Cyberscope

Audit Report

Tangible USDR Ecosystem

January 2022

Github	https://github.com/TangibleTNFT/usdr-contracts
Commit	8aabb3c13093026bb913acfc83e4afa0155b40cc
Audited by	© cyberscope

Table of Contents

Contracts Review

Github	https://github.com/TangibleTNFT/usdr-contracts
Commit	8aabb3c13093026bb913acfc83e4afa0155b40cc

Audit Updates

Initial Audit	24th November 2022 https://github.com/cyberscope-io/audits/blob/main/TNGBL/v1/audit.pdf
Corrected	14 December 2022

Testing Deploy

Contract Name	Explorer
USDR	https://testnet.bscscan.com/address/0xcE4fCB9fDc79e2172a6734903EAE85C8cdc26D49
pDAI	https://testnet.bscscan.com/address/0xE2B7335874c7d6C653DA1b9250A25e6fFe485fBb
USDRMigration	https://testnet.bscscan.com/address/0xcC60bBfA612364F6560B73228138AF1634a8bA42
WadRayMath	https://testnet.bscscan.com/address/0x14CCD639dA77017a3ec6A1FB8da3067270676426
AddressProvider	https://testnet.bscscan.com/address/0xE04b59EeA56F57341D4a1e3e2989A0b5853D8205
DAIBond	https://testnet.bscscan.com/address/0x82ab99DA543460a7e376195430cac4A6F8f185b4
IncentiveVault	https://testnet.bscscan.com/address/0x8cd4889264485050269A070578D7025a0BC275C1
RWACalculator	https://testnet.bscscan.com/address/0x9C9891F7AeCde5964ff790f1276718fA9cdCB8e1
AffiliateExchange	https://testnet.bscscan.com/address/0xAAb3f7a587d68C9A1E406c4ab5f4177141A2a652
TNGBLPriceOracle	https://testnet.bscscan.com/address/0x3f628f6f3084460B197eEfBA2E465C1527C17054
TokenSwap	https://testnet.bscscan.com/address/0x9a3555ce5191bC44eE8bCaa8C5F343785F1Af5a0
TreasuryTracker	https://testnet.bscscan.com/address/0x64E40EE397324ada86B0D9B66798c3F4b3455913
USDRTreasury	https://testnet.bscscan.com/address/0x5CF531496A9571197c165243c88fa98c3d490285

USDRBonding	https://testnet.bscscan.com/address/0x054C017372a96b59b0D34D1ca58CF2019140d7dE
USDRExchange	https://testnet.bscscan.com/address/0x7B3a673aC98E77aC2d31FdBdcd0338F47eC8bb50
USDRExchangeProxy	https://testnet.bscscan.com/address/0x1bc20907d0B2AbD05fa0915D461e28Eca270fD56
GoldPurchaseManager	https://testnet.bscscan.com/address/0x395453Ce327586e2749784730e3a9C06db3d0a17
GoldSellManager	https://testnet.bscscan.com/address/0x2aD1cDeE617ea36f49456af31f1A967a9113a954
REPurchaseManager	https://testnet.bscscan.com/address/0x6C61C6745984F0fA89606E6CF87d2A14f107FD3E
ReSellManager	https://testnet.bscscan.com/address/0x675358F10e8708839E604501e7385a74FC291F17
LiquidityTokenMath	https://testnet.bscscan.com/address/0x5CC78a6C7Ed42f3E26dC1593f3E7893526353955#code
TNGBLLockedValue	https://testnet.bscscan.com/address/0xebBca671b43006994Af072ecc761229BFaE5f118#code

Source Files

Filename	SHA256
AddressAccessor.sol	46ad8af4ff314b835974a6e026ce942ff3f56fd5c319c68d3f128c670b00a169
AddressProvider.sol	e463c6d775f0ad1c4c5bef219b2391fefb0d264c45b14d848e9d8a0762be2eb4
AffiliateExchange.sol	9003678791100a2fb76b982cd167eb59d8179227d77b888e77747f0f6e872717
constants/addresses.sol	c31754a2076cb7d559c2bf0f1df0a8a63da1e6eead5d879e019d4dd7d75b257e
constants/constants.sol	7d2a05b12ba5b135dfbc67ab2693d65a82817e5f3d7a40832ba32847ba318073
constants/roles.sol	493938a66b0c64554700a1b7c1f2233709889f04ff961cca33a27f034358068e
DAIBond.sol	9a80923bdc8ff83e41752f6e2bf7325f53ea76aa9d936f04db640c73ab23568f
exchangeHelpers/CurveWrapper.sol	2526ceb9cb5618a2cf467c9af0db65db761f2271d74c6d6e6ac2185f1c654a4a
IncentiveVault.sol	5d830e1d1335998d956109274caea5591262823f1017c449c5271b7fb06d5b95
interfaces/IExchange.sol	b1eac05067975153ee292030fcdf7b13c0bcf811f3f1fc7adc520732789216af
interfaces/ILiquidityManager.sol	ce89617a1b542c489905cf5caa47f60cde10a1d7f53538a91a922bb51fc864cc
interfaces/ILiquidityTokenMath.sol	ba076342c2a184981c420825e099837e5ec5f534ab07856dc5132e1623958769
interfaces/IPriceOracle.sol	ae7217ef73c1e591bb8c2011b02725ebb9a23858483f6386f83918cdecb86f0b

interfaces/IRWACalculator.sol	d51f50b9c236b701e7f5718832ac39da016f52cea295f2ca51fd8f8a2d375f6c
interfaces/ITokenSwap.sol	95d277d47fbfe5cfd4a628f6070911535f243369def359b3fdf752fdb1bc11b8
interfaces/ITreasury.sol	18d830188842c79a31cdc10529ced43d1244024e8360d24959fbc540c94a4daf
interfaces/ITreasuryTracker.sol	907f6bad9a6c3b544c1d87f1d240342eeefe415caa919f59aa002be3ba7a7f61
interfaces/IUSDR.sol	20dbd287b49061fb82e184c66e9b2514ad711df2e61bdb904cd5cde6ec4f3e1e
managers/GoldPurchaseManager.sol	d0a7570c64fe2690da8f05fea4f061c782e6ae0a5483b5c378a7d02d4f37a26a
managers/GoldSellManager.sol	5d66782cd962cdaef72c0b7048eb98a045f9b2490b11d999254c74d0f4d17e7d
managers/OnSaleTracker.sol	3a8dc69619590a69e4a58da5338ee84a282cb445cc5a40e3d67a1648e0d170c9
managers/PurchaseManager.sol	eab50fc8a9f54d0711f6574d2eeb3a385356e8478d2b0a2f27d0bda9f7575409
managers/REPurchaseManager.sol	ea93548deb60fa54eb82d03735bae0e26759ccb287b9581a2db39b437aa9a882
managers/RESellManager.sol	62aee70bf24b91fce1e7bfa909d7ea1ab6c1edbf1a96c7824895d64a92e4c7cd
mocks/MockDAIUSDOracle.sol	e4ac02cbf89a705bfd92c4048c7c6c61a83e3b57efa9c68b6be478b41819ec09
mocks/MockERC20.sol	7da7e85841d1a129c3c8a3f5a90946d5b67171aa999425bb1926540ab179a1d7
mocks/MockOracle.sol	bbf25e4605c4e4db4c31347926822f62460a0fd772afc662d0e93a7b7b502246
mocks/MockRouter.sol	5355d45c89f5ca4eadbb3f42669399a1dbc4161217f4fdc57354243725f2b605

mocks/MockTreasury.sol	b8e333e74b74ab4463b8093cdd350c87d2620784c6fdcb09b2ea5780e2a91531
RWACalculator.sol	ce4607e1b89cd49f37bf0c0c0469e6ef4477678640da9184bf550f2822085f47
strategies/IStrategy.sol	d31577b6d9baae01d854796d8f08261290fc6890271325dd4ae5f06933902186
strategies/sushi/IMiniChefV2.sol	a8c2b43069a70f4edc0927e764620eb31f874ab6898e285ed375bedd2daa1d49
strategies/sushi/SushiLPStrategy.sol	81231011b0ba41e89f9e984151a2c914752cafbfac16df8d2109caacef68f957
tangibleInterfaces/ICurrencyFeed.sol	2963764bbc69fa849f23f8fa9dd0a72934b7075e5a015bda21beaa090e130668
tangibleInterfaces/IInstantLiquidity.sol	c20fb58fda62afee9c2827bd1e97593f9e6e690390915010435 93b181d5e59f2
tangibleInterfaces/IPriceOracle.sol	2c5550244dbe7a670f802ce047735b83046a05f1640c3a850e6b53470146f140
tangibleInterfaces/ITangibleInterfaces.sol	d9cdd973e5dfd240a66bd5e13e03474b81be5682b97e41c01b9595273f8054a0
tangibleInterfaces/ITangibleMarketplace.sol	ea7e05dece3b0256027e7e8b0daa89e1676e4e9bf3fb39a78728bc0873c11243
tangibleInterfaces/ITangiblePiNFT.sol	ea4fc5a17bb5a8a1c3237fe87195d53a1310cb83f58e9db6f0df89eb571cc0e3
tangibleInterfaces/ITangibleRentShare.sol	2b3c3508bc80f59984f65392983ea1737f13cd4f21ed6e8b2f941f2b550c0fcb
tangibleInterfaces/ITangibleRevenueShare.sol	01ec5c77269060ee39e176e835913ec23b1e0cf72c277a0ab57611a0a5913ac0
TNGBLLockedValue.sol	5e6d60767026c31b183ea80f9bbdaa50239527c137297417a65ab599d221ae1c

TNGBLPriceOracle.sol	2cee6ecdaa1478a4ffba5846e2266b245c6d49d95dffea6d2663dc1121674437
tokens/interfaces/IMintableERC20.sol	49695950b7dae818cdb5e2dd5d67ba5b0f63bb129089a349da58f986532f145f
tokens/interfaces/ITangibleERC20.sol	933e943c676ae403d1c2594985d54de001fa4125bc98a900ddfe46145460b116
tokens/pDAI.sol	296342f000bb85c43f0cab1e11ef3d257cc71044378022674835cf6b637b0ae8
tokens/USDR.sol	23e7581f6b671543e55d59241fba4d28e3a95706ae416b41becf9f01f4783984
tokens/USDRMigration.sol	3124f3d094fa4a2317c04f26a0256e2989eacd4dfe8f3c2f729d7a75b350a728
tokens/WadRayMath.sol	af5f9434986200e7960994953c4964c7a7ec275b2f18d46bd0e10a5351ba9dc1
TokenSwap.sol	d6df5354075fef529ba95d7626e8302ddf0ee1031dabd8d158e153d04916f048
TreasuryTracker.sol	f083e0eab88f134875ab71743615b2263cc7cf025afd1977698a1d93288e4f9d
USDRBonding.sol	25e1a6577269848d8cb29d8cfc4163ecadcd8ef5c8d4ac48850246551e33dcd5
USDRExchange.sol	64bd37ab5cb6b575de72d9b920d82f72d5f1f0c58b0f510c9cde0589ea616d9b
USDRExchangeProxy.sol	296f086f87701597cae8954762c230fd8fb6b4804c1d689331defc5c21dabcc7
USDRTreasury.sol	d7a34e381c371d53971a5fd61dde3cc4accf0da576b1eee40583f61bdae711ef

DAO Architecture Concerns

The Tangible contracts depend on the correct administrator's configuration. One category of configuration is the funds allocation. The amount of funds that will be moved to the Bond program, the Affiliate and Incentive features are determined by the administrators. Another category of configuration is the direct state manipulation. As the following example of the USDRExchange contract depicts.

```
function updateMintingStats(int128[7] calldata delta)
      external
      onlyRole(DEFAULT_ADMIN_ROLE) {...}
```

There are code segments where the code implements safety checks. These checks are trying to guarantee the balance of the contracts according to the business logic. The abovementioned example depicts that there are cases where this approach could be more autonomous.

Recommendation

We advise the team to create contracts that will implement the expected business logic. These contracts could communicate with Tangible contracts and replace the required human interaction. For instance, a contract could check the accumulated amounts of the treasury and decide how to distribute the funds to the vaults like (Bond, Affiliate, Incentive), liquidity, reserves, USDR binding balance, etc. This methodology will remove the error-prone human interaction and make the ecosystem operate as autonomously as possible.

Team’s Reply 20 December 2022

The team has acknowledged that this is not a security issue and states:

“The functions mentioned was needed for migration from V1 of USDR contracts and it can only be called from multisig. In the future when we migrate to V3 we will remove this, since we just needed it that one time in migration.

Many DAOs allow changes to the business logic by the DAO multisig. This allows holders to vote on changes to the business logic and members of the mutlisig to implement those changes.

This is in our mind is definitely a feature not a bug. No one has ever created a stablecoin backed by real estate before and the exact percentages of the backing and many other parts of the business logic may need to be changed based on data that we gather whilst live. The ability to change this logic via a tx rather than a redployment and tokenswap is very much so deliberate.”

Decimal Architecture

The Tangible contracts utilize many tokens that interact with each other. Each token may have a different amount of decimals. As a result, the contracts are normalizing the tokens in order to proceed with the calculations.

The contracts do not have a single point of decimals normalization mechanism. As a result, a lot of issues may be produced:

There are many sections that are **repeating the decimals normalization logic**.

```
function _scaleAmount(
        uint256 amount,
        uint8 fromDecimals,
        uint8 toDecimals
    ) private pure returns (uint256)

function _convertToCorrectDecimals(
    uint256 price,
    uint8 salePriceDecimals,
    uint8 treasuryTokenDecimals
)

function scaleToUnderlying(uint256 amount) external view returns (uint256)
```

Creates **unnecessary dependencies** between contracts.

```
// IncentiveVault.sol
uint256 amount = (IExchange(exchange).scaleToUnderlying(
    IERC20(USDR).totalSupply()
) * apr) / 3_650_000;
```

Using **fixed values from token addresses** that may change. The Decimal Scale Missconsern finding describes a related issue.

```
uint256 amount = ((minted - redeemed) * 1e26) /
IPriceOracle(tngblOracle).quote(1e18);
```

Recommendation

The Tangible Ecosystem could implement a single point of decimals handling functionality similar to the Address Provider contract. All decimal-related calculations should always use the token's decimals and not a fixed number.

Team's Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

"We had different functions for different needs when we needed to handle decimals. They were made by different devs and work the same so essentially, they could be merged into one, but that is just code optimization, and our main goal and functionality of the contracts are NOT changed nor affected. Hardcoded values are there because decimals of TNGBL token, DAI, and our USDR will not change, so we decided to do it like this to optimize gas spending, and contract size."

Roles Architecture

The Tangible contracts have a role-based access mechanism. Every contract contains its own access layer. The roles that are used are:

- BURNER
- MINTER
- CONTROLLER
- TRACKER
- ROUTER_POLICY

The team is advised to use a multi-sig wallet which can provide an additional level of security. Additionally, the team should make sure that each role has a clear set of responsibilities, and that these responsibilities are not overlapping or ambiguous.

The following example depicts a potential conflict between the CONTROLLER role and the treasury address. Since this method is solely accessed by the treasury address then the role-based permissions are redundant.

```
function rebase(uint256 supplyDelta)
        external
        onlyRole(CONTROLLER_ROLE)
        whenNotPaused
    {
        (address treasury, address exchange) = abi.decode(
            addressProvider.getAddresses(
                abi.encode(TREASURY_ADDRESS, USDR_EXCHANGE_ADDRESS)
            ),
            (address, address)
        );
        require(msg.sender == treasury, "caller is not treasury");
```

Team’s Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

“We have general architecture in place for controller role and this is in place to make sure we are ok in possible migrations for this particular function you mentioned – address in AddressProvider could return treasury that doesn’t have controller role.”

Contract Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	AFI	Affiliate Token Issue	Acknowledged
Medium	STI	Staking Token Issue	Acknowledged
Minor / Informative	DMI	Defractionalize Manipulation Issue	Acknowledged
Minor / Informative	TBI	Token Balance Inconsistency	Acknowledged
Minor / Informative	PRD	Pair Reserves Diversion	Acknowledged
Minor / Informative	TAZFA	Transferred Amount Zero Fees Assumption	Acknowledged
Minor / Informative	AIC	Arguments Inconsistency	Acknowledged
Minor / Informative	ELFM	Exceeds Fees Limit	Acknowledged
Minor / Informative	DSM	Decimal Scale Missconsern	Acknowledged
Minor / Informative	PIL	Potential Infinite Loop	Acknowledged
Minor / Informative	STC	Succeed Transfer Check	Acknowledged
Minor / Informative	CO	Code Optimization	Acknowledged
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Acknowledged

●	L09	Dead Code Elimination	Acknowledged
●	L11	Unnecessary Boolean equality	Acknowledged
●	L12	Using Variables before Declaration	Acknowledged
●	L13	Divide before Multiply Operation	Acknowledged
●	L14	Uninitialized Variables in Local Scope	Acknowledged
●	L15	Local Scope Variable Shadowing	Acknowledged

AFI - Affiliate Token Issue

Criticality	medium
Location	contracts/AffiliateExchange.sol#82
Status	Acknowledged

Description

There are functions that are used to swap a specified amount of tokens. There fee or tax is an amount that is charged when tokens are swapped. According to the specification, the swapped amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

For instance, the contract utilizes the variable affiliatePayout which contains the total rewarded amount.

1. Every affiliate participation requires an amount of USDR. If the contract's USDT is not sufficient, then the proportional amount of TGBL is swapped to cover the cap.
2. The swapFromTNGBL is used to swap TGBL for USDR. The swapFromTNGBL method applies fees on the transaction. Thus, the actual total rewards will be lower than the aggregated amount.
3. During the redemption phase, the sendRewards will send more than the actual contract's reserves.

This will cause the method to revert since the balance of the contract will be less than the calculated _pending variable.

```
function _mint( uint256 amountIn, address receiver, uint256 affiliatePayout )
private {
        (
            address underlying,
            address exchange,
            address tngbl,
            address oracle,
            address usdr
        ) = abi.decode(
         //..
        uint256 excessUSDR = IERC20(usdr).balanceOf(address(this)) - _pending;
```

```
        if (affiliatePayout > excessUSDR) {
            uint256 mintExtra = affiliatePayout - excessUSDR;
            uint256 tngblPrice = IPriceOracle(oracle).quote(1e18);
            uint256 mintExtraInTNGBL = (mintExtra * 1e27) / tngblPrice;
            IERC20(tngbl).approve(exchange, mintExtraInTNGBL);
            USDRExchange(exchange).swapFromTNGBL(
                mintExtraInTNGBL,
                0,
                address(this)
            );
        }
        _pending += affiliatePayout;
    }
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

Team’s Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

“The contracts in question are there in marketing and incentive purposes and they are tailored to the current situation. When they become obsolete, we will replace them. For now, there is no fee on minting with TNGBL and these contracts work in alignment with that. If there will be a time to introduce a fee, we will replace those contracts with new ones.”

STI - Staking Token Issue

Criticality	medium
Location	contracts/DAIBonding.sol#102
Status	Acknowledged

Description

There are functions that are used to swap a specified amount of tokens. The fee or tax is an amount that is charged when tokens are swapped. According to the specification, the swapped amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

For instance, during the deposit phase, the contract mints USDR tokens in order to cover the amount that is going to be redeemed.

1. During the mint phase, if the USDR is not sufficient to cover the redeemed amount, the TNGBL is swapped.

2. The swapFromTNGBL method is used to swap TNGBL for USDR. The swapFromTNGBL applies fees but these fees are not deducted from the redeemed amount.

As a result, during the redeem phase, the user will not be able to claim the tokens since the actual USDR tokens of the contract will be less than the expected USDR amount. The expression uint256 remainingUSDR = v.usdr - amount; will revert.

```
function claim(uint256 amount) public {
        address who = msg.sender;
        uint256 maxAmount = claimable(msg.sender);
        //..
        uint256 totalClaimed = v.claimed + amount;
        USDRBonding(bonding).withdraw(amount, who);
        uint256 remainingUSDR = v.usdr - amount;
        //..
    }
```

Recommendation

The contract should take into consideration the corresponding fees in the total vested amount aggregation.

Team’s Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

“The contracts in question are there in marketing and incentive purposes and they are tailored to the current situation. When they become obsolete, we will replace them. For now, there is no fee on minting with TNGBL and these contracts work in alignment with that. If there will be a time to introduce a fee, we will replace those contracts with new ones.”

DMI - Defractionalize Manipulation Issue

Criticality	minor / informative
Location	contracts/USDTreasury.sol#325
Status	Acknowledged

Description

The nft defractionalize mechanism depends on the external function onERC721Received. The external function onERC721Received can be called by any user. Hence, the defractionalize mechanism could be manipulated.

```
if (
    (lastReceivedNFT == address(ftnft.tnft())) &&
    (lastReceivedTokenId == ftnft.tnftTokenId())
) {
    tracker.ftnftTreasuryPlaced(address(ftnft), tokenIds[0], false);
    tracker.tnftTreasuryPlaced(
        lastReceivedNFT,
        lastReceivedTokenId,
        true
    );
} else {
    tracker.updateFractionData(address(ftnft), tokenIds[0]);
}
```

Recommendation

The contract should take into consideration the public access of the onERC721Received() method.

Team’s Reply 14 December 2022

The team states:

“Since this is not crucial to treasury functioning, we will address this while migrating to V3.”

TBI - Token Balance Inconsistency

Criticality	minor / informative
Location	contracts/USDR.sol#61
Status	Acknowledged

Description

The contract balance is deleted when the account balance is equal to the transaction amount. On the contrary, if the transaction amount is greater than the account balance, then the balance is set to zero. As a result, it creates an inconsistency with the logic of the code. Since both statements fulfill the purpose.

```
        if (accountBalance == amount) {
            _totalSupply -= _balances[account];
            delete _balances[account];
        } else {
            uint256 amount_ = amount.wadToRay().rayDiv(liquidityIndex);
            if (amount_ > _balances[account]) {
                amount_ = _balances[account];
            }
            _totalSupply -= amount_;
            _balances[account] -= amount_;
        }
```

Recommendation

We state that both deleting and setting to zero produce the same result. But in terms of code readability and maintainability, the contract should manage the balances in the same manner.

Team’s Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

“This is not much of a problem, bytecode is the same, delete does the same thing as setting to 0.”

PRD - Pair Reserves Diversion

Criticality	minor / informative
Location	contracts/USDR.sol#95
Status	Acknowledged

Description

The contract balance could diverge in relation to the pair reserve balance. The pair addresses keep the reserve ratio proportionally to the balance of the token. The amount of the reserves is cached in the pair contract. So if the balance changes because of the rebase, then the pair should be synced manually.

```
    function rebase(uint256 supplyDelta) external onlyRole(CONTROLLER_ROLE)
        whenNotPaused
    {
        (address treasury, address exchange) = abi.decode(
            addressProvider.getAddresses(
                abi.encode(TREASURY_ADDRESS, USDR_EXCHANGE_ADDRESS)
            ),
            (address, address)
        );
        require(msg.sender == treasury, "caller is not treasury");
        uint256 ts = totalSupply();
        if (supplyDelta > 0) {
            supplyDelta =
IExchange(exchange).scaleFromUnderlying(supplyDelta);
            uint256 maxSupplyDelta = MAX_UINT128 - ts;
            if (supplyDelta > maxSupplyDelta) {
                supplyDelta = maxSupplyDelta;
            }
            if (supplyDelta > 0) {
                liquidityIndex = (liquidityIndex * (ts + supplyDelta)) / ts;
                int128[7] memory delta;
                delta[6] = int128(uint128(totalSupply() - ts));
                IExchange(exchange).updateMintingStats(delta);
            }
        }
        emit Rebase( block.number, block.timestamp / 1 days, ts, supplyDelta,
liquidityIndex);
    }
```

Recommendation

The contract should call the “sync” method from the pair contract every time the rebase mechanism is taking place.

Team’s Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

“Curve protocol doesn’t require syncing.”

TAZFA - Transferred Amount Zero Fees Assumption

Criticality	minor / informative
Location	contracts
Status	Acknowledged

Description

The contract performs token transactions without taking into account possible transaction fees. Additionally, the token properties are mutable. The contract owner has the authority to add any tokens. As a result, the actual transferred amount might diverge.

The following table depicts the inconsistency that may be produced

Function Calls	Fees	Transferred Amount	Assumed Transfer Amount
IERC20.transfer(user, 100);	0%	100	100
IERC20.transfer(user, 100);	10%	90	100

```
IERC20(underlying).transferFrom(msg.sender, address(this), amountIn);
IERC20(underlying).transfer(msg.sender, amount);
IERC20(underlying).transferFrom(onBehalfOf, address(this), deposit);
IERC20(underlying).transfer(treasury, amountIn);
IERC20(underlying).transferFrom( msg.sender, address(this), depositAmount)
IERC20(underlying).transferFrom( treasury, address(this), underlyingAmount);
IERC20(underlying).transferFrom( treasury, address(this), underlyingAmount);
..
..
```

Recommendation

The contract could take into consideration the underlying fees, or assert in case of the underlying token apply fees.

Team’s Reply 14 December 2022

The team states:

“We're not planning to use underlying tokens that incur transfer fees in those places when we do transfers, and we don’t have in plan to change any of that. And most of our contracts don’t hold any tokens except Treasury contracts.”

AIC - Arguments Inconsistency

Criticality	minor / informative
Location	contracts/TokenSwap.sol#L25 contracts/AffiliateExchange.sol#L129 contract/USDRTreasury.sol#L421
Status	Acknowledged

Description

The contract utilizes arguments that might diverge from the expected value.

The contract receives the `routerPathReverse` and the `routerPath` as arguments. Those arguments should be the opposite of each other but this is not guaranteed by the method's checks.

```
    function addSwapRoute(
        address router,
        address tokenInAddress,
        address tokenOutAddress,
        address[] calldata routerPath,
        address[] calldata routerPathReverse
    ) external onlyRole(ROUTER_POLICY_ROLE)
```

The contract receives the argument `total` as an aggregation of the `amounts`. The total arguments could be greater or lower than the aggregation of the amount array. Thus, the `total` argument could produce an inconsistency.

```
    function sendRewards(
        address batchSender,
        uint256 total,
        address[] calldata recipients,
        uint256[] calldata amounts
    ) external whenNotPaused onlyRole(CONTROLLER_ROLE
```

The contract is utilizing a `placed` argument on the function `updateTrackerTnft`. The `placed` argument determines if the NFT is minted to an account. The argument could produce inconsistency if it's true but the NFT already exists in the TreasureTracker.

```
    function updateTrackerTnft(
        address tnft,
        uint256 tokenId,
        bool placed
    ) internal
```

Recommendation

The contract could compose the `routerPathReverse` as the reserve path from `routerPath`.

The contract could calculate the `total` amount from the sum of the amounts.

The contract could incorporate the function ownerOf(tokenId) of the ERC721 interface or the internal structures instead of `placed` arguments.

Team’s Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

“addSwapRoute – only a config function and data provided are pairs on defi exchanges that we use, uniswap and curve so as long uniswap and curve don’t change anything we are ok. Because of the nature of the rewards and that their calculation is done off-chain – we made sure that we are the only ones who can send rewards. UpdateTrackerTnft is made so that it can be called from contract functions and none of those functions is without proper role.”

ELFM - Exceeds Fees Limit

Criticality	minor / informative
Location	contracts/USDRExchange.sol#L129
Status	Acknowledged

Description

The contract owner has the authority to increase the fees over the denominator value of 10000. This could be produced by calling the `setFees` function with a value greater than 10000.

```
    function setFees(uint256 depositFee_, uint256 withdrawalFee_)
        external
        onlyRole(DEFAULT_ADMIN_ROLE)
    {
        depositFee = depositFee_;
        withdrawalFee = withdrawalFee_;
    }
```

Recommendation

The contract could embody a check for the maximum acceptable value.

Team’s Reply 14 December 2022

The team states:

“Acknowledged, but right now it is not worth redeployment and we will address it in the future V3 release.”

DSM - Decimal Scale Missconsern

Criticality	minor / informative
Location	contracts/USDRExchange.sol#L274
Status	Acknowledged

Description

The function `_compoteScale` assumes that the first argument is always lower than the second argument. An unexpected value will be produced If the first argument is greater than the second argument. For instance,

Function Call	Result
_computeScale(1, 2)	10
_computeScale(2, 1)	340282366920938463463374607431768211456O

```
    function _computeScale(uint8 usdrDecimals, uint8 underlyingDecimals)
        private
        pure
        returns (uint256 scale)
    {
        assembly {
            switch lt(usdrDecimals, underlyingDecimals)
            case 0 {
                scale := shl(
                    128,
                    exp(10, sub(usdrDecimals, underlyingDecimals))
                )
            }
            default {
                scale := exp(10, sub(underlyingDecimals, usdrDecimals))
            }
        }
    }
```

Recommendation

The team is advised to carefully check if the implementation follows the expected business logic. The contract could embody a check for not allowing the first argument to be greater than the second one or could handle this case.

Team’s Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

“If we change the underlying to something with less than 9 decimals (e.g. USDC), computeScale will in fact return a huge number. This, however, is not unexpected. scale[To|From]Underlying rely on that behavior in order to decide whether amounts need to be scaled up or down.”

PIL - Potential Infinite Loop

Criticality	minor / informative
Location	contracts/PurchaseManager.sol#L147
Status	Acknowledged

Description

If the function “ITokenSwap(tokenSwap).quoteOut” returns a value that is lower than the amount then an infinite loop may be produced.

```
do {
    reserveAmount = ITokenSwap(tokenSwap).quoteOut(
        underlying,
        address(paymentToken),
        calcAmount
    );

    if (reserveAmount < amount) {
        calcAmount =
            calcAmount +
            _convertToCorrectDecimals(
                reserveAmount - amount,
                paymentDecimals,
                underlyingDecimals
            ) +
            10**uint256(underlyingDecimals); // add 1 dollar
    }
} while (reserveAmount < amount);
```

Recommendation

It is recommended to add a limitation in the potential number of loops due to the fact that the loops may escalate. As a result, the loop could easily exceed the maximum block size and make the contract run out of gas.

Team’s Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

“This was implemented that way intentionally. It is only used when calculating stable to stable because unfortunately, curve contract doesn’t have a way to provide input with exact output, so this was our only gas-optimized option.”

STC - Succeed Transfer Check

Criticality	minor / informative
Location	contracts/AffiliateExchange.sol#L68,79 contract/DAIBond.sol#L68,79 contracts/IncentiveVault.sol#36,52 contract/USDRBonding.sol#44,51,110,119,122,145 contracts/USDRExchange.sol#230 contract/USDRTreasury.sol#126 contracts/LiquidityManager.sol#109,213,231 contract/TNGBLLiquidityManager.sol#35,79,101,115,165,166,249,265 contracts/USDRMigration.sol#67
Status	Acknowledged

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(usdr).transfer(msg.sender, excessUSDR);
erc20.transfer(msg.sender, balance);

IERC20(token).transfer(msg.sender, balance);

IERC20(token).transfer(msg.sender, balance);
IERC20(underlying).transfer(msg.sender, amount);

IERC20(token).transfer(receiver, amount);
IERC20(token).transfer(_owner, amount);
IERC20(underlying).transferFrom(onBehalfOf, address(this), deposit);
```

Recommendation

The contract should check if the result of the transfer methods is successful.

Team’s Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

“Acknowledged, but not critical at this point. We're not using tokens, that don't follow the standard (like USDT).”

CO - Code Optimization

Criticality	minor / informative
Location	contracts/USDRBonding.sol#L57,155 contracts/USDR.sol#L95 contracts/RWACalculator.sol#L451 contract/TokenSwap.sol#L68
Status	Acknowledged

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The variable `percentage` and the function `setPercentage` are not utilized in contract implementation. Hence, they are redundant.

```
uint16 public percentage;
function setPercentage(uint16 value) external onlyRole(DEFAULT_ADMIN_ROLE) {
        percentage = value;
}
```

The `fetchPaymentTokenAndAmountTnft` method is using an entire code segment that is not used anywhere.

```
HelperStruct memory hs;
hs.weSellAt = new uint256[](1);
hs.lockedAmount = new uint256[](1);
hs.tokenIds = new uint256[](1);
hs.tokenIds[0] = tokenId;
ITNFTPriceManager priceManager = _getPriceManager();
(hs.weSellAt, , , , hs.lockedAmount) = priceManager
    .itemPriceBatchTokenIds(
```

```
        ITangibleNFT(tnft),
        paymentToken,
        hs.tokenIds
    );
```

The internal function `_verifyBacking` is utilized with constant arguments. As a result, the function arguments are redundant.

```
function _verifyBacking(uint8 threshold, bool includeTNGBL) internal view {
        address usdr = _fetchAddress(USDR_ADDRESS);
        ITreasury.TreasuryValue memory tv = getTreasuryValue();
        uint256 scaledMarketCap = IERC20(usdr).totalSupply() * 1e9;
        uint256 backing = tv.total;
        if (!includeTNGBL) {
            backing = backing - tv.tngbl - tv.tngblLiquidity.tngbl;
        }
        require(
            (scaledMarketCap * threshold) / 100 <= backing,
            "insufficient backing"
        );
    }
```

The `exchange` function doesn't check if the corresponding path exists on the `swappers` mapping, the caller should be aware of this instead of a generic revert error.

```
function exchange(
        address tokenIn,
        address tokenOut,
        uint256 amountIn,
        uint256 amountOut,
        EXCHANGE_TYPE exchangeType
    ) external override returns (uint256) {
        bytes memory tokenized = abi.encodePacked(tokenIn, tokenOut);
        address[] memory path = swappers[tokenized].path;
```

The `exchange` function performs an extra transaction. The contract initially transfers the exchanged amount to the contract and then from the contract to the msg.sender.

```
if (exchangeType == EXCHANGE_TYPE.EXACT_INPUT) {
            amounts = IUniswapV2Router01(swappers[tokenized].router)
                .swapExactTokensForTokens(
                    amountIn,
                    amountOut,
                    path,
                    address(this),
                    block.timestamp + 30 // on sushi?
                );
} else if (exchangeType == EXCHANGE_TYPE.EXACT_OUTPUT) {
            amounts = IUniswapV2Router01(swappers[tokenized].router)
```

Recommendation

The authors are advised to rewrite some code segments so the runtime will be more performant.

The contract could remove redundant arguments and functions.

The contract could have an informative message that the pair does not exist.

The contract could swap the exchanged amount directly to the `msg.sender`.

Team’s Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

“Dead code just increases the size of the contract and is only concerned in deployment. The end user doesn’t have any inconvenience about this. We will address this in future version V3. The Exchange contract is a helper contract where we wrapped Uniswap v3 and Curve and we only use a subset of tokens that we need, so we save space and gas needed thereby making sure our protocol uses only what we need.”

L04 - Conformance to Solidity Naming Conventions

Criticality	minor / informative
Location	contracts/AddressAccessor.sol#L24,12 contracts/interfaces/ITokenSwap.sol#L5 contracts/USDRExchange.sol#L240 contracts/USDRTreasury.sol#L75,478,76,78,477,77 contracts/RWACalculator.sol#L407,148
Status	Acknowledged

Description

Solidity defines a naming convention that should be followed. Rule exceptions:

- Allow constant variable name/symbol/decimals to be lowercase.
- Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

```
_addressProvider
EXCHANGE_TYPE
_purchaseStableMintedRedeemedThreshold
_tokenIds
_purchaseStableMarketcapThreshold
_incentiveThreshold
_nft
_tngblThreshold
_years
...
```

Recommendation

Follow the Solidity naming convention.
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-conventions.

Team's Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

"Naming is not too big of a problem, different team members have different coding styles, we will address it in possible future migrations."

L09 - Dead Code Elimination

Criticality	minor / informative
Location	contracts/RWACalculator.sol#L163
Status	Acknowledged

Description

Functions that are not used in the contract, and make the code's size bigger.

```
_isGoldTnft
```

Recommendation

Remove unused functions.

Team’s Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

“Code size only matters on deployment, and we use Polygon where gas prices are very cheap.”

L11 - Unnecessary Boolean equality

Criticality	minor / informative
Location	contracts/USDRTreasury.sol#L86
Status	Acknowledged

Description

The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
success == false
```

Recommendation

Remove the equality to the boolean constant.

Team’s Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

“We chose this because of readability.”

L12 - Using Variables before Declaration

Criticality	minor / informative
Location	contract/USDRTreasury.sol#L131 contract/TNGBLPriceOracle.sol#L47
Status	Acknowledged

Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or the variable has been declared in a different scope.

```
success
amount
```

Recommendation

The variables should be declared before any usage of them.

Team’s Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

“We are just exploiting language abilities to save on gas.”

L13 - Divide before Multiply Operation

Criticality	minor / informative
Location	contract/IncentiveVault.sol#L55 contract/RWACalculator.sol#L352
Status	Acknowledged

Description

Performing divisions before multiplications may cause loss of prediction.

```
amount = (IExchange(exchange).scaleToUnderlying(IERC20(USDR).totalSupply()) * apr) /
3_650_000
topPrice = ((((hs.weSellAt[0] + hs.lockedAmount[0]) * share) / 10000000) *
priceThreshold[address(tnft)]) / fullPercent
```

Recommendation

The multiplications should be prior to the divisions.

Team’s Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

“First example doesn’t have precision loss, and the second example is used only by our scripts and contracts where we make decisions on fraction purchases, where small precision losses are not significant in our use-case.”

L14 - Uninitialized Variables in Local Scope

Criticality	minor / informative
Location	contract/TreasuryTracker.sol#L155,92,509,248 contract/TokenSwap.sol#L82 contract/USDRTreasury.sol#L131,221,375,494 contract/TNGBLPriceOracle.sol#L47 contract/RWACalculator.sol#L331,451,285,372,276,322,308,326,421,280,262,164
Status	Acknowledged

Description

The are variables that are defined in the local scope and are not initialized.

```
i
fData
amounts
success
ah
amount
j
hs_scope_0
hs
...
```

Recommendation

All the local scoped variables should be initialized.

Team’s Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

“Those are mostly helper structs that we need because of issues with stack having too many local variables. We rely on Solidity here, that it initializes variables dependent on variable types https://docs.soliditylang.org/en/v0.8.17/types.html.”

L15 - Local Scope Variable Shadowing

Criticality	minor / informative
Location	contract/tangibleInterfaces/ITangibleInterfaces.sol#L34 contract/interfaces/ILiquidityManager.sol#L15
Status	Acknowledged

Description

The are variables that are defined in the local scope containing the same name from an upper scope.

```
fullShare
liquidity
```

Recommendation

The local variables should have different names from the upper scoped variables.

Team’s Reply 14 December 2022

The team has acknowledged that this is not a security issue and states:

“That has no effect on interface declarations”

Contract Functions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
AddressAccessor	Implementation	AccessControl		
	setAddressProvider	Public	✓	onlyRole
AddressAccessorUpgradable	Implementation	AccessControlUpgradeable		
	setAddressProvider	Public	✓	onlyRole
AddressProvider	Implementation	OwnableUpgradeable		
	initialize	Public	✓	initializer
	setAddress	External	✓	onlyOwner
	getAddresses	External		-
IBatchSender	Interface			
	send	External	✓	-
IExchangeProxy	Interface			
	swapFromToken	External	✓	-
AffiliateExchange	Implementation	Pausable, AddressAccessor		
	<Constructor>	Public	✓	-
	mint	External	✓	whenNotPaused
	pause	External	✓	whenNotPaused onlyRole
	unpause	External	✓	whenPaused onlyRole

	sendRewards	External	✓	whenNotPaused onlyRole
	withdrawExcessUSDR	External	✓	onlyRole
	withdrawToken	External	✓	whenPaused onlyRole
	_mint	Private	✓	
DAIBond	Implementation	AddressAccessor, Pausable		
	<Constructor>	Public	✓	-
	claimAll	External	✓	-
	deposit	External	✓	whenNotPaused
	earned	External		-
	pause	External	✓	onlyRole whenNotPaused
	recoverLostTokens	External	✓	onlyRole
	unpause	External	✓	onlyRole whenPaused
	claimable	Public		-
	claim	Public	✓	-
	_earned	Private		
CurveExchanges	Interface			
	underlying_coins	External	✓	-
	get_exchange_amount	External		-
	get_input_amount	External		-
	exchange	External	✓	-
CurveRegistry	Interface			
	find_pool_for_coins	External		-
	get_coin_indices	External		-
CurveWrapper	Implementation	AccessControl		

	<Constructor>	Public	✓	-
	addPoolForTokens	External	✓	onlyRole
	getAmountsIn	External		-
	getAmountsOut	External		-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
IncentiveVault	Implementation	AddressAccessor, Pausable		
	<Constructor>	Public	✓	-
	pause	External	✓	onlyRole
	recoverLostTokens	External	✓	onlyRole
	setAPR	External	✓	onlyRole
	unpause	External	✓	onlyRole
	withdraw	External	✓	onlyRole
	availableAmount	Public		-
IExchange	Interface			
	scaleFromUnderlying	External		-
	scaleToUnderlying	External		-
	swapFromUnderlying	External	✓	-
	updateMintingStats	External	✓	-
ILiquidityManager	Interface			
	getTokenAmounts	External		-
	liquidity	External		-
	increaseLiquidity	External	✓	-
ILiquidityTokenMath	Interface			
	getTokenAmounts	External		-
IPriceOracle	Interface			
	quote	External		-

IRWACalculator	Interface			
	calculate	External		-
	fetchPaymentTokenAndAmountFtnft	External		-
	fetchPaymentTokenAndAmountTnft	External		-
	calcFractionNativeValue	External		-
	calcTnftNativeValue	External		-
ITokenSwap	Interface			
	quoteOut	External		-
	quoteIn	External		-
	exchange	External	✓	-
ITreasury	Interface			
	purchaseStableMintedRedeemedThreshold	External		-
	purchaseStableMarketcapThreshold	External		-
	multicall	External	✓	-
	withdraw	External	✓	-
	getTreasuryValue	External		-
	updateTrackerFtnftExt	External	✓	-
	updateTrackerTnftExt	External	✓	-
	purchaseReInitialSale	External	✓	-
ITreasuryTracker	Interface			
	tnftTreasuryPlaced	External	✓	-
	ftnftTreasuryPlaced	External	✓	-
	updateFractionData	External	✓	-
	getFractionTokensDataInTreasury	External		-
	getRwaUsdValue	External		-
	addValueAfterPurchase	External	✓	-
	subValueAfterPurchase	External	✓	-

IUSDR	Interface	IERC20Upgradeable		
	burn	External	✓	-
	mint	External	✓	-
	rebase	External	✓	-
GoldPurchaseManager	Implementation	PurchaseManager, IERC721Receiver		
	<Constructor>	Public	✓	-
	setGoldTnfts	External	✓	onlyRole
	purchaseTnft	External	✓	onlyRole
	purchaseTnftCb	External	✓	-
	purchaseFtnft	External	✓	onlyRole
	purchaseFtnftCb	External	✓	-
	_fetchGoldAddressFromFraction	Internal		
	onERC721Received	External	✓	-
	_onERC721Received	Private	✓	
GoldSellManager	Implementation	OnSaleTracker, IERC721Receiver		
	<Constructor>	Public	✓	-
	setGoldTnfts	External	✓	onlyRole
	sellTnft	External	✓	onlyRole
	sellTnftCb	External	✓	-
	modifyTnftSale	External	✓	onlyRole
	sellFtnft	External	✓	onlyRole
	sellFtnftCb	External	✓	-
	modifyFtnftSale	External	✓	onlyRole
	sellFtnftInitial	External	✓	onlyRole
	sellFtnftInitialCb	External	✓	-
	_fetchGoldAddressFromFraction	Internal		
	stopSellFtnft	External	✓	onlyRole
	stopSellTnft	External	✓	onlyRole

	withdrawToken	External	✓	-
	onERC721Received	External	✓	-
	_onERC721Received	Private	✓	
OnSaleTracker	Implementation	AddressAccessor		
	getFractionContractsOnSale	External		-
	fractionContractsOnSaleSize	External		-
	getFractionTokensOnSale	External		-
	getFractionTokensOnSaleBatch	External		-
	fractionTokensOnSaleSize	External		-
	getTnftCategoriesOnSale	External		-
	tnftCategoriesOnSaleSize	External		-
	getTnftTokensOnSale	External		-
	getTnftTokensOnSaleBatch	External		-
	tnftTokensOnSaleSize	External		-
	tnftSalePlacedExt	External	✓	onlyRole
	tnftSalePlaced	Internal	✓	
	ftnftSalePlacedExt	External	✓	onlyRole
	ftnftSalePlaced	Internal	✓	
	_removeCurrentlySellingFraction	Internal	✓	
	_removeCurrentlySellingTnft	Internal	✓	
	_removeCategory	Internal	✓	
	_removeFraction	Internal	✓	
IUSDRExchange	Interface			
	mintingStats	External		-
IUSDR	Interface	IERC20Upgradeable		
	totalSupply	External		-
PurchaseManager	Implementation	AddressAccessor		
	_validatePurchase	Internal		

	_convertTreasuryTokenToPayment	Internal	✓	
	_convertToCorrectDecimals	Internal		
	_checkPaymentTokenAndAmountNeeded	Internal		
RePurchaseManager	Implementation	PurchaseManager, IERC721Receiver		
	<Constructor>	Public	✓	-
	setRETnft	External	✓	onlyRole
	purchaseTnft	External	✓	onlyRole
	purchaseTnftCb	External	✓	-
	purchaseFtnft	External	✓	onlyRole
	purchaseFtnftCb	External	✓	-
	_fetchReAddressFromFraction	Internal		
	onERC721Received	External	✓	-
	_onERC721Received	Private	✓	
ReSellManager	Implementation	OnSaleTracker, IERC721Receiver		
	<Constructor>	Public	✓	-
	setRETnft	External	✓	onlyRole
	sellTnft	External	✓	onlyRole
	sellTnftCb	External	✓	-
	modifyTnftSale	External	✓	onlyRole
	sellFtnft	External	✓	onlyRole
	sellFtnftCb	External	✓	-
	modifyFtnftSale	External	✓	onlyRole
	sellFtnftInitial	External	✓	onlyRole
	sellFtnftInitialCb	External	✓	-
	stopSellFtnft	External	✓	onlyRole
	stopSellTnft	External	✓	onlyRole
	_fetchReAddressFromFraction	Internal		
	withdrawToken	External	✓	-

	onERC721Received	External	✓	-
	_onERC721Received	Private	✓	
Owned	Implementation			
	<Constructor>	Public	✓	-
	transferOwnership	External	✓	onlyOwner
	acceptOwnership	External	✓	-
AggregatorInterface	Interface			
	latestAnswer	External		-
	latestTimestamp	External		-
	latestRound	External		-
	getAnswer	External		-
	getTimestamp	External		-
AggregatorV3Interface	Interface			
	decimals	External		-
	description	External		-
	version	External		-
	getRoundData	External		-
	latestRoundData	External		-
AggregatorV2V3Interface	Interface	AggregatorInterface, AggregatorV3Interface		
MockDAIUSDOracle	Implementation	AggregatorV2V3Interface, Owned		
	latestAnswer	Public		-
	latestRoundData	Public		-
	decimals	External		-
	version	External		-
	description	External		-

	getAnswer	External		-
	getTimestamp	External		-
	latestRound	External		-
	latestTimestamp	External		-
	getRoundData	External		-
MockERC20	Implementation	ERC20, AccessControl		
	<Constructor>	Public	✓	ERC20
	decimals	Public		-
	mint	External	✓	whitelisted
	transfer	Public	✓	whitelisted
	transferFrom	Public	✓	whitelisted
	toggleWhitelist	External	✓	onlyRole
	whitelist	External	✓	onlyRole
MockOracle	Implementation			
	consult	External		-
MockRouter	Implementation			
	getAmountsOut	External		-
MockTreasury	Implementation	ITreasury		
	withdraw	External	✓	-
	setStableValue	External	✓	-
	setUSDRValue	External	✓	-
	setRWAValue	External	✓	-
	setTNGBLValue	External	✓	-
	setLiquidityValue	External	✓	-
	setTNGBLLiquidityValue	External	✓	-
	setDebtValue	External	✓	-
	getTreasuryValue	External		-
	multicall	External	✓	-
	updateTrackerFtnftExt	External	✓	-

	updateTrackerTnftExt	External	✓	-
	purchaseReInitialSale	External	✓	-
	purchaseStableMintedRedeemedThreshold	External		-
	purchaseStableMarketcapThreshold	External		-
	_recomputeTotal	Private	✓	
ITreasuryTrackerExt	Interface	ITreasuryTracker		
	getFractionContractsInTreasury	External		-
	fractionContractsInTreasurySize	External		-
	getFractionTokensInTreasury	External		-
	fractionTokensInTreasurySize	External		-
	getFractionContractsOnSale	External		-
	fractionContractsOnSaleSize	External		-
	getFractionTokensOnSale	External		-
	fractionTokensOnSaleSize	External		-
	getTnftCategoriesInTreasury	External		-
	tnftCategoriesInTreasurySize	External		-
	getTnftTokensInTreasury	External		-
	tnftTokensInTreasurySize	External		-
	getTnftCategoriesOnSale	External		-
	tnftCategoriesOnSaleSize	External		-
	getTnftTokensOnSale	External		-
	tnftTokensOnSaleSize	External		-
ITNFTPriceManager	Interface			
	itemPriceBatchTokenIds	External		-
	itemPriceBatchFingerprints	External		-
	getPriceOracleForCategory	External		-
RWACalculator	Implementation	AddressAccessor, IRWACalculator		
	<Constructor>	Public	✓	-

	setGoldTnfts	External	✓	onlyRole
	setPriceAboveMarketThreshold	External	✓	onlyRole
	_isGoldTnft	Internal		
	_getPriceManager	Internal		
	_getTangibleMarketplace	Internal		
	_convertToCorrectDecimals	Internal		
	calculate	External		-
	_calculateTnftsOnSale	Internal		
	_calculateFtnftsOnSale	Internal		
	fetchPaymentTokenAndAmountFtnft	External		-
	fetchPaymentTokenAndAmountTnft	External		-
	calcFractionNativeValue	External		-
	calcTnftNativeValue	External		-
	_getTnftNativeValue	Internal		
	_checkStorageValue	Internal		
IStrategy	Interface			
	treasury	External		-
	want	External		-
	deposit	External	✓	-
	withdraw	External	✓	-
	balanceOf	External		-
	balanceOfWant	External		-
	balanceOfPool	External		-
	harvest	External	✓	-
	retire	External	✓	-
	panic	External	✓	-
	router	External		-
IMiniChefV2	Interface			
	poolLength	External		-
	updatePool	External	✓	-
	userInfo	External		-
	deposit	External	✓	-
	withdraw	External	✓	-

	harvest	External	✓	-
	withdrawAndHarvest	External	✓	-
	emergencyWithdraw	External	✓	-
SushiLPStrategy	Implementation	AccessControl, Pausable, IStrategy		
	<Constructor>	Public	✓	-
	deposit	Public	✓	whenNotPaused
	withdraw	External	✓	onlyTreasury
	harvest	External	✓	whenNotPaused onlyRole
	addLiquidity	Internal	✓	
	balanceOf	Public		-
	balanceOfWant	Public		-
	balanceOfPool	Public		-
	retire	External	✓	onlyTreasury
	panic	Public	✓	onlyRole
	pause	Public	✓	onlyRole
	unpause	External	✓	onlyRole
	_ensureAllowance	Internal	✓	
	_removeAllowances	Internal	✓	
ICurrencyFeed	Interface			
	currencyPriceFeeds	External		-
	conversionPremiums	External		-
IInstantLiquidity	Interface			
	sellInstant	External	✓	-
	buyInstant	External	✓	-
	sellInstantFraction	External	✓	-
	buyFractionInstant	External	✓	-
	withdrawUSDC	External	✓	-
	withdrawTNGBL	External	✓	-

IPriceOracle	Interface			
	latestPrices	External		-
	decimals	External		-
	marketPriceNativeCurrency	External		-
ITangibleNFT	Interface			
	storagePricePerYear	External		-
	storagePercentagePricePerYear	External		-
	storagePriceFixed	External		-
	storageRequired	External		-
	tnftToPassiveNft	External		-
	claim	External	✓	-
	tokensFingerprint	External		-
ITangibleFractionsNFT	Interface			
	defractionalize	External	✓	-
	tnft	External		-
	tnftTokenId	External		-
	tnftFingerprint	External		-
	fractionShares	External		-
	fullShare	External		-
	claim	External	✓	-
	claimableIncome	External		-
IFractionStorageManager	Interface			
	payShareStorage	External	✓	-
IFactoryExt	Interface			
	storageManagers	External		-
	defUSD	External		-
	paymentTokens	External		-
	fractionToTnftAndId	External		-
	initReSeller	External		-

ITangibleMarketplace	Interface			
	marketplace	External		-
	marketplaceFract	External		-
	factory	External		-
	sellBatch	External	✓	-
	stopBatchSale	External	✓	-
	buy	External	✓	-
	buyUnminted	External	✓	-
	buyFraction	External	✓	-
	sellFraction	External	✓	-
	payStorage	External	✓	-
	sellFractionInitial	External	✓	-
	stopFractSale	External	✓	-
ITangiblePiNFT	Interface			
	claim	External	✓	-
	claimableIncome	External		-
ITangibleRentShare	Interface			
	forToken	External	✓	-
ITangibleRevenueShare	Interface			
	claimForToken	External	✓	-
	revenueToken	External		-
TNGBLLockedValue	Implementation	AddressAccessor		
	<Constructor>	Public	✓	-
	getTngblLockedValue	External		-
IBaseOracle	Interface			

	consult	External		-
TNGBLPriceOracle	Implementation	AddressAccessor, IPriceOracle		
	<Constructor>	Public	✓	-
	quote	External		-
	setOracleLookBackPeriod	External	✓	onlyRole
	setSwapRoute	Public	✓	onlyRole
IMintableERC20	Interface			
	mint	External	✓	-
ITangibleERC20	Interface			
	approve	External	✓	-
	burn	External	✓	-
pDAI	Implementation	AddressAccessor, ERC20Permit, Pausable		
	<Constructor>	Public	✓	ERC20 ERC20Permit
	mint	External	✓	onlyRole whenNotPaused
	pause	External	✓	onlyRole
	redeem	External	✓	-
	redeemFor	Public	✓	whenNotPaused
	unpause	External	✓	onlyRole

USDR	Implementation	IUSDR, AddressAccessorUpgradable, ERC20PermitUpgradeable, PausableUpgradeable		
	initialize	Public	✓	initializer
	burn	External	✓	whenNotPaused
	mint	External	✓	onlyRole whenNotPaused
	pause	External	✓	onlyRole whenNotPaused
	rebase	External	✓	onlyRole whenNotPaused
	unpause	External	✓	onlyRole whenPaused
	allowance	Public		-
	approve	Public	✓	-
	balanceOf	Public		-
	decimals	Public		-
	decreaseAllowance	Public	✓	-
	increaseAllowance	Public	✓	-
	totalSupply	Public		-
	transfer	Public	✓	whenNotPaused
	transferAll	Public	✓	whenNotPaused
	transferAllFrom	Public	✓	whenNotPaused
	transferFrom	Public	✓	whenNotPaused
	_approve	Internal	✓	
IStaking	Interface			
	unstake	External	✓	-

	usdrMarketCap	External		-
LegacySUSDR	Interface			
	transferAll	External	✓	-
	transferAllFrom	External	✓	-
USDRMigration	Implementation	AddressAccessor		
	<Constructor>	Public	✓	-
	initialize	External	✓	onlyRole
	migrate	External	✓	-
WadRayMath	Library			
	rayDiv	Internal		
	rayMul	Internal		
	rayToWad	Internal		
	wadToRay	Internal		
TokenSwap	Implementation	ITokenSwap, AccessControl		
	<Constructor>	Public	✓	-
	addSwapRoute	External	✓	onlyRole
	removeSwapRoute	External	✓	onlyRole
	exchange	External	✓	-
	quoteOut	External		-
	quoteIn	External		-
TreasuryTracker	Implementation	AddressAccessor, ITreasuryTracker		
	getFractionContractsInTreasury	External		-
	getTnftFractionContractsInTreasury	External		-
	fractionContractsInTreasurySize	External		-
	tnftFractionContractsInTreasurySize	External		-

	getFractionTokensInTreasury	External		-
	getFractionTokensInTreasuryBatch	External		-
	fractionTokensInTreasurySize	External		-
	getFractionTokensDataInTreasury	External		-
	getTnftCategoriesInTreasury	External		-
	tnftCategoriesInTreasurySize	External		-
	getTnftTokensInTreasury	External		-
	getTnftTokensInTreasuryBatch	External		-
	tnftTokensInTreasurySize	External		-
	<Constructor>	Public	✓	-
	tnftTreasuryPlaced	External	✓	-
	ftnftTreasuryPlaced	External	✓	-
	_fetchTnftForFraction	Internal		
	_removeCurrentlySellingFraction	Internal	✓	
	updateFractionData	External	✓	-
	_removeCurrentlySellingTnft	Internal	✓	
	_removeCategory	Internal	✓	
	_removeFraction	Internal	✓	
	_removeFractionFromTnftMap	Internal	✓	
	setCurrencyData	External	✓	onlyRole
	updateTotalNativeValue	External	✓	onlyRole
	addValueAfterPurchase	External	✓	-
	_convertToCorrectDecimals	Internal		
	subValueAfterPurchase	External	✓	-
	removeCurrency	External	✓	onlyRole
	currencySize	External		-
	getRwaUsdValue	External		-
	convertPriceToUSDCustom	Internal		
IExchange	Interface			
	scaleFromUnderlying	External		-
	swapFromUnderlying	External	✓	-
	swapFromTNGBL	External	✓	-
BondingVault	Implementation			

	<Constructor>	Public	✓	-
	withdraw	External	✓	-
	sweep	External	✓	-
USDRBonding	Implementation	AddressAccessor		
	<Constructor>	Public	✓	-
	sweepVault	External	✓	onlyRole
	mint	External	✓	onlyRole
	recoverLostTokens	External	✓	onlyRole
	reset	External	✓	onlyRole
	setPercentage	External	✓	onlyRole
	withdraw	External	✓	onlyRole
USDRExchange	Implementation	AddressAccessor, IExchange, Pausable		
	<Constructor>	Public	✓	-
	pause	External	✓	onlyRole
	scaleFromUnderlying	External		-
	updateMintingStats	External	✓	onlyRole
	maxTNGBLMintingAmount	External		-
	mintAgainstGains	External	✓	onlyRole
	scaleToUnderlying	External		-
	setFees	External	✓	onlyRole
	swapToPromissory	External	✓	whenNotPaused
	swapToTNGBL	External	✓	whenNotPaused
	swapToUnderlying	External	✓	whenNotPaused
	swapFromTNGBL	External	✓	whenNotPaused
	swapFromUnderlying	External	✓	whenNotPaused
	unpause	External	✓	onlyRole
	setAddressProvider	Public	✓	onlyRole

	updateUnderlying	Public	✓	onlyRole
	_applyFee	Private		
	_computeScale	Private		
	_maxTNGBLMintingAmount	Private		
	_prepareTNGBLWithdrawal	Private	✓	
	_preparePromissoryWithdrawal	Private	✓	
	_prepareUnderlyingWithdrawal	Private	✓	
	_scaleAmount	Private		
	_scaleFromUnderlying	Private		
	_scaleToUnderlying	Private		
USDRExchangeProxy	Implementation	AddressAccessor, Pausable		
	<Constructor>	Public	✓	-
	pause	External	✓	onlyRole
	swapFromToken	External	✓	whenNotPaused
	unpause	External	✓	onlyRole
ITreasuryTrackerExt	Interface	ITreasuryTracker		
	getFractionContractsInTreasury	External		-
	getFractionTokensInTreasury	External		-
	getTnftCategoriesInTreasury	External		-
	getTnftTokensInTreasury	External		-
USDRTreasury	Implementation	AddressAccessor, ITreasury, IERC721Receiver		
	<Constructor>	Public	✓	-
	setThresholds	External	✓	onlyRole
	multicall	External	✓	onlyRole
	triggerRebase	External	✓	onlyRole
	purchaseReInitialSale	External	✓	-
	_swapToTreasuryToken	Internal	✓	

	getTreasuryValue	Public		-
	withdraw	External	✓	validToken
	defractionalize	External	✓	onlyRole
	updateTrackerFtnftExt	External	✓	onlyRole
	updateTrackerFtnft	Internal	✓	
	updateTrackerTnftExt	External	✓	onlyRole
	updateTrackerTnft	Internal	✓	
	toggleStop	External	✓	onlyRole
	withdrawToken	External	✓	onlyRole
	withdrawDepositNFT	External	✓	onlyRole
	_withdrawDepositNFT	Internal	✓	
	claimRentForToken	External	✓	onlyRole
	claimTngblRevenue	External	✓	-
	payFractionStorage	External	✓	onlyRole
	onERC721Received	External	✓	-
	_fetchAddress	Internal		
	_verifyBacking	Internal		

Contract Flow

Summary

The Tangible USDR Ecosystem implements a stable coin mechanism. This audit focused on investigating possible security issues and potential improvements. It investigates the diversion from a classic DAO implementation and comments about the architectural decisions.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. 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 Cyberscope's prior written consent. 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 regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document 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 represents an extensive assessment 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 Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope'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 Cyberscope are subject to dependencies and are 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.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io