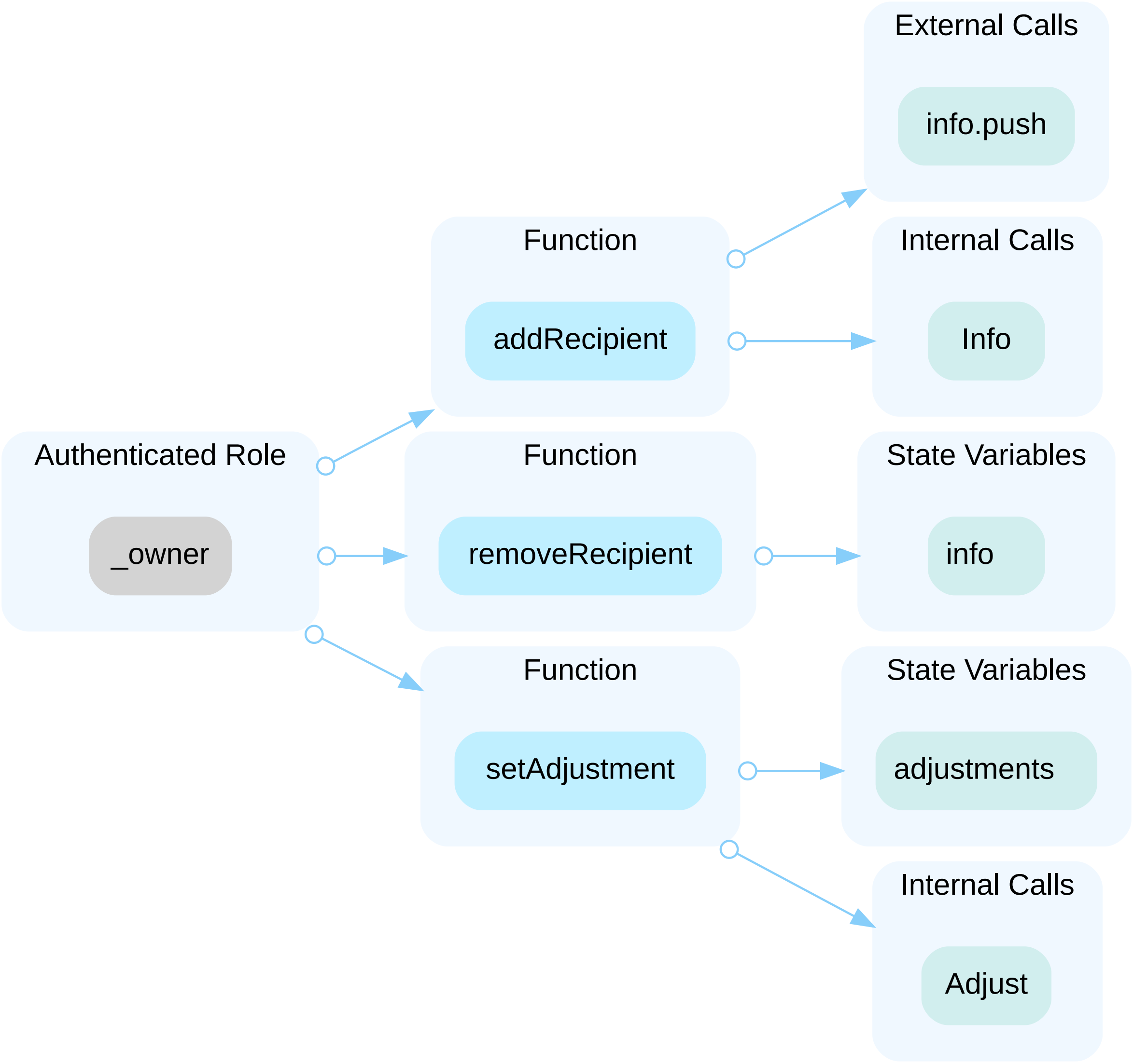
In the contract `StakingV2` , the role `_owner` / `_manager` has authority over the functions shown in the diagram below. Any compromise to the `_owner` / `_manager` account may allow the hacker to take advantage of this authority and set important contract addresses, and set the warmup period.



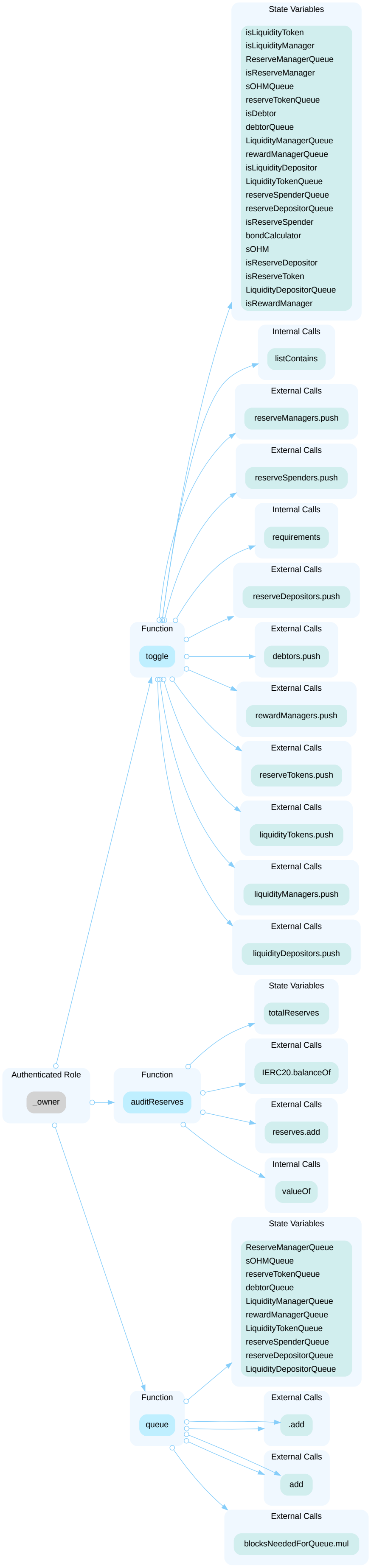
In the contract `StakingV2` , the role `locker` has authority over the functions shown in the diagram below. Any compromise to the `locker` account may allow the hacker to take advantage of this authority and potentially drains all the sOHM token from the staking contract.



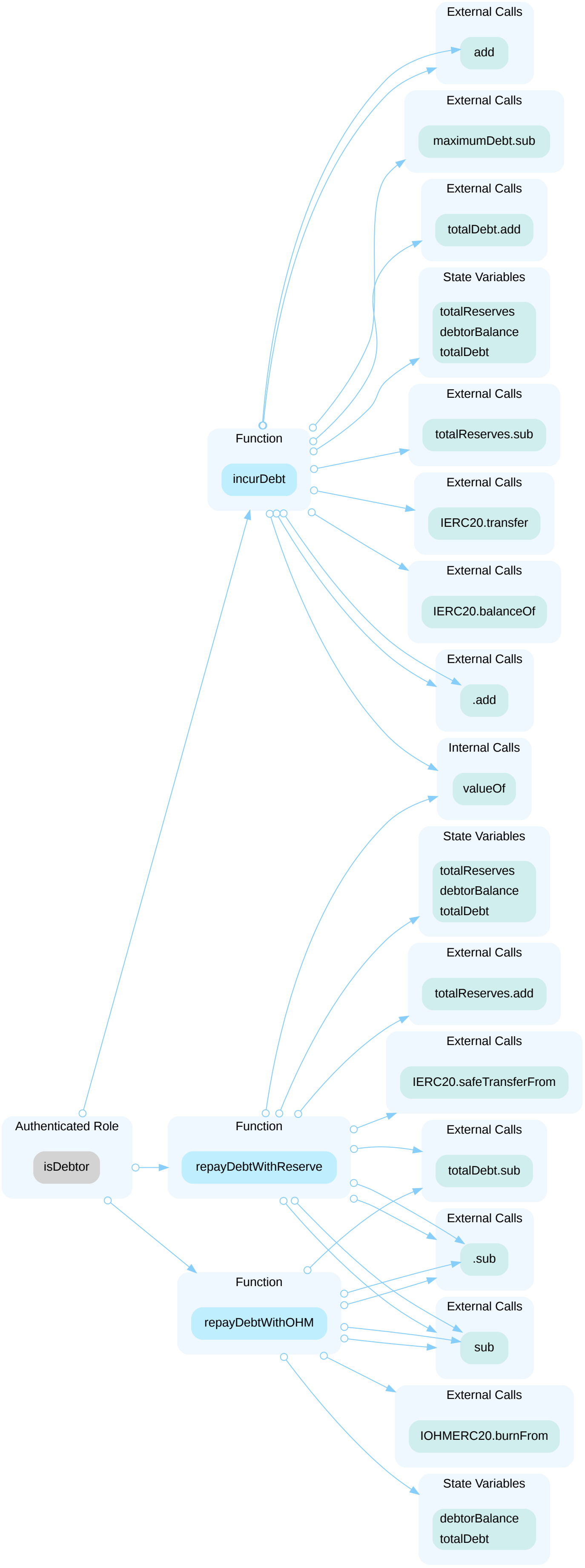
In the contract `Distributor` , the role `_owner` / `policy` has authority over the functions shown in the diagram below. Any compromise to the `_owner` / `policy` account may allow the hacker to take advantage of this authority and add a recipient with a reward rate, remove a recipient from the info list, or set adjustments with given parameters.



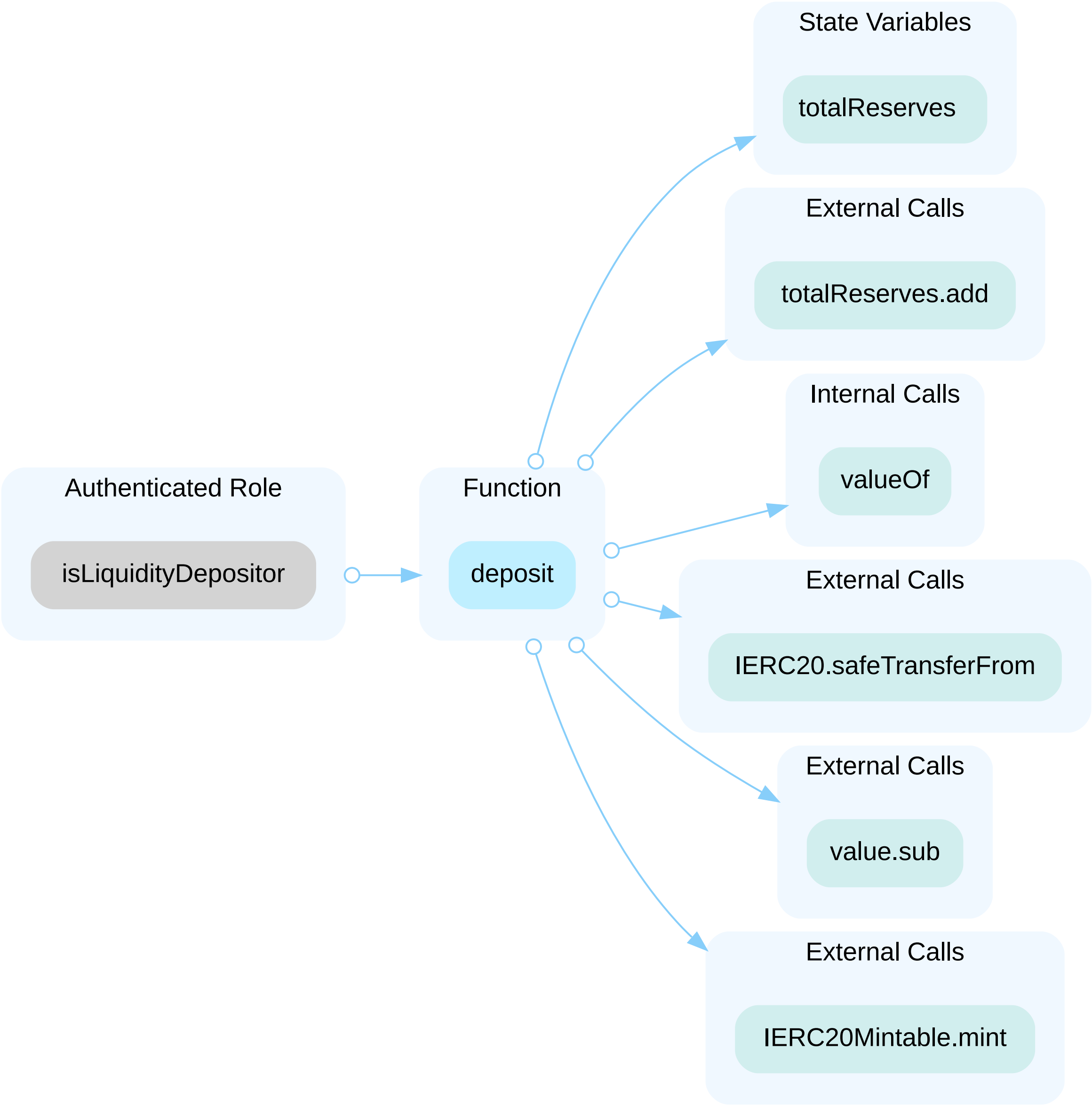
In the contract `Treasury`, the role `_owner` / `_manager` has authority over the functions shown in the diagram below. Any compromise to the `_owner` / `_manager` account may allow the hacker to take advantage of this authority and toggle specific management status for an address, audit and update total reserves, and queue managing address for future action.


State Variables
isLiquidityToken
isLiquidityManager
ReserveManagerQueue
isReserveManager
sOHMQueue
reserveTokenQueue
isDebtor
debtorQueue
LiquidityManagerQueue
rewardManagerQueue
isLiquidityDepositor
LiquidityTokenQueue
reserveSpenderQueue
reserveDepositorQueue
isReserveSpender
bondCalculator
sOHM
isReserveDepositor
isReserveToken
LiquidityDepositorQueue
isRewardManager
Internal Calls
listContains
External Calls
reserveManagers.push
External Calls
reserveSpenders.push
Internal Calls
requirements
External Calls
reserveDepositors.push
Function
toggle
External Calls
debtors.push
External Calls
rewardManagers.push
External Calls
reserveTokens.push
External Calls
liquidityTokens.push
External Calls
liquidityManagers.push
External Calls
liquidityDepositors.push
State Variables
totalReserves
External Calls
IERC20.balanceOf
Authenticated Role
_owner
Function
auditReserves
External Calls
reserves.add
Internal Calls
valueOf
State Variables
ReserveManagerQueue
sOHMQueue
reserveTokenQueue
debtorQueue
LiquidityManagerQueue
rewardManagerQueue
LiquidityTokenQueue
reserveSpenderQueue
reserveDepositorQueue
LiquidityDepositorQueue
Function
queue
External Calls
.add
External Calls
add
External Calls
blocksNeededForQueue.mul

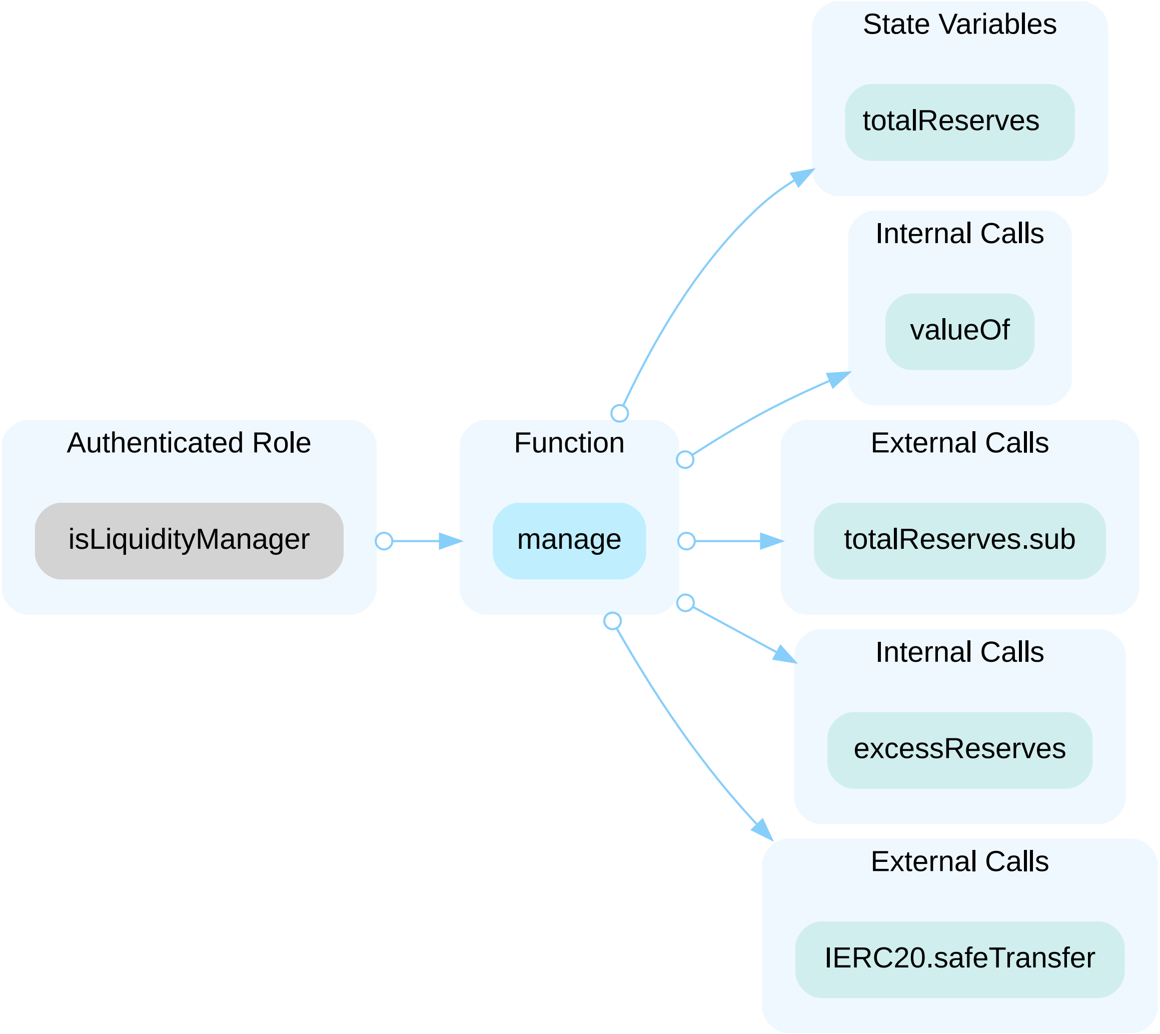
In the contract `Treasury`, the role `isDebtor` has authority over the functions shown in the diagram below. Any compromise to the `isDebtor` account may allow the hacker to take advantage of this authority and incur debt with tokens, update balances, and repay debt using reserve tokens or with OHM tokens.


External Calls
add
External Calls
maximumDebt.sub
External Calls
totalDebt.add
State Variables
totalReserves
debtorBalance
totalDebt
External Calls
totalReserves.sub
Function
incurDebt
External Calls
IERC20.transfer
External Calls
IERC20.balanceOf
External Calls
.add
Internal Calls
valueOf
State Variables
totalReserves
debtorBalance
totalDebt
External Calls
totalReserves.add
External Calls
IERC20.safeTransferFrom
Authenticated Role
isDebtor
Function
repayDebtWithReserve
External Calls
totalDebt.sub
External Calls
.sub
Function
repayDebtWithOHM
External Calls
sub
External Calls
IOHMERC20.burnFrom
State Variables
debtorBalance
totalDebt

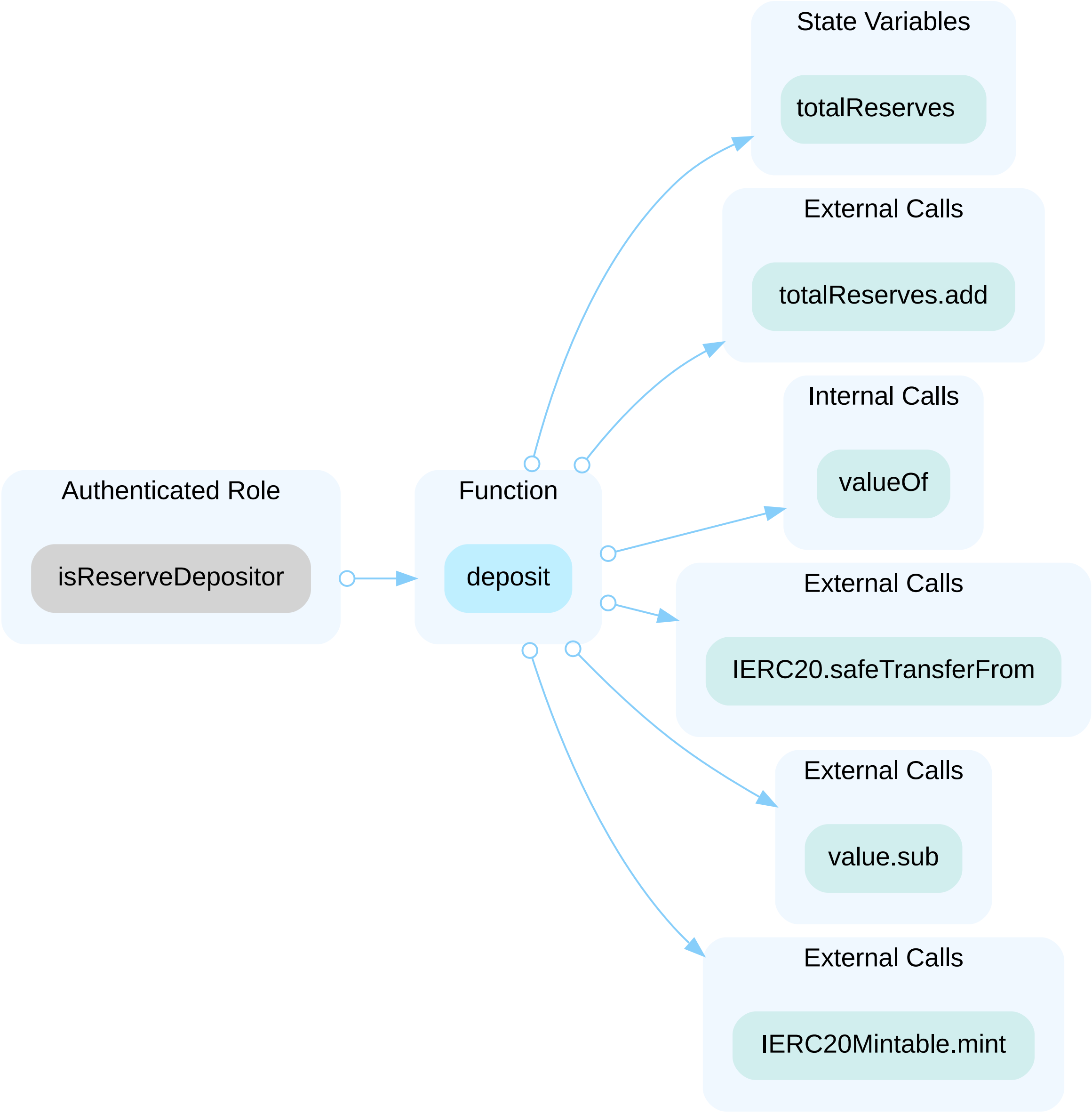
In the contract `Treasury`, the role `isLiquidityDepositor` has authority over the functions shown in the diagram below. Any compromise to the `isLiquidityDepositor` account may allow the hacker to take advantage of this authority and process a deposit of tokens.



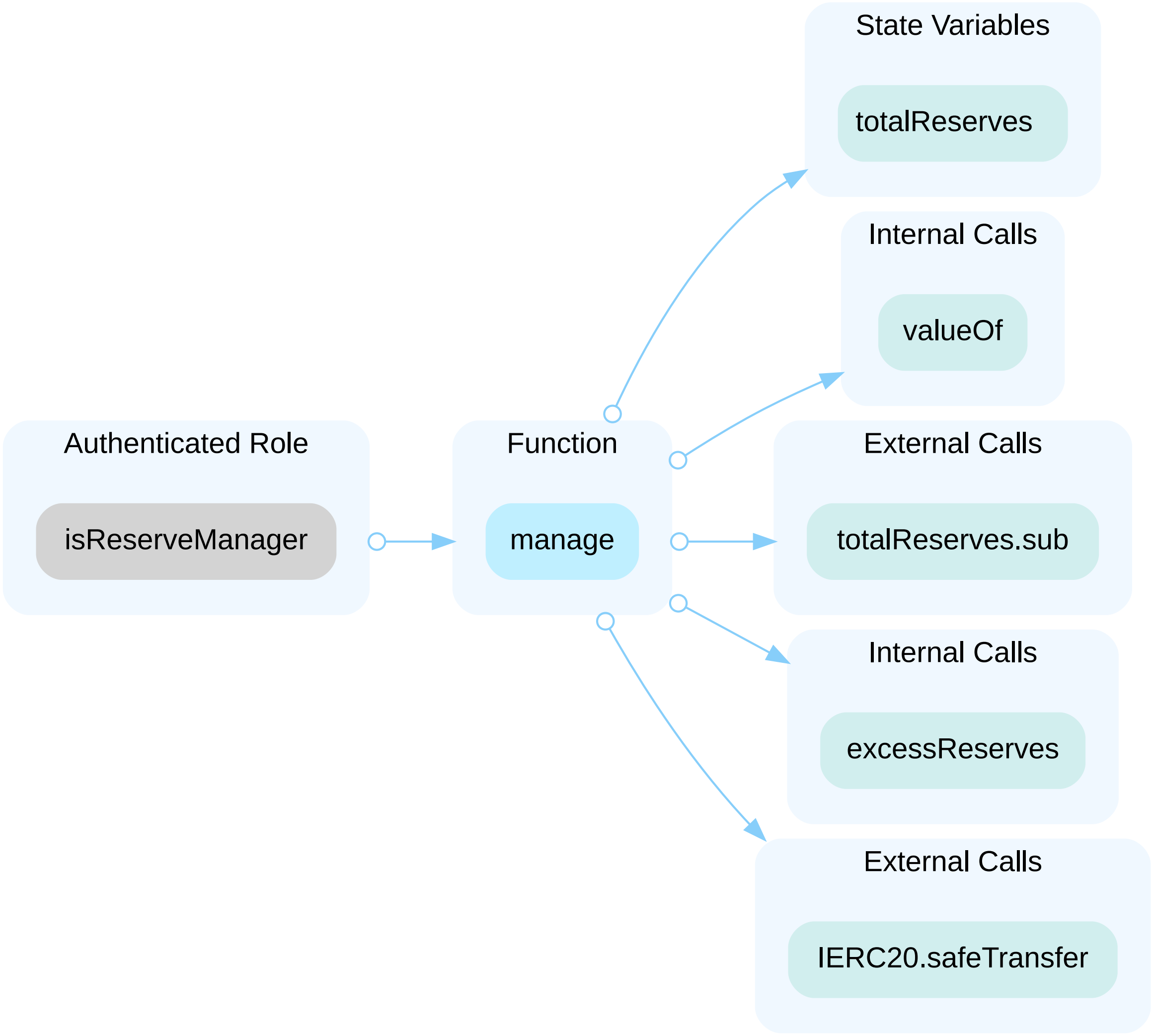
In the contract `Treasury`, the role `isLiquidityManager` has authority over the functions shown in the diagram below. Any compromise to the `isLiquidityManager` account may allow the hacker to take advantage of this authority and withdraw assets.



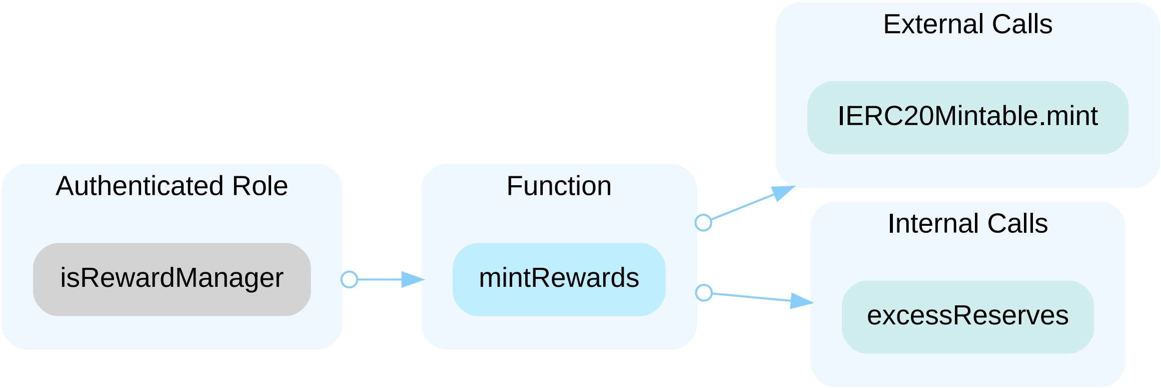
In the contract `Treasury`, the role `isReserveDepositor` has authority over the functions shown in the diagram below. Any compromise to the `isReserveDepositor` account may allow the hacker to take advantage of this authority and process deposits as well as mint tokens.



In the contract `Treasury`, the role `isReserveManager` has authority over the functions shown in the diagram below. Any compromise to the `isReserveManager` account may allow the hacker to take advantage of this authority and manage withdraw assets.

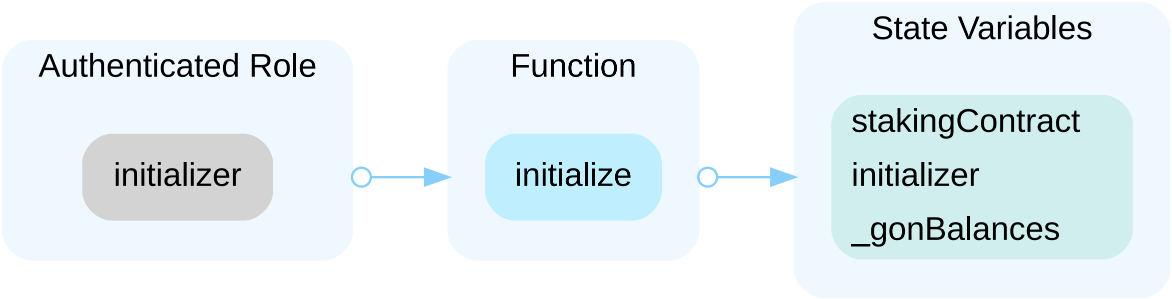


In the contract `Treasury`, the role `isRewardManager` has authority over the functions shown in the diagram below. Any compromise to the `isRewardManager` account may allow the hacker to take advantage of this authority and mint rewards to recipients.



In the contract `sERC20`, the role `_owner` / `_manager` has authority over the functions shown in the diagram below. Any compromise to the `_owner` / `_manager` account may allow the hacker to take advantage of this authority and set the index if the current index is zero. ![](https://accelerator-tasks-prod.s3.amazonaws.com/11ef-c95f-8a6adcf0-ac89-09cd97481b93/diagrams/centralization_sERC20-sERC20-_owner.svg

In the contract `sERC20` , the role `initializer` has authority over the functions shown in the diagram below. Any compromise to the `initializer` account may allow the hacker to take advantage of this authority and initialize the staking contract with validation and emit events.



Important Note: Certain identification procedures were attempted to be applied to the project team in order to better understand the centralization situation and potential risks of the project. We strongly advise end users to conduct further research and exercise due diligence before engaging with the project given the centralization related risks. It is crucial for end users to independently verify and assess all available information

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, and decided not to change the current codebase.

[CertiK, 01/15/2025]: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

ERC-02 | NO CAP ON FEES

Category	Severity	Location	Status
Centralization	● **Major**	**projects/audit-fed1/ERC20.sol: 474~482**	● **Acknowledged**

Description

Important Note: Certain identification procedures were attempted to be applied to the project team in order to better understand the centralization situation and potential risks of the project. We strongly advise end users to conduct further research and exercise due diligence before engaging with the project given the centralization related risks. It is crucial for end users to independently verify and assess all available information

There's no cap on the `buyFeeRatio` and `sellFeeRatio` , and they can be up to 100%. If the sell fee is 100%, then users would not be able to receive any proceeds when they try to sell the token.

Recommendation

We recommend adding a reasonable upper bound to both sell fees and adequately disclose them to the community.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, refused to change the current codebase, and provided the following statement:

Issue acknowledged. it's ok. Cap is checked with script.

[CertiK, 01/15/2025]: It is suggested to implement the recommended fix to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

TCK-01 | UNRESTRICTED REWARD MINTER PRIVILEGES AND POTENTIAL MINTER ROLE MISCONFIGURATION

Category	Severity	Location	Status
Centralization	● **Major**	**projects/audit-fed2/Treasury.sol: 526~527**	● **Acknowledged**

Description

Important Note: Certain identification procedures were attempted to be applied to the project team in order to better understand the centralization situation and potential risks of the project. We strongly advise end users to conduct further research and exercise due diligence before engaging with the project given the centralization related risks. It is crucial for end users to independently verify and assess all available information

The treasury contract contains functionality that permits addresses on an allowed list to mint an arbitrary amount of the reward token. This capability is not subject to any restrictions, which could result in the over-issuance of the reward token, leading to its inflation and devaluation. This issue can undermine the contract's economic stability and investor confidence.

Moreover, the reward token contract enforces a role-based access control that only permits addresses with a specific minter role to execute the mint function. If the treasury contract or the addresses with minting privileges are not assigned the minter role within the reward token contract, attempts to mint reward tokens will fail, disrupting the intended reward distribution process.

Scenario

Suppose an attacker gains control of an authorized address that can mint reward tokens. The attacker can mint many reward tokens and exchange these tokens in DEX(e.g., Uniswap). As a large number of reward tokens are sold, the reward token will rapidly depreciate in value

Recommendation

We recommend introducing strict controls and criteria within the treasury contract to govern the minting of reward tokens, ensuring that only authorized actions can trigger the minting process and that the amount minted is within acceptable limits. Additionally, ensure that the vault/treasury contract or the designated minting addresses are correctly configured with the minter role in the reward token contract to avoid operational failures.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, refused to change the current codebase, and provided the following statement:

`isRewardManager` is the mint role mapping.

[CertiK, 01/15/2025]: It is suggested to implement the recommended fix to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

TCK-02 | DEFECTS OF INCURDEBT

Category	Severity	Location	Status
Logical Issue	● Major	projects/audit-fed2/Treasury.sol: 443~444	● Acknowledged

Description

In function `incurDebt` of contract `Treasury` , if one user has `maximumDebt` `sOHM` , he can borrow a total of `maximumDebt` `_token` . This may cause a problem. Consider A, B, and C in `isDebtor` , A has 100 `sOHM` , so he borrows 100 `_token` . Then A transfers these `sOHM` to B, then B can also borrow 100 `_token` . B can transfer these `sOHM` to C , and so on. The reserve token in `Treasury` may suffer a loss.

Users don't need to repay the debt, because they don't mortgage anything.

Proof of Concept

Suppose the contract has 200 UDST, and `address(1)` and `address(2)` are debtors. `address(1)` has 100 SOHM, while `address(2)` does not.

```
function testDrainTokenByincurDebt() public{

        assertEq(_USDT.balanceOf(address(this)), 200 * 10 ** 18);

         _treasury.queue(Treasury.MANAGING.SOHM, address(_sOHM));
         _treasury.queue(Treasury.MANAGING.DEBTOR, address(this));
         _treasury.queue(Treasury.MANAGING.RESERVEDEPOSITOR, address(this));
         _treasury.queue(Treasury.MANAGING.DEBTOR, address(1));
         _treasury.queue(Treasury.MANAGING.DEBTOR, address(2));

         vm.roll(11);

         _treasury.toggle(Treasury.MANAGING.DEBTOR, address(this), address(0));
         _treasury.toggle(Treasury.MANAGING.RESERVEDEPOSITOR, address(this),
address(0));
         _treasury.toggle(Treasury.MANAGING.DEBTOR, address(1), address(0));
         _treasury.toggle(Treasury.MANAGING.DEBTOR, address(2), address(0));
         _treasury.toggle(Treasury.MANAGING.SOHM, address(_sOHM), address(0));

        _USDT.approve(address(_treasury), _USDT.balanceOf(address(this)));
        _treasury.deposit(_USDT.balanceOf(address(this)), address(_USDT), 0);

        uint256 debtAmount = 100* 10 ** 18;

        vm.startPrank(address(1));
        _treasury.incurDebt(debtAmount, address(_USDT));
        _sOHM.transfer(address(2), _sOHM.balanceOf(address(1)));
        vm.stopPrank();

        vm.startPrank(address(2));
        _treasury.incurDebt(debtAmount, address(_USDT));
        vm.stopPrank();

        assertEq(_USDT.balanceOf(address(this)), 0);
    }
```

```
Ran 1 test for test/Treasury.t.sol:TreasuryTest
[PASS] testDrainTokenByincurDebt() (gas: 565048)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 5.70ms (1.49ms CPU
time)
```

Recommendation

We advise the team to consider designing a safer debt strategy for the `Treasury` contract.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, refused to change the current codebase, and provided the following statement:

DAO wallet will audit who can call the incurDebt.

BVC-02 | LOGIC ISSUE IN FUNCTION `deposit()`

Category	Severity	Location	Status
Logical Issue	● Medium	projects/audit-fed1/BondV2.sol: 1115, 1170	● Acknowledged

Description

In the function `deposit()` in contract `BondDepositoryDai` , the require statement in L1115 checks `totalDebt <= terms.maxDebt` to ensure that the depositing does not exceed `terms.maxDebt` . After the depositing, the new deposit value will be added to `totalDebt` in L1170. Thus, the current depositing may still result that `totalDebt` exceeds `terms.maxDebt` and only the next depositing will be blocked.

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design and recommend fixing it if it is against the original design.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, refused to change the current codebase, and provided the following statement:

The `maxDebt` will not be reached in RWA project , it is also acceptable for us.

BVC-03 | INCONSISTENT IMPLEMENTATION OF `terms.minimumPrice` IN FUNCTION `bondPrice()` AND `_bondPrice()`

Category	Severity	Location	Status
Logical Issue	● Medium	projects/audit-fed1/BondV2.sol: 1382, 1393~1394	● Acknowledged

Description

There are two bond price calculation functions:

- `_bondPrice()` , used for nativePrice calculation in `deposit`
- `bondPrice()` , used for priceInUSD calculation in `deposit`

The function `_bondPrice()` update the bond price with the `terms.controlVariable` set by the contract owner. It will check whether the `price_` is less than `terms.minimumPrice` and make sure the new price is greater than `terms.minimumPrice` . However, when the `price_` is greater than or equal to `terms.minimumPrice` at the beginning, the value of `terms.minPrice` is set to 0 making the latter minimum price checking invalid.

We could know the real minimum price would be 100 from this formula:

```
       function _bondPrice() internal returns ( uint price_ ) {
        price_ = terms.controlVariable.mul( debtRatio() ).add( 1000000000 ).div( 1e7
);
        if ( price_ < terms.minimumPrice ) {
            price_ = terms.minimumPrice;
        } else if ( terms.minimumPrice != 0 ) {
            terms.minimumPrice = 0;
        }
    }
```

Recommendation

Recommend to check the usage of terms.minimumPrice to make sure the implementation is expected and aware of the real minimum price declaration.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, decided not to change the current codebase, and provided the following statement:

It is designed for once time.

CKP-03 | OWNERSHIP CAN BE REGAINED AFTER RENOUNCEMENT

Category	Severity	Location	Status
Logical Issue	● Medium	projects/audit-fed1/BondV2.sol: 58~62; projects/audit-fed1/StakingV2.sol: 511~515; projects/audit-fed2/StakingDistributor.sol: 320~324; projects/audit-fed2/Treasury.sol: 177; projects/audit-fed2/sERC20.sol: 983~987	● Acknowledged

Description

After ownership is renounced, the potential for the original owner to regain ownership exists due to the contract code not resetting the state variable for the new owner candidate to a default value.

This situation arises because the function to renounce ownership does not properly clear or update the related state variable. This allows the original owner to exploit this vulnerability under certain conditions, bypassing security checks and regaining ownership of the contract by calling the relevant function again.

Recommendation

We recommend modifying codes to set `_newOwner` to `0` to avoid regaining ownership after renouncement.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, refused to change the current codebase, and provided the following statement:

The owner will not `pushManagement` before `renounceManagement` .

CKP-04 | ANYONE CAN CALL `redeem()` AND `claim()` FOR ANY ARBITRARY `_recipient` ADDRESS

Category	Severity	Location	Status
Access Control	● Medium	projects/audit-fed1/BondV2.sol: 1209, 1253; projects/audit-fed1/StakingV2.sol: 643	● Acknowledged

Description

The public `redeem()` and `claim()` functions in the linked contracts accept an arbitrary `_recipient` address parameter, rather than using `msg.sender`. This design allows any user to call `redeem()` for another account without the account holder's consent. While this could be by design, to facilitate claims through the `Helper` contract on behalf of users, it also opens the possibility for unauthorized claims on other users' behalf. We would like to confirm if this behavior is intentional.

Recommendation

We recommended implementing proper access control mechanisms to prevent unauthorized claim or redeem operations.

Alleviation

[RWA Team, 01/15/2025]: It is safe to call those functions.

From an economic perspective, it is good for users if someone call for them

CKP-05 | LACK OF A PERMISSIONLESS MECHANISM TO REDEEM PRINCIPAL

Category	Severity	Location	Status
Logical Issue, Design Issue	● Medium	projects/audit-fed1/StakingV2.sol: 683; projects/audit-fed2/Treasury.sol: 424~425, 444~445	● Acknowledged

Description

Users deposit `principle` token to the BondV2 contract, and its corresponding OHM token is staked in the StakingV2 contract when users redeem. In the StakingV2 contract, the `unstake()` function can be used to receive OHM token. However, in the Treasury contract, the mechanism to redeem the original `principle` token by using OHM token requires permission from a privileged role, due to the `isReserveSpender()` and `isDebtor()` check. There is no permissionless mechanism for a user to redeem its original `principle` token, not including trading via DEXes outside of the project.

Recommendation

We'd like to understand if this is the intended design. If so, the project team should provide adequate disclosure that unless approved, users might be unable to retrieve its original `principal` token via the in-scope contracts.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, decided not to change the current codebase, and provided the following statement:

To staking users, if they staked the RWA and they will get the same amount of sRWA. sRWA will rebase. They can unstake the sRWA which amount is same as the firs-time staked whenever and retrieve the RWA, the left sRWA will continue rebasing. To treasury , the treasury is very important asset manager , we will manage it with DAO wallet (multi-signature-wallet).

SBC-01 | POTENTIAL FLASHLOAN ATTACK

Category	Severity	Location	Status
Design Issue	● Medium	projects/audit-fed2/StandardBondingCalculator.sol: 262	● Acknowledged

Description

Flash loans are a way to borrow large amounts of money for a certain fee. The requirement is that the loans need to be returned within the same transaction in a block. If not, the transaction will be reverted.

An attacker can use the borrowed money as the initial funds for an exploit to enlarge the profit and/or manipulate the token price in the decentralized exchanges.

We find that the contract `BondingCalculator` relies on price calculations that are based on-chain, meaning that they would be susceptible to flash-loan attacks by manipulating the price of given pairs to the attacker's benefit.

Recommendation

If a project requires price references, it needs to be cautious of flash loans that might manipulate token prices. To minimize the chance of happening, we recommend the client consider following according to the project's business model.

1. Use multiple reliable on-chain price oracle sources, such as Chainlink and Band protocol.
2. Use Time-Weighted Average Price (TWAP). The TWAP represents the average price of a token over a specified time frame. If an attacker manipulates the price in one block, it will not affect too much on the average price.
3. If the business model allows, restrict the function caller to a non-contract/EOA address.
4. Flash loans only allow users to borrow money within a single transaction. If the contract use cases are allowed, force critical transactions to span at least two blocks.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, refused to change the current codebase, and provided the following statement:

RWA pool is very huge which is difficult to influence. And it is acceptable for bond logic.

BVC-04 INCONSISTENT AND MISSING VALIDATIONS IN BOND TERM MANAGEMENT FUNCTIONS

Category	Severity	Location	Status
Volatile Code	● Minor	projects/audit-fed1/BondV2.sol: 952, 994	● Acknowledged

Description

The `setBondTerms()` and `initializeBondTerms()` functions exhibit inconsistencies and omissions in their validation checks:

1. `maxPayout` and `fee` Validation:

    - `setBondTerms()` includes checks for `maxPayout` and `fee` to ensure they meet required conditions.
    - `initializeBondTerms()` omits these checks, potentially allowing invalid bond term configurations.

2. `_inviteRatio` Validation:

    - `initializeBondTerms()` enforces `_inviteRatio` to be strictly less than `10000` .
    - `setBondTerms()` permits `_inviteRatio` to be less than or equal to `10000` , introducing inconsistent logic.

3. `vestingTerm` Validation:

    - Neither function validates the `vestingTerm` , which could lead to misconfigured bond terms.

Recommendation

1. Add `maxPayout` and `fee` validations to `initializeBondTerms()` to match `setBondTerms()` .
2. Standardize the `_inviteRatio` validation logic to ensure consistency across both functions.
3. Introduce a validation check for `vestingTerm` in both functions to prevent invalid configurations.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, decided not to change the current codebase, and provided the following statement:

All param validation will be down by off-chain scripts before calling functions.

BVC-05 | INCONSISTENT SCALING FACTORS IN `getNewBCV()` AND `getNewPrice()` CALCULATIONS

Category	Severity	Location	Status
Inconsistency	● Minor	projects/audit-fed1/BondV2.sol: 1459	● Acknowledged

Description

The `getNewBCV()` and `getNewPrice()` methods use different scaling factors, resulting in an inconsistency in how the calculations are performed:

1. `getNewBCV()` Scaling:
    - When `isLiquidityBond` is `false` , the calculation for `_newbcv` is performed as follows:

```
1431    _newbcv = _price
1432            .mul(1e9)
1433            .div(10 ** IERC20(principle).decimals())
1434            .sub(1000000000)
1435            .div(debtRatio());
```

    - This uses `1e9` as the scaling factor to normalize the price.

2. `getNewPrice()` Scaling:
    - In contrast, the formula used for `_newPrice` is:

```
1455  _newPrice = _bcv
1456        .mul(debtRatio())
1457        .add(1000000000)
1458        .mul(10 ** IERC20(principle).decimals())
1459        .div(100);
```

    - Here, the result is scaled down by dividing by `100` instead of `1e9` . This difference in scaling may lead to inconsistent price and BCV outputs.

Recommendation

To resolve this issue, modify the division in `getNewPrice()` to `1e9` to ensure both functions use the same scaling factor.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding and decided not to change the current codebase.

BVC-06 | USERS CAN ONLY STAKE WHEN THEY REDEEM

Category	Severity	Location	Status
Design Issue	● Minor	projects/audit-fed1/BondV2.sol: 1208~1245	● Acknowledged

Description

The `redeem()` function in the `BondV2` contract has a `_stake` boolean variable as function argument. However, in line 1222 and 1244, the `_stake` variable is not used, but `true` is used in all cases which means that users only have the option to stake their tokens when they call `redeem()`.

Recommendation

We'd like to confirm if this is the intended design, and if so, consider removing unused function argument from the `redeem()` function and ensure that users are aware their tokens will always be staked when they call `redeem()`.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, decided not to change the current codebase, and provided the following statement:

It is designed for this.

CKP-06 HIDDEN ROLE IN THE CONTRACT MAY RAISE CENTRALIZATION CONCERNS

Category	Severity	Location	Status
Coding Issue	● Minor	projects/audit-fed1/BondV2.sol: 59; projects/audit-fed1/StakingV2.sol: 512; projects/audit-fed2/StakingDistributor.sol: 321; projects/audit-fed2/sERC20.sol: 984	● Acknowledged

Description

The contract performs access control check over a certain role. However, the role is currently unavailable via `getter` function. This makes it hard for normal user to get transparent information of the contract and may arise potential confusion.

Recommendation

Consider changing the visibility of the internal role to enhance transparency.

Alleviation

[RWA Team, 01/15/2025]: Event logs are proof when changing owner.

CKP-07 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	projects/audit-fed1/BondV2.sol: 938; projects/audit-fed1/ERC20.sol: 461, 471; projects/audit-fed1/StakingV2.sol: 597, 598; projects/audit-fed2/sERC20.sol: 1142, 1155, 1168	● Acknowledged

Description

The cited address input is missing a check that it is not `address(0)` .

Recommendation

We recommend adding a check the passed-in address is not `address(0)` to prevent unexpected errors.

Alleviation

[RWA Team, 01/15/2025]: Issue acknowledged. I won't make any changes for the current version.

CKP-08 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS (NON-STANDARD ERC20 TOKEN)

Category	Severity	Location	Status
Volatile Code	● Minor	projects/audit-fed1/BondV2.sol: 1159; projects/audit-fed1/StakingV2.sol: 618; projects/audit-fed2/Treasury.sol: 399, 399, 473, 794	● Acknowledged

Description

The project design may not be compatible with non-standard ERC20 tokens, such as deflationary tokens or rebase tokens.

The functions use `transferFrom()` / `transfer()` to move funds from the sender to the recipient but fail to verify if the received token amount matches the transferred amount. This could pose an issue with fee-on-transfer tokens, where the post-transfer balance might be less than anticipated, leading to balance inconsistencies. There might be subsequent checks for a second transfer, but an attacker might exploit leftover funds (such as those accidentally sent by another user) to gain unjustified credit.

Scenario

When transferring deflationary ERC20 tokens, the input amount may not equal the received amount due to the charged transaction fee. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrive to the contract. However, a failure to discount such fees may allow the same user to withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

Recommendation

We advise the client to regulate the set of tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support non-standard ERC20 tokens.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding and decided not to change the current codebase.

CKP-09 | SUSCEPTIBLE TO SIGNATURE MALLEABILITY

Category	Severity	Location	Status
Volatile Code	● Minor	projects/audit-fed1/BondV2.sol: 582; projects/audit-fed1/ERC20.sol: 347; projects/audit-fed2/sERC20.sol: 909	● Acknowledged

Description

The signature malleability is possible within the Elliptic Curve cryptographic system. An Elliptic Curve is symmetric on the X-axis, meaning two points can exist with the same `X` value. In the `r`, `s` and `v` representation this permits us to carefully adjust `s` to produce a second valid signature for the same `r`.

Recommendation

We advise to utilize a `recover()` function similar to that of the `ECDSA.sol` implementation of OpenZeppelin.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding and decided not to change the current codebase.

CKP-10 | POSSIBILITY OF REPLAY ATTACK IN `Permit`

Category	Severity	Location	Status
Volatile Code	● Minor	projects/audit-fed1/BondV2.sol: 582; projects/audit-fed1/ERC20.sol: 347; projects/audit-fed2/sERC20.sol: 909	● Acknowledged

Description

The `permit` function performs the operation of deriving signer address from the signature values of `v` , `r` and `s` . The state variable `DOMAIN_SEPARATOR` that is used to calculate hash has a value of `chainid` that is derived only once in the constructor, which does not change after contract deployment. The issue arises in the event of fork when the cross-chain replay attacks can be executed.

The attack scenario can be thought of as if a fork of Ethereum happens and two different networks have id of for example `1` and `9` . The `chainid` coded in `DOMAIN_SEPARATOR` will be the same on contracts residing in both of the forks. If the chainid `1` is stored in the contract then the `permit` transaction signed for chainid `1` will be executable on both of the forks.

Recommendation

We advise to construct the `DOMAIN_SEPARATOR` hash inside the permit function so the current `chainid` could be fetched and only the transactions signed for current network could succeed.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding and decided not to change the current codebase.

CKP-11 | THIRD-PARTY DEPENDENCIES

Category	Severity	Location	Status
Volatile Code	● Minor	projects/audit-fed1/BondV2.sol: 1129; projects/audit-fed1/ERC20.sol: 514; projects/audit-fed1/StakingV2.sol: 703~705; projects/audit-fed2/StandardBondingCalculator.sol: 252	● Acknowledged

Description

The in-scope contracts interact with several out of scope contracts, such `IFeeReceiver(feeReceiver)` , `ICommunity(community)` , `IReleasePool(releasePool)` , `IUniswapV2Pair(_pair)` . The scope of the audit treats these entities as black boxes and assume their functional correctness. However, in the real world, they can be compromised and this may lead to lost or stolen assets. In addition, upgrades of out of scope contracts can possibly create severe impacts.

Recommendation

We encourage the team to consider adding these contracts to the audit scope, and constantly monitor the statuses of out of scope or 3rd party contracts to mitigate the side effects when unexpected activities are observed.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, decided not to change the current codebase, and provided the following statement:

It is designed for this.

ERC-03 | FUNCTION `_burnFrom()` SHOULD BE `internal`

Category	Severity	Location	Status
Logical Issue	● Minor	projects/audit-fed1/ERC20.sol: 413	● Acknowledged

Description

Function `_burnFrom()` is written as `public`. It is supposed to be `internal` and called by `burnFrom()`.

Recommendation

We recommend changing the visibility to `internal`.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding and decided not to change the current codebase.

ERK-02 | DIVIDE BY ZERO

Category	Severity	Location	Status
Incorrect Calculation	● Minor	projects/audit-fed2/sERC20.sol: 1103	● Acknowledged

Description

In the function `_storeRebase()` , the variable `profit_` is divided by `previousCirculating_` which may be zero. Because in the `initialize()` function, all the initial supply are given to the `stakingContract` account, the `circulatingSupply()` function will return 0.

```
        uint rebasePercent = profit_.mul( 1e18 ).div( previousCirculating_ );
```

Recommendation

We recommend adding zero validation to skip the calculation if `previousCirculating_` is zero.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, decided not to change the current codebase, and provided the following statement:

The `previouseCirculating` will not be 0 in RWA system.

SWC-01 | RETURN VALUE NOT HANDLED

Category	Severity	Location	Status
Volatile Code	● Minor	projects/audit-fed2/StakingWarmup.sol: 91	● Acknowledged

Description

The return value of the `transfer()` function in `retrieve()` is not checked.

```
IERC20( sOHM ).transfer( _staker, _amount );
```

Recommendation

We recommend using variable to receive the return value of the function mentioned above and handle both success and failure cases if needed by the business logic.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, decided not to change the current codebase, and provided the following statement:

The transfer status in RWA system just success or revert.

TCK-03 | LIQUIDITY TOKEN CANNOT BE WITHDRAWN

Category	Severity	Location	Status
Logical Issue	● Minor	projects/audit-fed2/Treasury.sol: 398, 423, 424, 445	● Acknowledged

Description

In the `deposit()` function of contract `Treasury` , both the reserve and liquidity token can be deposited. But in L423 of function `withdraw()` , there is a `require` statement only allowing withdrawing reserve tokens. No withdrawal method is provided in this contract to withdraw the liquidity token.

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, decided not to change the current codebase, and provided the following statement:

It is designed for this.

TCK-04 MISSING VALIDATION FOR `sOHMQueue` IN `toggle` FUNCTION

Category	Severity	Location	Status
Volatile Code	● Minor	projects/audit-fed2/Treasury.sol: 757	● Acknowledged

Description

The queue is set with `sOHMQueue` , but the `toggle` function does not include a check to ensure that `sOHMQueue <= block.number` . This oversight allows the queue to potentially be toggled prematurely, which could lead to inconsistencies in the contract’s behavior, especially in scenarios where the queue's state should be controlled by the block number.

Without this check, the contract could experience issues where the toggling occurs out of order, potentially affecting the sequence of operations or triggering unintended actions.

Recommendation

Modify the `toggle` function to include a validation that ensures `sOHMQueue <= block.number` before allowing the queue to toggle. This check will ensure that the toggling process occurs only when the correct block number is reached, preventing premature actions and ensuring the integrity of the queue's timing mechanism.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding and decided not to change the current codebase.

BVC-07 | UNUSED VARIABLES

Category	Severity	Location	Status
Coding Issue	● Informational	projects/audit-fed1/BondV2.sol: 1080, 1296	● Acknowledged

Description

The `stakeOrSend()` function has a `_invite` boolean variable in function argument. However, the `_invite` variable is never used in the body of the function.

Similarly, the `setContract()` function can set a `rewardDistributor` address, but the `rewardDistributor` address is never utilized in the contract.

Recommendation

We'd like to confirm if this aligns with the intended design. Consider removing unused function arguments and state variables.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding and decided not to change the current codebase.

CKP-12 | EVENT NOT INDEXED

Category	Severity	Location	Status
Design Issue	● Informational	projects/audit-fed1/BondV2.sol: 851~856; projects/audit-fed1/ERC20.sol: 446	● Acknowledged

Description

If an event is not indexed in a smart contract, it means that the event's parameters are not tagged with the `indexed` keyword. This has implications for how the event data can be searched and filtered when looking through blockchain logs.

Without indexing, the event will still emit the data as part of the transaction log, but users won't be able to query for these events using the parameters. They'll have to retrieve the entire set of logs and manually sift through them to find events with the specific data. This can be less efficient and more time-consuming, especially on a blockchain with a high volume of transactions and events.

Recommendation

To mitigate this issue, it is recommended to index the most relevant parameters in the event to be defined.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, decided not to change the current codebase, and provided the following statement:

It is ok for RWA system.

CKP-13 | MISSING INPUT VALIDATION

Category	Severity	Location	Status
Logical Issue	● Informational	projects/audit-fed1/BondV2.sol: 392~399, 415~431, 432~443, 444~458; projects/audit-fed1/ERC20.sol: 216~220, 230~237, 238~242, 243~251; projects/audit-fed2/sERC20.sol: 612~616, 652~657, 670~674, 689~693, 1179~1184	● Acknowledged

Description

The mentioned functions lack the zero check for the parameter `amount`. When `amount == 0`, these functions won't make any changes in the contract. Additionally, self transfer where `to` equals `msg.sender` also should have no effect.

Recommendation

Consider adding the relevant checks in the transfer functions.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding and decided not to change the current codebase.

CKP-14 | SPENDERS WITH INFINITE ALLOWANCE HANDLED INCORRECTLY

Category	Severity	Location	Status
Coding Style	● Informational	projects/audit-fed1/BondV2.sol: 415~431; projects/audit-fed1/ERC20.sol: 230~237; projects/audit-fed2/sERC20.sol: 652~657	● Acknowledged

Description

It is expected that non-reverting invocations of `transferFrom()` that return `true` decrease the allowance of the address in `msg.sender` for the address in `sender` by the value in `amount` .

An allowance that equals `type(uint256).max` is treated as an exception and interpreted as an unlimited allowance that does not need to be reduced in order for this check to pass. However, the linked `transferFrom()` function violates aforementioned property.

Recommendation

It is recommended to account for the case of a spender's allowance being `type(uint256).max` by excluding it from an update to its allowance.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, decided not to change the current codebase, and provided the following statement:

It is safe.

CKP-15 | WRONG ADDRESS IN `_mint()` FUNCTION

Category	Severity	Location	Status
Logical Issue	● Informational	projects/audit-fed1/BondV2.sol: 478; projects/audit-fed1/ERC20.sol: 266; projects/audit-fed2/sERC20.sol: 730	● Acknowledged

Description

The function `_beforeTokenTransfer()` and event `Transfer` in the `_mint()` function generally use 0 address as a parameter, but here is `address(this)` .

Recommendation

We recommend to modify as follow:

```
function _mint(address account_, uint256 ammount_) internal virtual {
    require(account_ != address(0), "ERC20: mint to the zero address");
    _beforeTokenTransfer(address(0), account_, ammount_);
    _totalSupply = _totalSupply.add(ammount_);
    _balances[account_] = _balances[account_].add(ammount_);
    emit Transfer(address(0), account_, ammount_);
}
```

Alleviation

[RWA Team, 01/15/2025]: It is designed for this.

CKP-16 | CONTRACTS WITH TODOS

Category	Severity	Location	Status
Coding Issue	● Informational	projects/audit-fed1/BondV2.sol: 349; projects/audit-fed1/ERC20.sol: 170; projects/audit-fed2/sERC20.sol: 516	● Acknowledged

Description

"TODO" comments within smart contract code could signal potential vulnerabilities due to the presence of undeveloped or incomplete logic. It is also possible that these comments were left behind after the completion of the intended features, indicating a lack of code cleanup and final review.

Additionally, if "TODO" features are implemented post-audit, there is a risk of introducing new vulnerabilities that were not present during the initial security assessment.

Recommendation

To mitigate this issue, it's important to:

1. Finalize all contract features and logic before deployment, removing any "TODO" comments to ensure the code is complete.
2. Conduct a comprehensive audit of the smart contract after any significant updates or additions, including those previously marked as "TODO."

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding and decided not to change the current codebase.

CKP-17 | USING LIBRARY FOR ALL IS DEPRECIATED

Category	Severity	Location	Status
Coding Style	● Informational	projects/audit-fed1/BondV2.sol: 829; projects/audit-fed2/StandardBondingCalculator.sol: 264	● Acknowledged

Description

When use `using LIB for *;` , it means the contract is attaching the library to all types. This is generally discouraged because it can lead to unexpected behavior and potential name conflicts. It essentially imports the functions of the library into the global namespace for all types, which can override existing functions or create confusion.

Recommendation

While using `using LIB for *;` is technically possible in Solidity, it's generally considered a risky practice due to the potential for unintended consequences and name clashes. It's usually better to explicitly specify which types you want to attach the library to

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding and decided not to change the current codebase.

CKP-18 | MISSING ERROR MESSAGES

Category	Severity	Location	Status
Coding Style	● Informational	projects/audit-fed1/BondV2.sol: 927, 929, 931, 933, 935, 1050, 1074, 1593, 1594; projects/audit-fed1/StakingV2.sol: 586, 588, 757, 767; projects/audit-fed2/StakingDistributor.sol: 321, 367, 369, 464, 477; projects/audit-fed2/StakingWarmup.sol: 83, 85, 90; projects/audit-fed2/StandardBondingCalculator.sol: 271; projects/audit-fed2/Treasury.sol: 374, 603, 658; projects/audit-fed2/sERC20.sol: 996, 1043, 1044, 1058	● Acknowledged

Description

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

Recommendation

We advise adding error messages to the linked **require** statements.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding and decided not to change the current codebase.

CKP-19 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	● Informational	projects/audit-fed1/BondV2.sol: 952, 994, 1024, 1049, 1070, 1084; projects/audit-fed1/ERC20.sol: 401, 470; projects/audit-fed1/StakingV2.sol: 756, 766, 778, 799; projects/audit-fed2/StakingDistributor.sol: 315, 463, 476, 489; projects/audit-fed2/sERC20.sol: 1057	● Acknowledged

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding and decided not to change the current codebase.

ERC-01 | DISCUSSION ON DESIGN

Category	Severity	Location	Status
Design Issue	● Informational	projects/audit-fed1/ERC20.sol: 514	● Resolved

Description

The current implementation of the contract logic may diverge from the expected design in the following areas:

ERC20TokenX Contract (_transfer()):

- When users sell tokens or add liquidity, the `feeReceiver` receives the sell fee and triggers the `triggerSwap()` function. However, when users buy tokens or remove liquidity, the `feeReceiver` only receives the buy fee but does not trigger the `triggerSwap()` function, which could be inconsistent with the project's expected behavior for fee handling during token purchases or liquidity removals.

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, decided not to change the current codebase, and provided the following statement:

It is designed for this.

ERK-03 | INCORRECT COMMENT

Category	Severity	Location	Status
Coding Style	● Informational	projects/audit-fed2/sERC20.sol: 518~536	● Acknowledged

Description

In contract `ERC20` , there are several variables that have the comment `Present in ERC777` , but this contract does not implement the `IERC777` interface. The comment `Present in ERC777` is incorrect.

Recommendation

Remove the comment `Present in ERC777` or use a correct comment.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding and decided not to change the current codebase.

SVC-01 | DISCUSSION ON LOCKBONUS

Category	Severity	Location	Status
Design Issue, Logical Issue	● Informational	projects/audit-fed1/StakingV2.sol: 733~740, 748~750, 756~760, 766~770	● Acknowledged

Description

The `epoch.distribute` value is calculated as `contractBalance() - IsOHM( sOHM ).circulatingSupply()` . The `contractBalance()` includes the `totalBonus` amount, which can be adjusted by the `locker` via the `giveLockBonus()` and `returnLockBonus()` functions. In these functions, the `totalBonus` increases or decreases for the same amount as the increase or decrease of the circulating supply of sOHM tokens. The result is that the changes in `contractBalance()` is offset by `IsOHM( sOHM ).circulatingSupply()` 1 for 1.

Recommendation

We'd like to understand the intention of the `giveLockBonus()` and `returnLockBonus()` functions, as they appear ineffective at regulating the `epoch.distribute` value.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding, decided not to change the current codebase, and provided the following statement:

It is a deprecated feature.

OPTIMIZATIONS | RWA ECOSYSTEM

ID	Title	Category	Severity	Status
BVC-01	User-Defined Getters	Gas Optimization	Optimization	● Acknowledged
CKP-01	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	● Acknowledged

BVC-01 | USER-DEFINED GETTERS

Category	Severity	Location	Status
Gas Optimization	● Optimization	projects/audit-fed1/BondV2.sol: 1088~1090	● Acknowledged

Description

The linked functions are equivalent to the compiler-generated getter functions for the respective variables.

Recommendation

We advise that the linked variables are instead declared as `public` as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding and decided not to change the current codebase.

CKP-01 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	● Optimization	projects/audit-fed1/BondV2.sol: 363; projects/audit-fed1/ERC20.sol: 189, 437; projects/audit-fed2/sERC20.sol: 535	● Acknowledged

Description

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

Recommendation

We recommend declaring these variables as immutable.

Alleviation

[RWA Team, 01/15/2025]: The team acknowledged the finding and decided not to change the current codebase.

FORMAL VERIFICATION | RWA ECOSYSTEM

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply` , which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows (note that overflow properties were excluded from the verification):

Property Name	Title
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-transfer-false	If `transfer` Returns `false` , the Contract State Is Not Changed
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-transferfrom-revert-zero-argument	`transferFrom` Fails for Transfers with Zero Address Arguments
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly

Property Name	Title
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Transfers
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Transfers
erc20-approve-never-return-false	`approve` Never Returns `false`
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20-approve-false	If `approve` Returns `false`, the Contract's State Is Unchanged
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-transfer-never-return-false	`transfer` Never Returns `false`
erc20-approve-succeed-normal	`approve` Succeeds for Valid Inputs
erc20-transferfrom-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance

Verification Results

In the remainder of this section, we list all contracts where formal verification of at least one property was not successful. There are several reasons why this could happen:

- False: The property is violated by the project.
- Inconclusive: The proof engine cannot prove or disprove the property due to timeouts or exceptions.
- Inapplicable: The property does not apply to the project.

Detailed Results For Contract sERC20 (projects/audit-fed2/sERC20.sol) In SHA256 Checksum 53b9aaefda4174ff004a104cf919b7f2b65e0a10

Verification of ERC-20 Compliance

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-exceed-balance	● Inapplicable	The property does not apply to the contract
erc20-transfer-revert-zero	● False	
erc20-transfer-false	● True	
erc20-transfer-correct-amount	● False	
erc20-transfer-never-return-false	● True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	
erc20-balanceof-succeed-always	● False	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	
erc20-allowance-succeed-always	● True	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-revert-zero-argument	● False	
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-correct-amount	● False	
erc20-transferfrom-never-return-false	● True	
erc20-transferfrom-false	● True	
erc20-transferfrom-fail-exceed-balance	● Inapplicable	The property does not apply to the contract

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-change-state	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-succeed-always	● True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-never-return-false	● True	
erc20-approve-false	● True	
erc20-approve-revert-zero	● False	
erc20-approve-correct-amount	● True	
erc20-approve-succeed-normal	● True	

APPENDIX | RWA ECOSYSTEM

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Checksum Calculation Method

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

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

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed ERC-20 Properties

Properties related to function `transfer`

erc20-transfer-correct-amount

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

Specification:

```
requires recipient != msg.sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(recipient) == \old(balanceOf(recipient) + amount)
&& balanceOf(msg.sender) == \old(balanceOf(msg.sender) - amount);
  also
requires recipient == msg.sender;
ensures \result ==> balanceOf(msg.sender) == \old(balanceOf(msg.sender));
```

erc20-transfer-exceed-balance

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

Specification:

```
requires amount > balanceOf(msg.sender);
ensures !\result;
```

erc20-transfer-false

If the `transfer` function in contract `sERC20` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transfer-never-return-false

The transfer function must never return `false` to signal a failure.

Specification:

```
ensures \result;
```

erc20-transfer-revert-zero

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

Specification:

```
ensures \old(recipient) == address(0) ==> !\result;
```

Properties related to function `balanceOf`

erc20-balanceof-change-state

Function `balanceOf` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-balanceof-correct-value

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner` .

Specification:

```
ensures \result == balanceOf(\old(account));
```

erc20-balanceof-succeed-always

Function `balanceOf` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `allowance`

erc20-allowance-change-state

Function `allowance` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-allowance-correct-value

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner` .

Specification:

```
ensures \result == allowance(\old(owner), \old(spender));
```

erc20-allowance-succeed-always

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `transferFrom`

erc20-transferfrom-correct-allowance

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount` .

Specification:

```
ensures \result ==> allowance(\old(sender), msg.sender) == \old(allowance(sender,
msg.sender)) - \old(amount)
                  || (allowance(\old(sender), msg.sender) == \old(allowance(sender,
msg.sender)) && \old(allowance(sender, msg.sender)) == type(uint256).max);
```

erc20-transferfrom-correct-amount

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest` .

Specification:

```
requires recipient != sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient) +
amount)
                  && balanceOf(\old(sender)) == \old(balanceOf(sender) - amount);
  also
requires recipient == sender;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient));
```

erc20-transferfrom-fail-exceed-allowance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

Specification:

```
requires msg.sender != sender;
requires amount > allowance(sender, msg.sender);
ensures !\result;
```

erc20-transferfrom-fail-exceed-balance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

Specification:

```
requires amount > balanceOf(sender);
ensures !\result;
```

erc20-transferfrom-false

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transferfrom-never-return-false

The `transferFrom` function must never return `false` .

Specification:

```
ensures \result;
```

erc20-transferfrom-revert-zero-argument

All calls of the form `transferFrom(from, dest, amount)` must fail for transfers from or to the zero address.

Specification:

```
ensures \old(sender) == address(0) ==> !\result;
also
ensures \old(recipient) == address(0) ==> !\result;
```

Properties related to function `totalSupply`

erc20-totalsupply-change-state

The `totalSupply` function in contract sERC20 must not change any state variables.

Specification:

```
assignable \nothing;
```

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract sERC20.

Specification:

```
ensures \result == totalSupply();
```

erc20-totalsupply-succeed-always

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `approve`

erc20-approve-correct-amount

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` .

Specification:

```
requires spender != address(0);
ensures \result ==> allowance(msg.sender, \old(spender)) == \old(amount);
```

erc20-approve-false

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-approve-never-return-false

The function `approve` must never returns `false` .

Specification:

```
ensures \result;
```

erc20-approve-revert-zero

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

Specification:

```
ensures \old(spender) == address(0) ==> !\result;
```

erc20-approve-succeed-normal

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

Specification:

```
requires spender != address(0);
ensures \result;
reverts_only_when false;
```

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK’s prior written consent in each instance.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED “AS IS” AND “AS AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER’S OR ANY OTHER PERSON’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Entire Web3 Journey

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