OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 92	2021-09-02	Zapmore, Donut	Audit Draft

Audit Notes

Audit Date	2021-07-01 - 2021-08-31
Auditor/Auditors	Donut, Plemonade, MrTeaThyme
Auditor/Auditors Contact Information	tibereum-obelisk@protonmail.com
Notes	Specified code and contracts are audited for security flaws. UI/UX (website), logic, team, and tokenomics are not audited.
Audit Report Number	OB581111287

Disclaimer

This audit is not financial, investment, or any other kind of advice and is for informational purposes only. This report is not a substitute for doing your own research and due diligence. Obelisk is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Obelisk has based this audit report solely on the information provided by the audited party and on facts that existed before or during the audit being conducted. Obelisk is not responsible for any outcome, including changes done to the contract/contracts after the audit was published. This audit is fully objective and only discerns what the contract is saying without adding any opinion to it. The audit is paid by the project but neither the auditors nor Obelisk has any other connection to the project and has no obligations other than to publish an objective report. Obelisk will always publish its findings regardless of the outcome of the findings. The audit only covers the subject areas detailed in this report and unless specifically stated, nothing else has been audited. Obelisk assumes that the provided information and material were not altered, suppressed, or misleading. This report is published by Obelisk, and Obelisk has sole ownership of this report. Use of this report for any reason other than for informational purposes on the subjects reviewed in this report including the use of any part of this report is prohibited without the express written consent of Obelisk.

Obelisk Auditing

Defi is a relatively new concept but has seen exponential growth to a point where there is a multitude of new projects created every day. In a fast-paced world like this, there will also be an enormous amount of scams. The scams have become so elaborate that it's hard for the common investor to trust a project, even though it could be legit. We saw a need for creating high-quality audits at a fast phase to keep up with the constantly expanding market. With the Obelisk stamp of approval, a legitimate project can easily grow its user base exponentially in a world where trust means everything. Obelisk Auditing consists of a group of security experts that specialize in security and structural operations, with previous work experience from among other things, PricewaterhouseCoopers. All our audits will always be conducted by at least two independent auditors for maximum security and professionalism.

As a comprehensive security firm, Obelisk provides all kinds of audits and project assistance.

Table of Contents

Project Information

Project Name	Argano
Description	Argano is a decentralized protocol and liquidity aggregator which is built and supported on the Polygon sidechain network. The Argano protocol aims to offer the best services of the decentralized exchange with obviously new and innovative features in the DeFi space.
Website	https://www.argano.io/
Contact	@Beka_ulis
Contact information	@Beka_ulis on TG
Token Name(s)	Argano Argano Bitcoin token Argano Dollar token Catena Bitcoin share token Catena Dollar share token
Token Short(s)	AGO AGOBTC AGOUSD CNBTC CNUSD
Contract(s)	See Appendix A
Code Language	Solidity
Chain	Polygon

Executive Summary

The audit of Argano was conducted by three of Obelisks' security experts between the 1st of July 2021 and the 31th of August 2021.

After finishing the full audit, there were multiple issues found at first, but the project team took in the feedback given and resolved most issues found. They where very open to suggestions and keen on getting everything right.

Issue #13, #21 & #46 all relates to the deployment which will be checked on after deployment and this audit will be updated if those are satisfactorily resolved. Issue #42 is behind a timelock. All other security issues of relevant risk levels are resolved.

Other Informational findings are there for informational purposes and don't impact the project on a larger scale on the audited implementation.

The team has not reviewed the UI/UX, logic, team, or tokenomics of the Argano project.

Please read the full document for a complete understanding of the audit.

Summary Table

Audited Part	ID	Severity	Mitigated
Migration Function	#0001	High Risk	Resolved
Unrestricted Proxy Call In Treasury	#0002	High Risk	Resolved
CNUSD Dev Vesting Duration Can Be Initialized To The Past	#0003	Medium Risk	Resolved
CNUSD Dev Vesting Duration Is 1 Day	#0004	Medium Risk	Resolved
Pool Can Be Unilaterally Paused	#0005	Medium Risk	Resolved
Oracle Prices Can Be Stale	#0006	Medium Risk	Resolved
Phishing Discount Fee	#0007	Medium Risk	Resolved
Redemption Can Be Toggled Between Target And Effective Collateralization	#0008	Medium Risk	Resolved
Rebalance Pool Can Be Removed	#0009	Medium Risk	Resolved
Migration Not Checked In Treasury	#0010	Medium Risk	Resolved
Duplicate Implementation Of Burn From	#0011	Low Risk	Resolved
Oracle Precision Mismatch	#0012	Low Risk	Resolved
Start Time Can Be Initialized To Past	#0013	Low Risk	See Comment
Swap Path Uses Pairs Instead Of Tokens	#0014	Low Risk	Resolved
Local Copies Of OpenZeppelin Contracts	#0015	Informational	Resolved
Local Copies Of Chainlink Contracts	#0016	Informational	Resolved
Local Copies Of Uniswap Libraries	#0017	Informational	Resolved
No Events Emitted For Important	#0018	Informational	Resolved

Function Calls			
No Events Emitted For Changes To Protocol Values	#0019	Informational	Partial
Multiple Deployable Contracts In A File	#0020	Informational	Resolved
Gensis Minting Trust	#0021	Low Risk	See Comment
ABIEncoderV2 Is Enabled By Default	#0022	Informational	Resolved
Gas Optimization 1	#0023	Informational	Resolved
Gas Optimization 2	#0024	Informational	Resolved
Gas Optimization 3	#0025	Informational	See Comment
No Dev Fund Allocation For Shares	#0026	Informational	Resolved
Oracle Information	#0027	Informational	Resolved
Duplicate And Different Contract Implementations	#0028	Informational	Resolved
Multiple Implementations Of Fixed Point Math	#0029	Informational	Resolved
Pool Cannot Handle More Than 18 Decimal Places	#0030	Informational	Resolved
No Receive Function In Timelock	#0031	Informational	Resolved
Unbound Loop And Duplicate Data	#0032	Informational	See Comment
Unused Interfaces	#0033	Informational	Resolved
Initialize Used In Non-Proxy Contracts	#0034	Informational	Resolved
Missing Check For Zero Address	#0035	Informational	Resolved
Incorrect safemath conversion	#0036	Medium Risk	Resolved
Oracle address never set	#0037	Medium Risk	Resolved
Uniswap oracle requires overflow	#0038	Medium Risk	Resolved

Treasury Effective Collateral Ratio Can Be Paused	#0039	Medium Risk	Resolved
Buyback and Recollateralize Restricted by Timelock	#0040	Low Risk	Resolved
No Oracles	#0041	High Risk	Resolved
Share Reward Controller Is EOA	#0042	High Risk	See Comment
Low Time Delay On Timelock	#0043	Medium Risk	Resolved
Community Reward Exceeds Hard Cap	#0044	Low Risk	Resolved
Multiple Timelocks	#0045	Informational	Resolved
Pair Oracles Are Not Initialized	#0046	High Risk	See Comment
Treasury Uses Share Oracle To Get Governance Token Price	#0047	Medium Risk	Resolved
Ownership Of Custom Token Oracles	#0048	Medium Risk	Resolved
Inconsistent Oracle Precision	#0049	Low Risk	Resolved

Introduction

Obelisk was commissioned by Argano on the 26th of June 2021 to conduct a comprehensive audit of Arganos' contracts. The following audit was conducted between the 1st of July 2021 and the 31st of August 2021 and delivered on the 4th of September 2021. Three of Obelisk's security experts went through the related contracts using industry standards to find if any vulnerabilities could be exploited.

The comprehensive test was conducted in a specific test environment that utilized exact copies of the published contract. The auditors also conducted a manual visual inspection of the code to find security flaws that automatic tests would not find.

While conducting the audit, the Obelisk security team uses best practices to ensure that the reviewed contracts are thoroughly examined against all angles of attack. This is done by evaluating the codebase and whether it gives rise to significant risks. During the audit, Obelisk assesses the risks and assigns a risk level to each section together with an explanatory comment. Take note that the comments from the project team are their opinion and not the opinion of Obelisk.

The audit was conducted on contracts that were not yet live in a production environment which turned out to be for the best as the project team could work on solving all severe coding issues found. A comprehensive on-chain analysis was conducted as the contracts went live in order to match the audited contracts with the published contracts to make sure that the project deployed what they said they would.

Initially, there were multiple findings of all severity that were conveyed to the project team. The project team worked hard to get the recommended solutions into the contracts before they were deployed. All issues that were of the higher severity scale were solved in the code besides issues #13, #21 & #46. These three issues can only be solved after the project is live, hence we will check back on those issues and update this audit if they are solved as recommended.

Issue #42 is complex to solve as the code is still there, but placed behind a timelock and as long as that is in place, this issue can be described as solved.

The informational findings are good to know while interacting with the project but don't directly damage the project in its current state.

Please see each section of the audit to get a full understanding of the audit.

Findings

Manual Analysis

Migration Function

SEVERITY	High Risk
RESOLVED	**YES**
FINDING ID	#0001
LOCATION	Treasury.sol -> 387-399 Pool.sol -> 252-256

```
1     function migrate(address _new_treasury) external onlyOwner notMigrated {
2         migrated = true;
3         uint256 _share_balance = IERC20(CNUSD).balanceOf(address(this));
4         if (_share_balance > 0) {
5             IERC20(CNUSD).safeTransfer(_new_treasury, _share_balance);
6         }
7         if (rebalancing_pool_collateral != address(0)) {
8             uint256 _collateral_balance =
  IERC20(rebalancing_pool_collateral).balanceOf(address(this));
9             if (_collateral_balance > 0) {
10                IERC20(rebalancing_pool_collateral).safeTransfer(_new_treasury,
  _collateral_balance);
11            }
12        }
13    }
```

```
1     function migrate(address _new_pool) external override nonReentrant onlyOwner notMigrated {
2         migrated = true;
3         uint256 availableCollateral =
  ERC20(collateral).balanceOf(address(this)).sub(unclaimed_pool_collateral);
4         ERC20(collateral).safeTransfer(_new_pool, availableCollateral);
5     }
```

DESCRIPTION	Treasury and pools have migration functions. Migration functions can cause the immediate loss of all deposited funds to a malicious user as the contract owner.
RECOMMENDATION	Ensure that a timelock is in place to give forward warning of migration. Timelock should exceed various internal time restrictions (such as withdrawal lockups) by a sufficient amount to give users time to react to such a critical change.

MITIGATED/COMMENT	Ownership of treasury and pool contracts was transferred to a timelock contract. Timelock (2nd deployment) 0xA41819313D9b0f7680a9cAf1009203a16811b349

Unrestricted Proxy Call In Treasury

SEVERITY	High Risk
RESOLVED	**YES**
FINDING ID	#0002
LOCATION	Treasury.sol -> 506-526

```
function executeTransaction(
    address target,
    uint256 value,
    string memory signature,
    bytes memory data
) public onlyOwner returns (bytes memory) {
    bytes memory callData;

    if (bytes(signature).length == 0) {
        callData = data;
    } else {
        callData = abi.encodePacked(bytes4(keccak256(bytes(signature))), data);
    }

    // solium-disable-next-line security/no-call-value

    (bool success, bytes memory returnData) = target.call{value: value}(callData);
    require(success, string("Treasury::executeTransaction: Transaction execution reverted."));
    emit TransactionExecuted(target, value, signature, data);
    return returnData;
}
```

DESCRIPTION	Treasury has a proxy function for use in emergencies. This function can be called at any time, however, bypassing many restrictions on contract calls from the treasury. For example, this can be used to call *IERC20(CNUSD).safeTransfer* to withdraw all shares. It can also be used to call *IPool(rebalancing_pool).transferCollateralToTreasury* and then *IERC20(rebalancing_pool_collateral).safeTransfer* to withdraw all collateral from the pools. A malicious actor as the contract owner can therefore remove all tokens and collateral from the protocol.
RECOMMENDATION	Remove this function and specify the types of emergency behavior to be managed by the Treasury contract.
MITIGATED/COMMENT	The proxy call function was removed entirely.

CNUSD Dev Vesting Duration Can Be Initialized To The Past

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0003
LOCATION	Dollar.sol -> 59-74 Share.sol -> 56-71

```
function initialize(
    address _devFund,
    address _rewardController,
    uint256 _startTime
) external onlyOwner {
    require(!initialized, "alreadyInitialized");
    require(_rewardController != address(0), "!rewardController");
    initialized = true;
    devFund = _devFund;
    rewardController = _rewardController;
    startTime = _startTime;
    endTime = _startTime + VESTING_DURATION;
    devFundLastClaimed = _startTime;

    _mint(msg.sender, 5000 ether);
}
```

DESCRIPTION	Vesting time can be initialized to the past. This can potentially allow the *claimDevFundRewards* to mint up to the *DEV_FUND_ALLOCATION* amount (20 million) as soon as the contract is deployed.
RECOMMENDATION	Add a check to prevent the vesting start time from being in the past.
MITIGATED/COMMENT	The vesting mechanism was removed.

CNUSD Dev Vesting Duration Is 1 Day

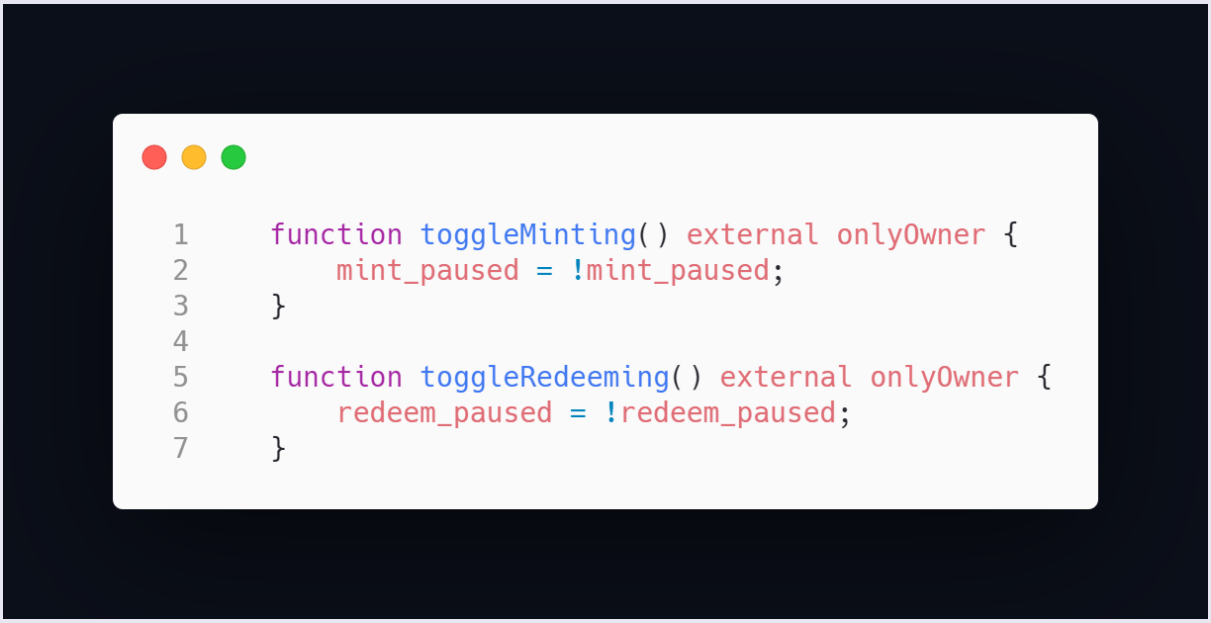
SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0004
LOCATION	Dollar.sol -> 35 Share.sol -> 32

```
1    uint256 public constant VESTING_DURATION = 1 days;//365 days; // 12 months
2
```

DESCRIPTION	The vesting duration is 1 day, for 20% of the initial supply. Please confirm that this behavior is intended.
RECOMMENDATION	Extend the vesting duration to 365 days as per the comment.
MITIGATED/COMMENT	The vesting mechanism was removed.

Pool Can Be Unilaterally Paused

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0005
LOCATION	Pool.sol -> 258-264



```
1     function toggleMinting() external onlyOwner {
2         mint_paused = !mint_paused;
3     }
4
5     function toggleRedeeming() external onlyOwner {
6         redeem_paused = !redeem_paused;
7     }
```

DESCRIPTION	The pool contract can have minting and redemption paused by the contract owner. This will prevent users from interacting with it. If used in concert with the migration function, a malicious attacker may prevent users from withdrawing collateral during a migration.
RECOMMENDATION	Ensure that a timelock is in place to give forward warning of migration. Timelock should exceed various internal time restrictions (such as withdrawal lockups) by a sufficient amount to give users time to react to such a critical change.
MITIGATED/COMMENT	Ownership of pool contracts was transferred to a timelock contract. Timelock (2nd deployment) 0xA41819313D9b0f7680a9cAf1009203a16811b349

Oracle Prices Can Be Stale

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0006
LOCATION	DollarCollateralPairOracle.sol -> 209 ShareWCoinPairOracle.sol -> 211 Treasury.sol -> 140-150

```
1     uint256 public PERIOD = 600; // 10-minute TWAP (time-weighted average price)
2
```

```
1     function dollarPrice() public view returns (uint256) {
2         return IOracle(oracleDollar).consult();
3     }
4
5     function sharePrice() public view returns (uint256) {
6         return IOracle(oracleShare).consult();
7     }
8
9     function gov_token_price() public view returns (uint256) {
10        return IOracle(oracleGovToken).consult();
11    }
```

DESCRIPTION	If using uniswap v2 directly as an oracle: “the oracle consumer should check that the oracle has been recently updated (and call update() if it hasn't), to avoid reading stale data” -Source: Uniswap audit The oracle update periods are 10 minutes while the collateralization rates are updated every 60 minutes. An external call to the oracle 50 minutes into the refresh cooldown means that only the last 10 minutes will be

	considered. This discrepancy can allow external oracle calls to set up arbitrage opportunities on the deposited collateral.
RECOMMENDATION	Add check such that the protocol is not reading stale data. Change the oracle period to match the treasury refresh cooldown. Add an automatic update call to ensure the oracle is up to date.
MITIGATED/COMMENT	The *withOracleUpdates* modifier was added to ensure that the Treasury is always acting with an up to date oracle. The Pool also calls updating on oracles as well. Additionally, the oracle period and treasury refresh times now match at 10 minutes.

Phishing Discount Fee

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0007
LOCATION	Treasury.sol -> 164-167

```
1     function redemption_fee_adjusted() public view returns (uint256) {
2         if (governanceToken == address(0)) return redemption_fee;
3         if (IERC20(governanceToken).balanceOf(tx.origin) > discount_requirenment() ) return
  redemption_fee.div(2);
4     }
```

DESCRIPTION	SWC-115 Authorization through tx.origin. A malicious attacker can phish a user to call their contract and then use their tx.origin to receive a discount on the redemption fee. Someone could even give other people discounts with the same method. Be aware that tx.origin might not stay useful, Vitalik: "Do NOT assume that tx.origin will continue to be usable or meaningful."
RECOMMENDATION	Never use tx.origin for any authorization. Always use msg.sender and then send the address manually if necessary for use in an external contract.
MITIGATED/COMMENT	The project team has modified the check to use *msg.sender* instead of *tx.origin*. Refer to commit 7724f977716ff933c0c4df63e40c4d56e5fdcafd

Redemption Can Be Toggled Between Target And Effective Collateralization

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0008
LOCATION	Treasury.sol -> 273-277 Treasury.sol -> 433-435 Pool.sol -> 183-191

```
1         if (using_effective_collateral_ratio) {
2             effective_collateral_ratio = calcEffectiveCollateralRatio();
3         } else {
4             effective_collateral_ratio = target_collateral_ratio;
5         }
```

```
1     function toggleEffectiveCollateralRatio() public onlyOwner {
2         using_effective_collateral_ratio = !using_effective_collateral_ratio;
3     }
```

```
1         if (_effective_collateral_ratio < COLLATERAL_RATIO_MAX) {
2             uint256 _share_output_value =
  _dollar_amount_post_fee.sub(_dollar_amount_post_fee.mul(_effective_collateral_ratio).div(PRICE_PRECI
  SION));
3             _share_output_amount = _share_output_value.mul(PRICE_PRECISION).div(_share_price);
4         }
5
6         if (_effective_collateral_ratio > 0) {
7             uint256 _collateral_output_value =
  _dollar_amount_post_fee.div(10**missing_decimals).mul(_effective_collateral_ratio).div(PRICE_PRECISI
  ON);
8             _collateral_output_amount =
  _collateral_output_value.mul(PRICE_PRECISION).div(_collateral_price);
9         }
```

DESCRIPTION	The treasury can switch whether the pool will allow redemption of assets using the target or the effective collateralization ratio. Once the collateralization ratio is unpaused, these two values can and will drift apart. If the target ratio is used while it is higher than the effective ratio, redeeming will further decrease the effective ratio. This can result in users being unable to redeem at all because the pool assumes that there is collateral available. For example, suppose the following: the token supply is 100, the collateralization is $80, the effective ratio is 80%, the target ratio is 100%. The first 80 tokens to be redeemed will receive $80. The last 20 tokens will have no collateral at all but can be recorded as redeemed. Furthermore, the redeeming process happens in two steps. Firstly, redeemed collateral is recorded as unclaimed, then it is redeemed. However, there is no that the unclaimed amounts do not exceed the collateral balance. This can cause other protocol functionality to break. The *Pool.collateralDollarBalance* function will revert on underflow by subtracting the unclaimed amount from the collateral balance. Ultimately this will prevent the *Treasury.refreshCollateralRatio* from calculating the effective collateralization ratio when toggling back to using the effective ratio.
RECOMMENDATION	Remove the ability to toggle between the target collateralization and effective collateralization. Only use the effective collateralization ratio.
MITIGATED/COMMENT	Project team has implemented the recommended fix.

Rebalance Pool Can Be Removed

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0009
LOCATION	Treasury.sol -> 328-339 Treasury.sol -> 475-480

```
function removePool(address pool_address) public onlyOwner notMigrated {
    require(pools[pool_address] == true, "!pool");
    // Delete from the mapping
    delete pools[pool_address];
    // 'Delete' from the array by setting the address to 0x0
    for (uint256 i = 0; i < pools_array.length; i++) {
        if (pools_array[i] == pool_address) {
            pools_array[i] = address(0); // This will leave a null in the array and keep the indices the same
            break;
        }
    }
}
```

```
function setRebalancePool(address _rebalance_pool) public onlyOwner {
    require(pools[_rebalance_pool], "!pool");
    require(IPool(_rebalance_pool).getCollateralToken() != address(0), "!poolCollateralToken");
    rebalancing_pool = _rebalance_pool;
    rebalancing_pool_collateral = IPool(_rebalance_pool).getCollateralToken();
}
```

DESCRIPTION	The Treasury ensures that the rebalance pool is part of the treasury when setting it. However, it allows the rebalance pool to be removed later before a new rebalance pool is set. This can adversely affect the functionality of the protocol.
RECOMMENDATION	Check that the rebalance pool is not being removed.
MITIGATED/COMMENT	The project team has implemented the recommended fix.

Migration Not Checked In Treasury

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0010
LOCATION	Treasury.sol -> 373

```
1    function allocateSeigniorage() external nonReentrant checkCondition checkEpoch {
2
```

DESCRIPTION	Treasury contract does not check for the migration status when allocating seigniorage. This may cause unexpected behavior if the contract is migrated in the future while retaining existing links to the foundry.
RECOMMENDATION	Add a *notMigrated* modifier to the *allocateSeigniorage* call.
MITIGATED/COMMENT	Project team has implemented the recommended fix.

Duplicate Implementation Of Burn From

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0011
LOCATION	ERC20Custom.sol -> 79-84 ERC20Custom.sol -> 104-107 Dollar.sol -> 174-177 Dollar.sol -> 105-108 Share.sol -> 102-105

```
1    function burnFrom(address account, uint256 amount) public virtual {
2        uint256 decreasedAllowance = allowance(account, _msgSender()).sub(amount);
3
4        _approve(account, _msgSender(), decreasedAllowance);
5        _burn(account, amount);
6    }
```

```
1    function _burnFrom(address account, uint256 amount) internal virtual {
2        _burn(account, amount);
3        _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount));
4    }
```

```
1    function poolBurnFrom(address b_address, uint256 b_amount) external onlyPools {
2        super._burnFrom(b_address, b_amount);
3        emit ShareBurned(b_address, address(this), b_amount);
4    }
```

DESCRIPTION	The ERC20Custom contract implements both *burnFrom* and *_burnFrom* which have the exact same functionality (though in a slightly different order). The *onlyPools* check may be unnecessary given the allowance check within the burning functions.

RECOMMENDATION	Remove duplicate implementation. Confirm the interaction with the *onlyPools* check is expected.
MITIGATED/COMMENT	Custom implementations of ERC20 replaced with OpenZeppelin.

Oracle Precision Mismatch

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0012
LOCATION	MATICOracle.sol -> 129 ShareOracle.sol -> 142 USDTOracle.sol -> 129

```
1     uint256 private constant PRICE_PRECISION = 1e6;
2
```

DESCRIPTION	Prices from the MATICOracle, ShareOracle, and USDTOracle are given with a precision of 6 digits, however, 18 digits are expected and used in other oracles. Note that chainlink oracles typically have 8 or 18 decimals (See documentation).
RECOMMENDATION	Use the correct precision.
MITIGATED/COMMENT	Note: Precision is flexible in the noted oracles. The precision will be confirmed on chain to match the treasury. Project team has ensured correctness via on chain testing.

Start Time Can Be Initialized To Past

SEVERITY	Low Risk
RESOLVED	**NO**
FINDING ID	#0013
LOCATION	Treasury.sol -> 130-136 Treasury.sol -> 486-490

```
1    function initialize(uint256 _startTime, uint256 _epoch_length) external onlyOwner {
2        require(initialized == false, "alreadyInitialized");
3        startTime = _startTime;
4        epoch_length = _epoch_length;
5        lastEpochTime = _startTime.sub(epoch_length);
6        initialized = true;
7    }
```

```
1    function resetStartTime(uint256 _startTime) external onlyOwner {
2        require(_epoch == 0, "already started");
3        startTime = _startTime;
4        lastEpochTime = _startTime.sub(8 hours);
5    }
```

DESCRIPTION	Treasury start time can be initialized into the past. During epoch 0, the start time can also be changed. This can allow multiple epochs to be passed in rapid succession.
RECOMMENDATION	Add a require to ensure that the start time is set to an appropriate time range. Prevent *resetStartTime* from changing when initialized.
MITIGATED/COMMENT	The project team has implemented the recommended fix for the constructor but the resetStartTime() is still unchecked. The ability to reset the start time is automatically disabled after epoch 0. This will be confirmed at a later time.

Swap Path Uses Pairs Instead Of Tokens

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0014
LOCATION	Treasury.sol -> 298-317

```
1     function _swap(
2         address _input_token,
3         uint256 _input_amount,
4         uint256 _min_output_amount
5     ) internal returns (uint256) {
6         require(uniswap_router != address(0) && uniswap_pair_SHARE_WMATIC != address(0) &&
  uniswap_pair_WMATIC_COLLATERALL != address(0), "!quickswap");
7         if (_input_amount == 0) return 0;
8         address[] memory _path = new address[](2);
9         if (_input_token == CNUSD) {
10            _path[0] = uniswap_pair_SHARE_WMATIC;
11            _path[1] = uniswap_pair_WMATIC_COLLATERALL;
12        } else {
13            _path[0] = uniswap_pair_WMATIC_COLLATERALL;
14            _path[1] = uniswap_pair_SHARE_WMATIC;
15        }
16        IERC20(_input_token).safeApprove(uniswap_router, 0);
17        IERC20(_input_token).safeApprove(uniswap_router, _input_amount);
18        uint256[] memory out_amounts =
  IUniswapRouter(uniswap_router).swapExactTokensForTokens(_input_amount, _min_output_amount, _path,
  address(this), block.timestamp.add(1800));
19        return out_amounts[out_amounts.length - 1];
20    }
```

DESCRIPTION	Uniswap router path is expected to be an array of token addresses. The code suggests that the LP tokens are what will be traded. Refer to uniswap documentation for swapExactTokensForTokens (https://uniswap.org/docs/v2/smart-contracts/router02/#swapexacttokensfortokens).
RECOMMENDATION	Ensure that the swap path is set correctly.
MITIGATED/COMMENT	Swap path now correctly uses tokens.

Local Copies Of OpenZeppelin Contracts

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0015
LOCATION	Address.sol Context.sol ERC20.sol ERC20Custom.sol ERC20Detailed.sol Ownable.sol ReentrancyGuard.sol SafeERC20.sol SafeMath.sol

DESCRIPTION	The contract includes local copies of OpenZeppelin contracts. The logic within the contracts mostly matches OpenZeppelin 3.3 but appears to be missing some functionality (eg. mod within SafeMath). *ERC20Detailed.sol* and *ERC20.sol* combined appear to replicate OpenZeppelin's *ERC20*. *ERC20Custom.sol* appears to replicate part of OpenZeppelin's ERC20Burnable. Noteworthy changes: • In the ERC20 contracts, the balance and allowance variables were changed from private to internal. This may allow inherited contracts to modify user balances and allowances. For example, ARGANO uses a highly atypical method to mint the initial supply. • In ERC20Custom, name, symbol and decimals were removed. • In Ownable the owner is public and may be modified by inherited contracts contract. • In Ownable, the event announcing the change of ownership was removed. • In ReentrancyGuard, the entered status is checked after the modified contract executes.
RECOMMENDATION	Import OpenZeppelin instead of using local copies. Additionally, SafeMath is no longer necessary as of solidity version 0.8.0 and can be removed.
MITIGATED/COMMENT	Project team has implemented the recommended fix.

Local Copies Of Chainlink Contracts

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0016
LOCATION	AggregatorV3Interface.sol DollarOracle.sol -> 30-49 MATICOracle.sol -> 105-124 ShareOracle.sol -> 30-49 USDTOracle.sol -> 105-124

```
1   function latestAnswer() external view returns(int256);
```

DESCRIPTION	The contract includes local copies of Chainlink contracts. The logic within the contracts mostly matches those of Chainlink 0.8 contracts. However, there is an additional function *latestAnswer* which is not documented on Chainlink.
RECOMMENDATION	Import Chainlink instead of using local copies.
MITIGATED/COMMENT	Project team has implemented the recommended fix.

Local Copies Of Uniswap Libraries

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0017
LOCATION	DollarCollateralPairOracle.sol -> 98-112 ShareWCoinPairOracle.sol -> 98-112 DollarCollateralPairOracle.sol -> 29-82 ShareWCoinPairOracle.sol -> 29-82

```
1 library Babylonian {
2     // ...
3 }
```

```
1 library FixedPoint {
2     // ...
3 }
```

DESCRIPTION	The contract includes local copies of uniswap-lib contracts. The logic is identical to the uniswap-lib 1.1.4 contracts.
RECOMMENDATION	Import uniswap-lib instead of using local copies.
MITIGATED/COMMENT	Duplicate copies of the contract were removed. However local copies are still used. Project Team Comment: “local copies desired”

No Events Emitted For Important Function Calls

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0018
LOCATION	ARGANO.sol -> 18-22 Dollar.sol -> 76-96 Share.sol -> 73-93 Dollar.sol -> 117-124 Share.sol -> 114-121 Dollar.sol -> 185-187 Ownable.sol -> 28-32

```
constructor() ERC20Detailed("Argano", "AGO", 18) {
    _totalSupply = 65000000 * (10**uint256(18));
    _balances[msg.sender] = _totalSupply;
}
```

```
function claimCommunityRewards(uint256 amount) external onlyOwner {
        require(amount > 0, "invalidAmount");
        require(initialized, "!initialized");
        require(rewardController != address(0), "!rewardController");
        uint256 _remainingRewards = COMMUNITY_REWARD_ALLOCATION.sub(communityRewardClaimed);
        require(amount <= _remainingRewards, "exceedRewards");
        communityRewardClaimed = communityRewardClaimed.add(amount);
        _mint(rewardController, amount);
    }

    /* ========== RESTRICTED FUNCTIONS ========== */

    function setTreasuryAddress(address _treasury) external onlyOwner {
        treasury = _treasury;
    }

    function setDevFund(address _devFund) external {
        require(msg.sender == devFund, "!dev");
        require(_devFund != address(0), "zero");
        devFund = _devFund;
    }
```

```
1     function claimDevFundRewards() external {
2         require(msg.sender == devFund, "!dev");
3         uint256 _pending = unclaimedDevFund();
4         if (_pending > 0 && devFund != address(0)) {
5             _mint(devFund, _pending);
6             devFundLastClaimed = block.timestamp;
7         }
8     }
```

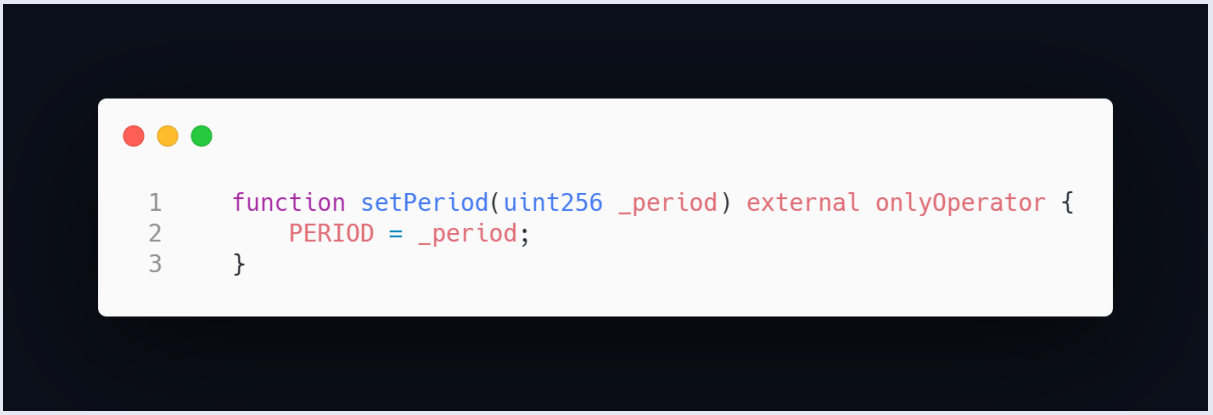
```
1     function setTreasuryAddress(address _treasury) public onlyOwner {
2         treasury = _treasury;
3     }
```

```
1     function transferOwnership(address newOwner) public onlyOwner {
2         if (newOwner != address(0)) {
3             owner = newOwner;
4         }
5     }
```

DESCRIPTION	Functions that change important variables should include emit logs such that users can more easily monitor the change.
RECOMMENDATION	Add the required events and emit them.
MITIGATED/COMMENT	Project team has implemented the recommended fix.

No Events Emitted For Changes To Protocol Values

SEVERITY	Informational
RESOLVED	**PARTIAL**
FINDING ID	#0019
LOCATION	DollarCollateralPairOracle.sol -> 234-236 DollarOracle.sol -> 167-173 MATICOracle.sol -> 138-140 ShareWCoinPairOracle.sol -> 236-238 ShareOracle.sol -> 167-173 USDTOracle.sol -> 138-140 Foundry.sol -> 100-107 Treasury.sol -> 401-502 Pool.sol -> 258-292



```
1    function setPeriod(uint256 _period) external onlyOperator {
2        PERIOD = _period;
3    }
```

```
1    function setchainlinkUsdtUsd(address _chainlinkUsdtUsd) external onlyOperator {
2        chainlinkUsdtUsd = _chainlinkUsdtUsd;
3    }
4
5    function setOracleAGOUSDUSDT(address _oracleAGOUSDUSDT) external onlyOperator {
6        oracleAGOUSDUSDT = _oracleAGOUSDUSDT;
7    }
```

```
function setChainlinkMATICUsd(address _chainlinkMATICUsd) external onlyOperator {
    chainlinkMATICUsd = _chainlinkMATICUsd;
}
```

```
function setPeriod(uint256 _period) external onlyOperator {
    PERIOD = _period;
}
```

```
function setChainlinkWMaticUsd(address _chainlinkWMaticUsd) external onlyOperator {
    chainlinkWMaticUsd = _chainlinkWMaticUsd;
}

function setOracleCnusdWMatic(address _oracleCnusdWMatic) external onlyOperator {
    oracleCnusdWMatic = _oracleCnusdWMatic;
}
```

```
function setChainlinkUsdtUsd(address _chainlinkUSDTUsd) external onlyOperator {
    chainlinkUSDTUsd = _chainlinkUSDTUsd;
}
```

```
function setLockUp(uint256 _withdrawLockupEpochs) external onlyOwner {
    require(_withdrawLockupEpochs <= 56, "_withdrawLockupEpochs: out of range"); // <= 2 week
    withdrawLockupEpochs = _withdrawLockupEpochs;
}

function setOracle(address _oracle) external onlyOwner {
    oracle = _oracle;
}
```

```
    function setGovTokenValueForDiscount(uint256 _gov_token_value_for_discount) public onlyOwner {
        gov_token_value_for_discount = _gov_token_value_for_discount;
    }

    function setRedemptionFee(uint256 _redemption_fee) public onlyOwner {
        redemption_fee = _redemption_fee;
    }

    function setMintingFee(uint256 _minting_fee) public onlyOwner {
        minting_fee = _minting_fee;
    }

    function setRatioStep(uint256 _ratio_step) public onlyOwner {
        ratio_step = _ratio_step;
    }

    function setPriceTarget(uint256 _price_target) public onlyOwner {
        price_target = _price_target;
    }

    function setRefreshCooldown(uint256 _refresh_cooldown) public onlyOwner {
        refresh_cooldown = _refresh_cooldown;
    }

    function setPriceBand(uint256 _price_band) external onlyOwner {
        price_band = _price_band;
    }

    function toggleCollateralRatio() public onlyOwner {
        collateral_ratio_paused = !collateral_ratio_paused;
    }

    function toggleEffectiveCollateralRatio() public onlyOwner {
        using_effective_collateral_ratio = !using_effective_collateral_ratio;
    }

    function setOracleDollar(address _oracleDollar) public onlyOwner {
        oracleDollar = _oracleDollar;
    }

    function setOracleShare(address _oracleShare) public onlyOwner {
        oracleShare = _oracleShare;
    }

    function setGovTokenShare(address _oracleGovToken) public onlyOwner {
        oracleGovToken = _oracleGovToken;
    }

    function setDollarAddress(address _AGOUSD) public onlyOwner {
        AGOUSD = _AGOUSD;
    }

    function setShareAddress(address _CNUSD) public onlyOwner {
        CNUSD = _CNUSD;
    }

    function setGovTokenAddress(address _governanceToken) public onlyOwner {
        governanceToken = _governanceToken;
    }

    function setStrategist(address _strategist) external onlyOwner {
        strategist = _strategist;
    }

    function setUniswapParams(
        address _uniswap_router,
        address _uniswap_pair_CNUSD_WMATIC,
        address _uniswap_pair_WMATIC_USDT
    ) public onlyOwner {
        uniswap_router = _uniswap_router;
        uniswap_pair_SHARE_WMATIC = _uniswap_pair_CNUSD_WMATIC;
        uniswap_pair_WMATIC_COLLATERALL = _uniswap_pair_WMATIC_USDT;
    }

    function setRebalancePool(address _rebalance_pool) public onlyOwner {
        require(pools[_rebalance_pool], "!pool");
        require(IPool(_rebalance_pool).getCollateralToken() != address(0), "!poolCollateralToken");
        rebalancing_pool = _rebalance_pool;
        rebalancing_pool_collateral = IPool(_rebalance_pool).getCollateralToken();
    }

    function setRebalanceCooldown(uint256 _rebalance_cooldown) public onlyOwner {
        rebalance_cooldown = _rebalance_cooldown;
    }

    function resetStartTime(uint256 _startTime) external onlyOwner {
        require(_epoch == 0, "already started");
        startTime = _startTime;
        lastEpochTime = _startTime.sub(8 hours);
    }

    function setFoundry(address _foundry) public onlyOwner {
        foundry = _foundry;
    }

    function setEpochLength(uint256 _epoch_length) public onlyOwner {
        epoch_length = _epoch_length;
    }

    function setExcessDistributionRatio(uint256 _excess_collateral_distributed_ratio) public
onlyStrategistOrOwner {
        excess_collateral_distributed_ratio = _excess_collateral_distributed_ratio;
    }
```

```
function toggleMinting() external onlyOwner {
    mint_paused = !mint_paused;
}

function toggleRedeeming() external onlyOwner {
    redeem_paused = !redeem_paused;
}

function setOracle(address _oracle) external onlyOwner {
    oracle = _oracle;
}

function setPoolCeiling(uint256 _pool_ceiling) external onlyOwner {
    pool_ceiling = _pool_ceiling;
}

function setCollectRedemptionDelay(uint256 _collect_redemption_delay) external onlyOwner {
    collect_redemption_delay = _collect_redemption_delay;
}

function setRedemptionDelay(uint256 _redemption_delay) external onlyOwner {
    redemption_delay = _redemption_delay;
}

function setTreasury(address _treasury) external onlyOwner {
    emit TreasuryTransferred(treasury, _treasury);
    treasury = _treasury;
}

// Transfer collateral to Treasury to execute strategies
function transferCollateralToTreasury(uint256 amount) external override onlyTreasury {
    require(amount > 0, "zeroAmount");
    require(treasury != address(0), "invalidTreasury");
    ERC20(collateral).safeTransfer(treasury, amount);
}
```

DESCRIPTION	Functions that change important variables should include emit logs such that users can more easily monitor the change.
RECOMMENDATION	Add emit logs to these functions. Add require statements to ensure reasonable values for numeric values. Ensure that these values are secured via a timelock.
MITIGATED/COMMENT	Some events were added for various function calls, but not for all. Project team comment: “adding events and zero address check is required hight amount of events. We will restrict amount of setters in next version of protocol.”

Multiple Deployable Contracts In A File

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0020
LOCATION	Dollar.sol -> 16 Dollar.sol -> 135

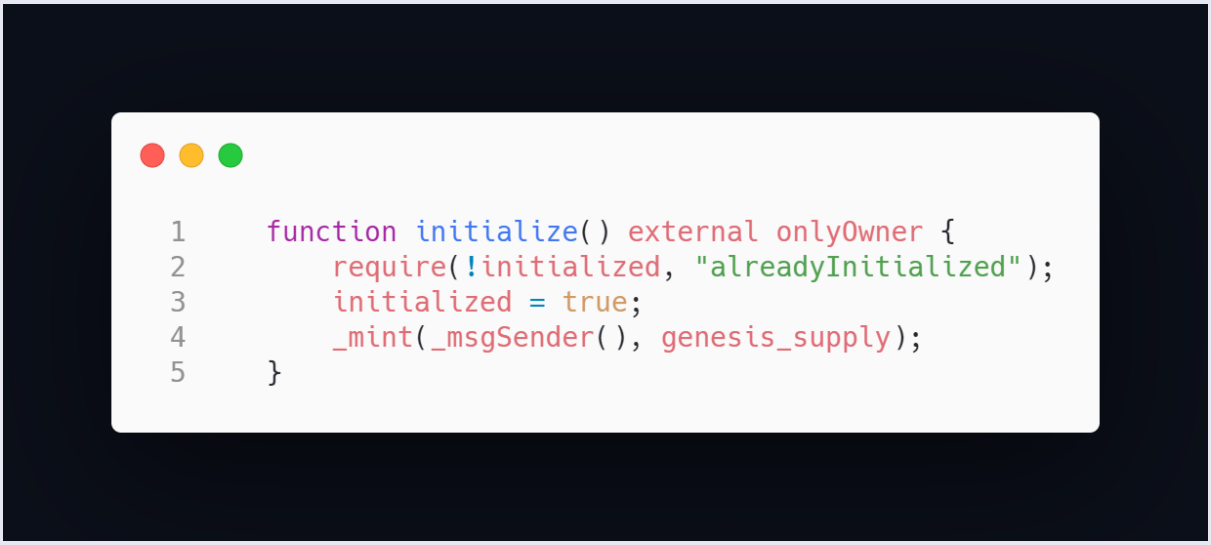
```
1 contract CNUSD is ERC20Custom, Ownable {
2
```

```
1 contract Dollar is ERC20Custom, IDollar, Ownable {
2
```

DESCRIPTION	Dollar.sol includes multiple deployable contracts. The best practice is to separate deployable contracts into separate files.
RECOMMENDATION	Separate the contracts into different files.
MITIGATED/COMMENT	The CNUSD contract was removed entirely.

Genesis Minting Trust

SEVERITY	Low Risk
RESOLVED	**SEE COMMENT**
FINDING ID	#0021
LOCATION	Dollar.sol -> 165-168

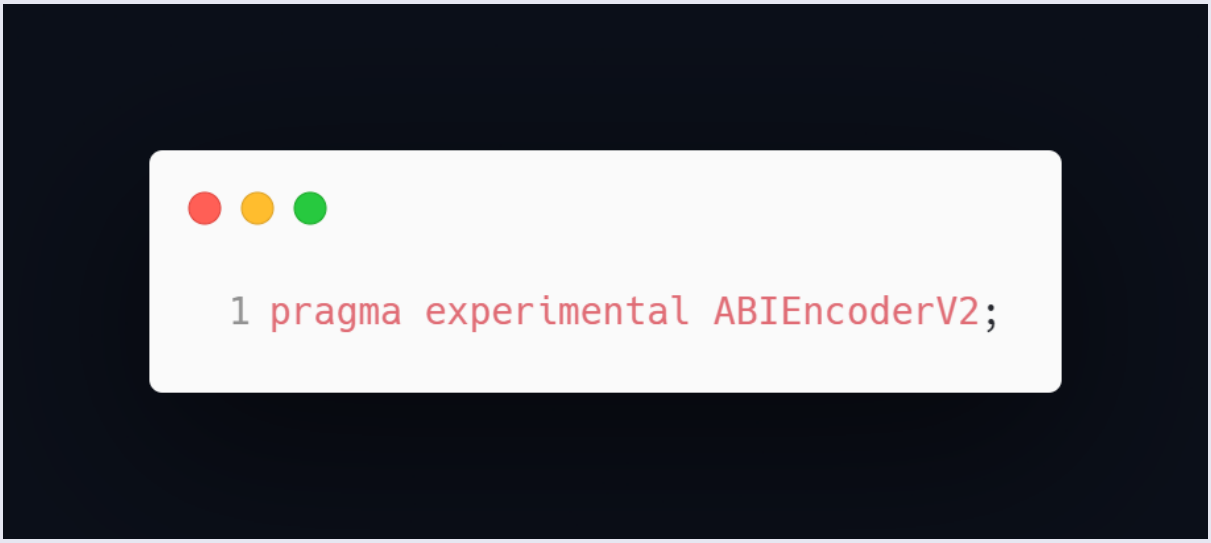


```
function initialize() external onlyOwner {
    require(!initialized, "alreadyInitialized");
    initialized = true;
    _mint(_msgSender(), genesis_supply);
}
```

DESCRIPTION	The genesis supply is minted to the caller which is the owner in this case. A better way would be to provide the initial liquidity in a decentralized way where the caller isn't trusted with the *genesis_supply*.
RECOMMENDATION	Add the initial liquidity via the contract instead of having that done externally.
MITIGATED/COMMENT	Project team: "The main governance token $AGO is minted to the EOA wallet with privileged possibilities in order to manage correct token distribution according to the latest tokenomics featured in the documentation. This was made due to the fact that we didn't establish a partnership with the fundraising platform at the moment of the audit process."

ABIEncoderV2 Is Enabled By Default

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0022
LOCATION	Dollar.sol -> 3 Foundry.sol -> 3 IDollar.sol -> 4 IShare.sol -> 4 Pool.sol -> 3 ReentrancyGuard.sol -> 3 Share.sol -> 3 ShareWrapper.sol -> 3 Treasury.sol -> 3



```
1 pragma experimental ABIEncoderV2;
```

DESCRIPTION	The ABIEncoderV2 is enabled by default in solidity 0.8.0 and higher. Refer to the solidity documentation (https://docs.soliditylang.org/en/v0.8.0/080-breaking-changes.html).
RECOMMENDATION	Remove unnecessary pragma statements.
MITIGATED/COMMENT	Project team has implemented the recommended fix.

Gas Optimization 1

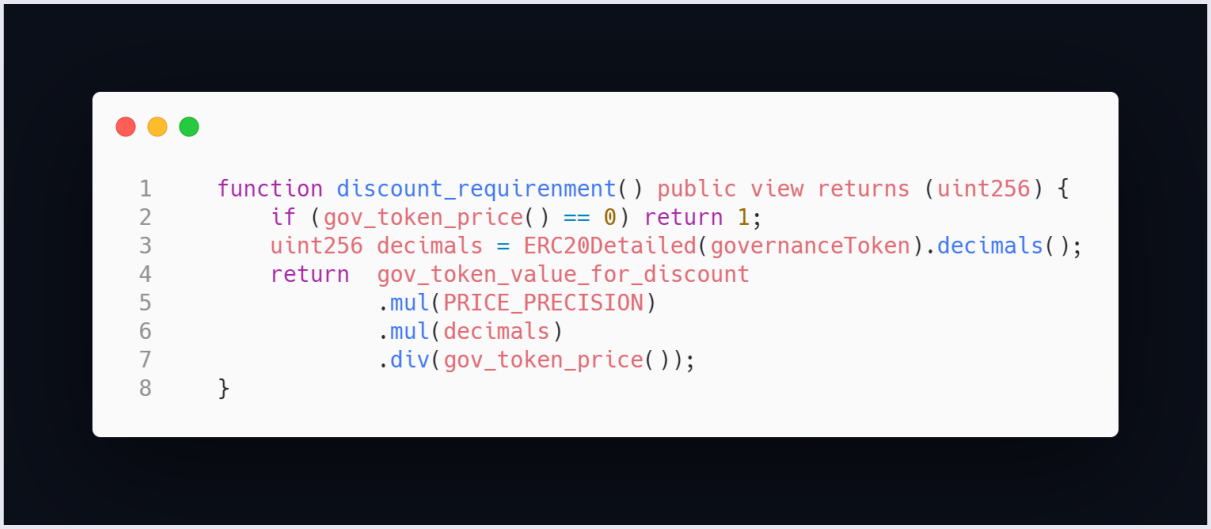
SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0023
LOCATION	Dollar.sol -> 41 Share.sol -> 38

```
1     uint256 public devFundEmissionRate = DEV_FUND_ALLOCATION / VESTING_DURATION;
2
```

DESCRIPTION	The variable should be assigned the constant keyword to optimize gas usage.
RECOMMENDATION	Add the constant keyword.
MITIGATED/COMMENT	Dev fund mechanism was removed.

Gas Optimization 2

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0024
LOCATION	Treasury.sol -> 169-176



```
1    function discount_requirenment() public view returns (uint256) {
2        if (gov_token_price() == 0) return 1;
3        uint256 decimals = ERC20Detailed(governanceToken).decimals();
4        return  gov_token_value_for_discount
5                .mul(PRICE_PRECISION)
6                .mul(decimals)
7                .div(gov_token_price());
8    }
```

DESCRIPTION	*Gov_token_price* is called twice in this function if it's not 0 which would result in a higher gas price than storing it locally the first time.
RECOMMENDATION	Store the result in a local variable instead of calling the function twice.
MITIGATED/COMMENT	Project team has implemented the recommended fix.

Gas Optimization 3

SEVERITY	Informational
RESOLVED	**See Comment**
FINDING ID	#0025
LOCATION	Treasury.sol -> 220-229 Treasury.sol -> 327-339

```
function globalCollateralValue() public view returns (uint256) {
      uint256 total_collateral_value = 0;
      for (uint256 i = 0; i < pools_array.length; i++) {
          // Exclude null addresses
          if (pools_array[i] != address(0)) {
              total_collateral_value =
total_collateral_value.add(IPool(pools_array[i]).collateralDollarBalance());
          }
      }
      return total_collateral_value;
  }
```

```
  // Remove a pool
  function removePool(address pool_address) public onlyOwner notMigrated {
      require(pools[pool_address] == true, "!pool");
      // Delete from the mapping
      delete pools[pool_address];
      // 'Delete' from the array by setting the address to 0x0
      for (uint256 i = 0; i < pools_array.length; i++) {
          if (pools_array[i] == pool_address) {
              pools_array[i] = address(0); // This will leave a null in the array and keep the
indices the same
              break;
          }
      }
  }
```

DESCRIPTION	The array can be filled with many *address*(0) which would result in extra gas costs when *globalCollateralValue*() is called (from a contract). This can also simplify the function since it removes the need to check for 0 addresses.
RECOMMENDATION	Considering using the pop() function to clear up the array values thereby saving gas when *globalCollateralValue*().
MITIGATED/COMMENT	The project team has decided to let the current implementation stay and change it in the next version of the protocol.

No Dev Fund Allocation For Shares

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0026
LOCATION	Share.sol -> 31

```
1 uint256 public constant DEV_FUND_ALLOCATION = 0 ether; // 0
```

DESCRIPTION	Share contract includes a mechanism for vesting of dev fund but has no fund allocation.
RECOMMENDATION	Make sure this is the expected behavior. If so, remove the dev fund allocation mechanism.
MITIGATED/COMMENT	Dev fund mechanism was removed.

Oracle Information

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0027
LOCATION	DollarCollateralPairOracle.sol -> 209 ShareWCoinPairOracle.sol -> 211

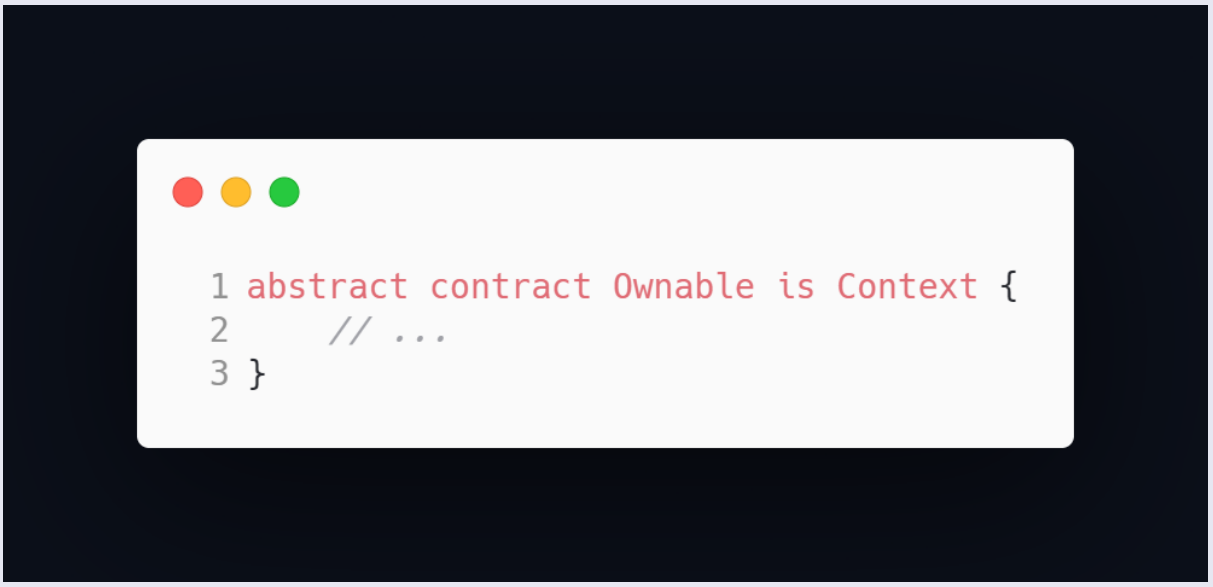
```
1    uint256 public PERIOD = 600; // 10-minute TWAP (time-weighted average price)
2
```

DESCRIPTION	When using an on-chain single oracle two important factors are (Accuracy, Security). Increasing the time will raise the security but lower the accuracy. Decreasing time will lower security but raise accuracy.
RECOMMENDATION	Make sure that the loss of accuracy does not cause too much negative(in terms of value lost for the protocol) arbitrage and the security is enough to not have the pool manipulated.
MITIGATED/COMMENT	Project team has settled on a 10 minute period.

Duplicate And Different Contract Implementations

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0028
LOCATION	Context.sol DollarCollateralPairOracle.sol -> 114-123 DollarOracle.sol -> 60-69 MATICOracle.sol -> 5-14 ShareOracle.sol -> 60-69 ShareWCoinPairOracle.sol -> 114-123 USDTOracle.sol -> 5-14 Ownable.sol DollarCollateralPairOracle.sol -> 125-154 DollarOracle.sol -> 71-100 MATICOracle.sol -> 16-45 ShareOracle.sol -> 71-100 ShareWCoinPairOracle.sol -> 125-154 USDTOracle.sol -> 16-45 SafeMath.sol DollarCollateralPairOracle.sol -> 5-27 DollarOracle.sol -> 6-28 MATICOracle.sol -> 81-103 ShareOracle.sol -> 6-28 ShareWCoinPairOracle.sol -> 5-27 USDTOracle.sol -> 81-103

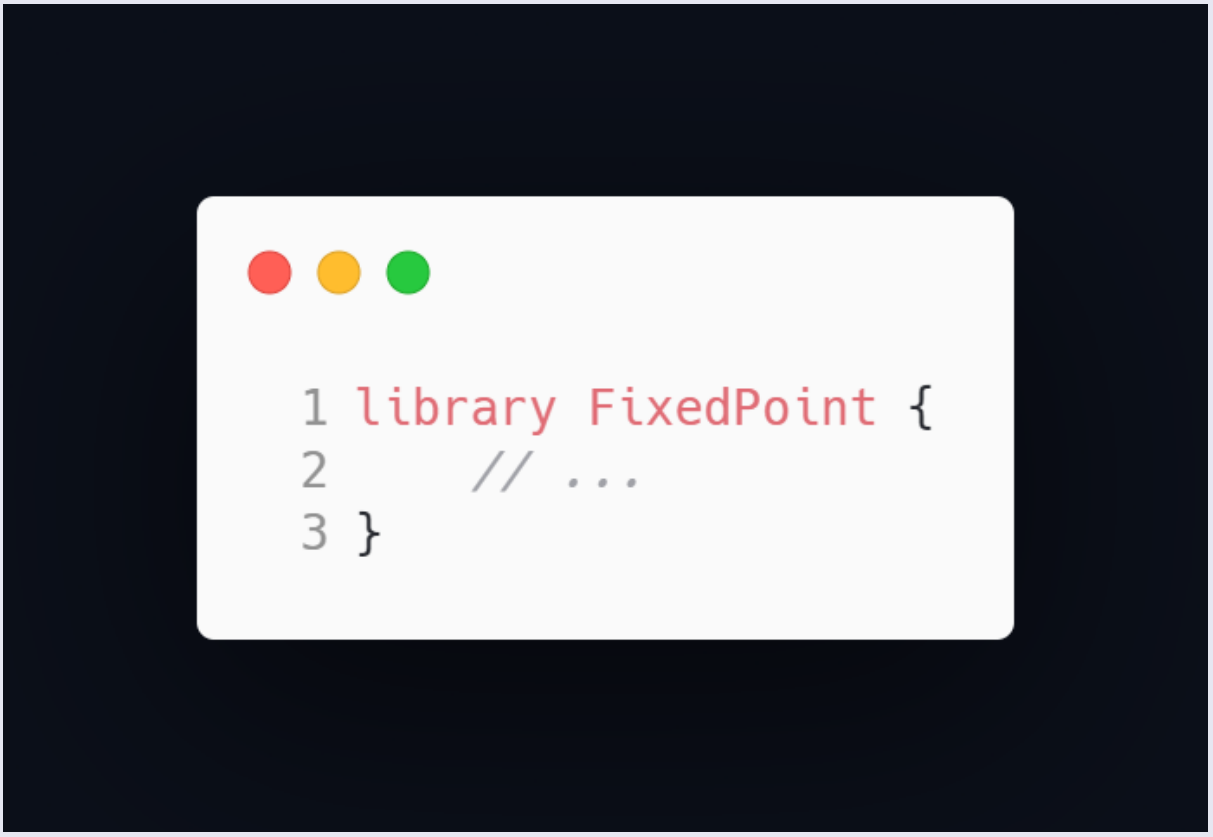
```
1 abstract contract Context {
2     // ...
3 }
```



DESCRIPTION	Implementations of Context, Ownable, and SafeMath within the oracle folder do not match those provided in *libraries* and *utilityContracts* folders.
RECOMMENDATION	Ensure that a single consistent implementation of the contracts is used between contracts. Since these contracts are variants of the openzeppelin, it is best to import from openzeppelin. Additionally, SafeMath is no longer necessary as of solidity 0.8.0.
MITIGATED/COMMENT	The project team has removed the local copies and is now using openzeppelin import instead.

Multiple Implementations Of Fixed Point Math

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0029
LOCATION	DollarCollateralPairOracle.sol -> 29-82 ShareWCoinPairOracle.sol -> 29-82 DollarCollateralPairOracle.sol -> 84-96 ShareWCoinPairOracle.sol -> 84-96



DESCRIPTION	Fixed point arithmetic functionality is duplicated between the two indicated files.
RECOMMENDATION	Remove UQ112x112 as it does not appear to be used anywhere.
MITIGATED/COMMENT	Duplicate contracts were removed and replaced with equivalent OpenZeppelin contracts.

Pool Cannot Handle More Than 18 Decimal Places

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0030
LOCATION	Pool.sol -> 91

```
1         missing_decimals = uint256(18).sub(ERC20Detailed(_collateral).decimals());
2
```

DESCRIPTION	Pool automatically adjusts for tokens with fewer than 18 decimals. However, it will not work for tokens with more than 18 decimals.
RECOMMENDATION	Most tokens will use 18 decimal places, so this is not a concern. However, it is recommended to check that tokens used as collateral will have the required decimals.
MITIGATED/COMMENT	Project team noted that a token with more than 18 decimals could not be used. Attempting to do so would revert the constructor on underflow.

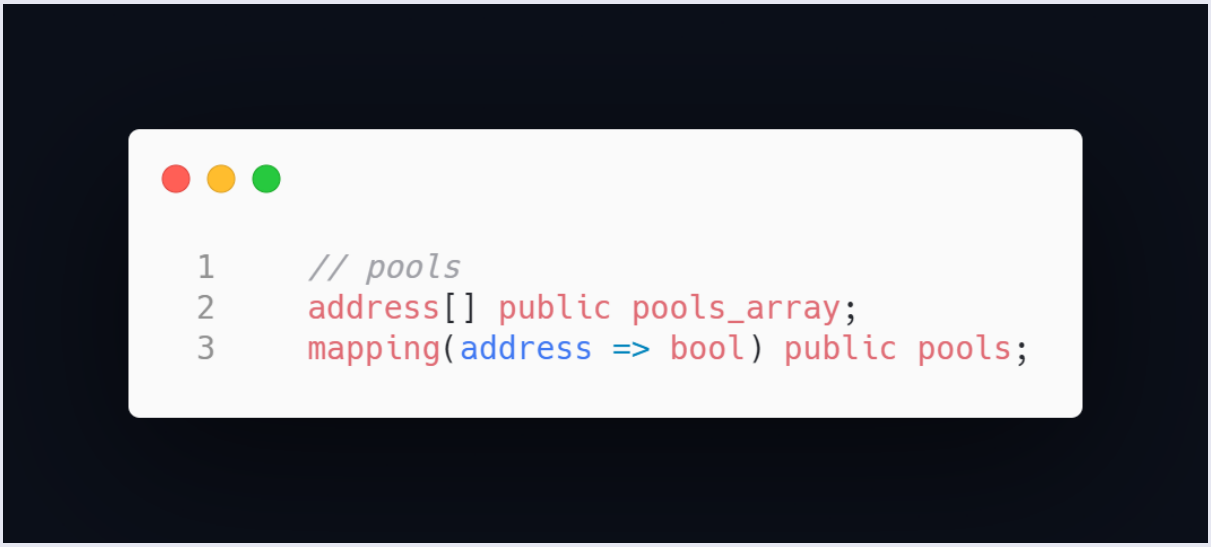
No Receive Function In Timelock

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0031
LOCATION	Timelock.sol

DESCRIPTION	Timelock is missing *receive* function. This may prevent it from executing payable functions.
RECOMMENDATION	Add a default *receive* function.
MITIGATED/COMMENT	Project team has implemented the recommended fix.

Unbound Loop And Duplicate Data

SEVERITY	Informational
RESOLVED	**See Comment**
FINDING ID	#0032
LOCATION	Treasury.sol -> 36-38 Treasury.sol -> 328-339



```
1    // pools
2    address[] public pools_array;
3    mapping(address => bool) public pools;
```

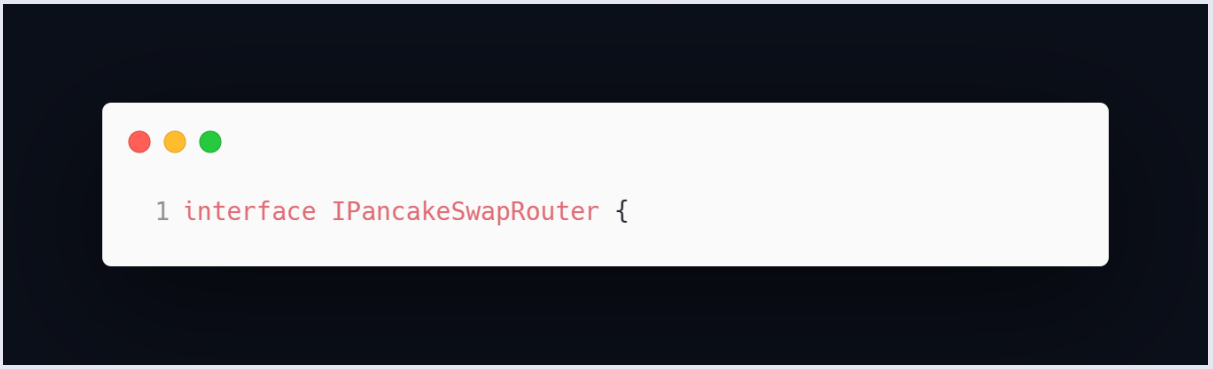
```
1    function removePool(address pool_address) public onlyOwner notMigrated {
2        require(pools[pool_address] == true, "!pool");
3        // Delete from the mapping
4        delete pools[pool_address];
5        // 'Delete' from the array by setting the address to 0x0
6        for (uint256 i = 0; i < pools_array.length; i++) {
7            if (pools_array[i] == pool_address) {
8                pools_array[i] = address(0); // This will leave a null in the array and keep the
  indices the same
9                break;
10           }
11       }
12   }
```

DESCRIPTION	Using unbound loops may cause transactions to revert due to excess gas fees. Duplicate data is stored in both the *pools_array* and the *pools* variables. This makes the code unnecessarily complex. An alternative data structure can be used to avoid looping on the EVM while only verifying the result on-chain.
RECOMMENDATION	Consider using a struct array to store the necessary data

	needed instead. Remove the loop by specifying a pool index directly.
MITIGATED/COMMENT	The project team has decided to leave the implementation as is, for now. The project team will change it in the next version of the protocol.

Unused Interfaces

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0033
LOCATION	IAggregatorV3.sol -> 5 IPairOracle.sol -> 6 IPancakeSwapRouter.sol -> 5 IShareDevFund.sol -> 6 IStakePool.sol -> 6 IUniswapLP.sol -> 4 IValueLiquidPair.sol -> 6 IValuePool.sol -> 6 IWETH.sol -> 5




1 interface IShareDevFund {


1 interface IStakePool {


1 interface IUniswapLP {


1 interface IValueLiquidPair {


1 interface IValuePool {

DESCRIPTION	The noted interfaces are not used. Some of them are duplicated in the flattened oracle contracts.
RECOMMENDATION	Remove the unused interfaces. Replace instances in flattened contracts with imports to the interfaces folder.
MITIGATED/COMMENT	The unused interfaces were removed or used in other contracts.

Initialize Used In Non-Proxy Contracts

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0034
LOCATION	Dollar.sol -> 59-74 Dollar.sol -> 165-169 Foundry.sol -> 88-98 Share.sol -> 56-71 Treasury.sol -> 130-136

```
1     function initialize(
2         address _devFund,
3         address _rewardController,
4         uint256 _startTime
5     ) external onlyOwner {
6         require(!initialized, "alreadyInitialized");
7         require(_rewardController != address(0),
  "!rewardController");
8         initialized = true;
9         devFund = _devFund;
10        rewardController = _rewardController;
11        startTime = _startTime;
12        endTime = _startTime + VESTING_DURATION;
13        devFundLastClaimed = _startTime;
14
15        _mint(msg.sender, 5000 ether);
16    }
```

```
    function initialize() external onlyOwner {
        require(!initialized, "alreadyInitialized");
        initialized = true;
        _mint(_msgSender(), genesis_supply);
    }
```

```
    function initialize(address _collateral, address _share, address _treasury) public notInitialized {
        collateral = _collateral;
        share = _share;
        treasury = _treasury;

        FoundrySnapshot memory genesisSnapshot = FoundrySnapshot({time: block.number, rewardReceived: 0, rewardPerShare: 0});
        foundryHistory.push(genesisSnapshot);
        withdrawLockupEpochs = 8; // Lock for 8 epochs before release withdraw
        initialized = true;
        emit Initialized(msg.sender, block.number);
    }
```

```
    function initialize(
        address _devFund,
        address _rewardController,
        uint256 _startTime
    ) external onlyOwner {
        require(!initialized, "alreadyInitialized");
        require(_rewardController != address(0),
"!rewardController");
        initialized = true;
        devFund = _devFund;
        rewardController = _rewardController;
        startTime = _startTime;
        endTime = _startTime + VESTING_DURATION;
        devFundLastClaimed = _startTime;

        _mint(msg.sender, 5000 ether);
    }
```

```
    function initialize(uint256 _startTime, uint256
_epoch_length) external onlyOwner {
        require(initialized == false,
"alreadyInitialized");
        startTime = _startTime;
        epoch_length = _epoch_length;
        lastEpochTime = _startTime.sub(epoch_length);
        initialized = true;
    }
```

DESCRIPTION	The initialize function is typically used to set up proxy contracts.
RECOMMENDATION	Remove initialize functions and move the functionality into the constructor or rename to better describe functionality.
MITIGATED/COMMENT	Initialize functionality was moved to the constructor of all noted contracts.

Incorrect Safemath Conversion

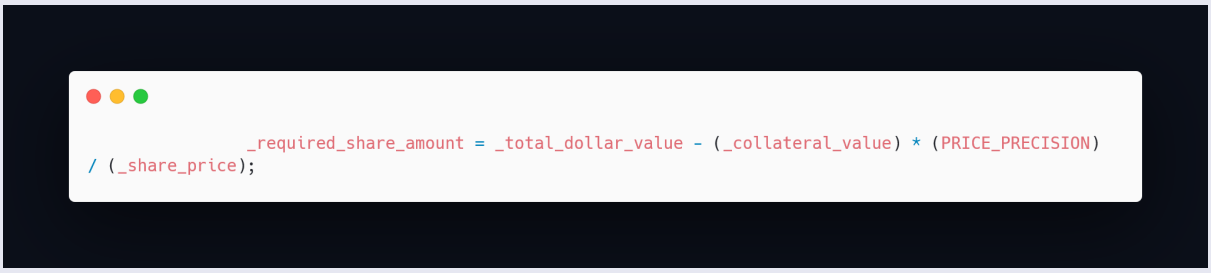
SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0036
LOCATION	Initial contracts Commit [3e3885ab9c6423df3bdf3c1507de8667f5697a0c] [Pool.sol -> 99] [Pool.sol -> 151]

```
        return
(ERC20(collateral).balanceOf(address(this)).sub(unclaimed_pool_collateral)).mul(10**missing_decimals).m
ul(collateral_usd_price).div(PRICE_PRECISION);
```

```
                _required_share_amount =
_total_dollar_value.sub(_collateral_value).mul(PRICE_PRECISION).div(_share_price);
```

LOCATION	Revised contracts Commit [802abf1f52bc59cbc80767d872839a8d8c7408ce] [Pool.sol -> 93] [Pool.sol -> 146]

```
        return ERC20(collateral).balanceOf(address(this)) - unclaimed_pool_collateral *
10**missing_decimals * collateral_usd_price / PRICE_PRECISION;
```



```
                _required_share_amount = _total_dollar_value - (_collateral_value) * (PRICE_PRECISION)
/ (_share_price);
```

DESCRIPTION	Order of operations has changed during conversion from SafeMath to builtin operators. This will cause incorrect behaviour of the protocol.
RECOMMENDATION	Ensure that the order of operations is consistent.
MITIGATED/COMMENT	Project team has fixed the issue. Refer to commit: ef6c894ccc62083a792d6ecaf1f91c6c5d473f01

Oracle Address Never Set

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0037
LOCATION	Revised contracts Commit 802abf1f52bc59cbc80767d872839a8d8c7408ce Treasury.sol -> 21 Treasury.sol -> 144-146 Treasury.sol -> 166-198 Pool.sol -> 130-210

```
address public oracleGovToken;//custom token oracle
```

```
function gov_token_price() public view returns (uint256) {
    return IOracle(oracleGovToken).consult();
}
```

```
    function discount_requirenment() public view returns (uint256) {
        uint256 govTokenPrice = gov_token_price();
        if (govTokenPrice == 0) return 1;
        uint256 decimals = IERC20Metadata(governanceToken).decimals();
        return  gov_token_value_for_discount * PRICE_PRECISION * decimals / govTokenPrice;
    }

    function info()
        external
        view
        override
        returns (
            uint256,
            uint256,
            uint256,
            uint256,
            uint256,
            uint256,
            uint256,
            uint256
        )
    {
        return (
            dollarPrice(),
            sharePrice(),
            IERC20(dollar).totalSupply(),
            target_collateral_ratio,
            effective_collateral_ratio,
            globalCollateralValue(),
            minting_fee,
            redemption_fee_adjusted()
        );
    }
```

```
    function mint(
        uint256 _collateral_amount,
        uint256 _share_amount,
        uint256 _dollar_out_min
    ) external notMigrated {
        // ...
        (, uint256 _share_price, , uint256 _target_collateral_ratio, , , uint256 _minting_fee, ) =
ITreasury(treasury).info();
        // ...
    }

    function redeem(
        uint256 _dollar_amount,
        uint256 _share_out_min,
        uint256 _collateral_out_min
    ) external notMigrated {
        // ...
        (, uint256 _share_price, , , uint256 _effective_collateral_ratio, , , uint256 _redemption_fee)
= ITreasury(treasury).info();
        // ...
    }
```

DESCRIPTION	Governance token oracle cannot be set. Various functions related to the governance oracle price will revert and will be non-functional.

RECOMMENDATION	Initialize the oracle in the constructor or add a setter function.
MITIGATED/COMMENT	Project team added a setter to the contract. Refer to commit: ef6c894ccc62083a792d6ecaf1f91c6c5d473f01

Uniswap Oracle Requires Overflow

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0038
LOCATION	Revised contracts Commit 1e1b2c7420c5d1a3aae5e9273ba541083674705e PairOracle.sol -> 79

```
1     function currentCumulativePrices(address _pair)
  internal view returns (
2         uint256 price0Cumulative,
3         uint256 price1Cumulative,
4         uint32 blockTimestamp
5     ){
6         blockTimestamp = currentBlockTimestamp();
7         IUniswapLP uniswapPair = IUniswapLP(_pair);
8         price0Cumulative =
  uniswapPair.price0CumulativeLast();
9         price1Cumulative =
  uniswapPair.price1CumulativeLast();
10
11        (uint112 reserve0, uint112 reserve1, uint32
  _blockTimestampLast) = uniswapPair.getReserves();
12        if (_blockTimestampLast != blockTimestamp) {
13            uint32 timeElapsed = blockTimestamp -
  _blockTimestampLast;
14            price0Cumulative +=
  uint256(FixedPoint.fraction(reserve1, reserve0)._x) *
  timeElapsed;
15            price1Cumulative +=
  uint256(FixedPoint.fraction(reserve0, reserve1)._x) *
  timeElapsed;
16        }
17    }
```

DESCRIPTION	Uniswap oracles require overflow functionality to continue operation when the cumulative price rolls over. Using solidity 0.8.x fixes many overflow issues, however, this particular contract requires the use of overflow.

RECOMMENDATION	Use unchecked to enable overflow within the calculations or use an older version of solidity that doesn't include overflow protection on this contract.
MITIGATED/COMMENT	Project team has implemented the recommended fix. Refer to commit 53fe4408d6771343fe04981aba4a951fb812c8af

Treasury Effective Collateral Ratio Can Be Paused

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0039
LOCATION	Revised contracts Commit 1e1b2c7420c5d1a3aae5e9273ba541083674705e Treasury.sol -> 358 Treasury.sol -> 196

```
1     function toggleCollateralRatio() public onlyOwner
  {collateral_ratio_paused = !collateral_ratio_paused;}
2
```

```
1     function refreshCollateralRatio() public
  withOracleUpdates{
2         // ...
3         effective_collateral_ratio =
  calcEffectiveCollateralRatio();
4         last_refresh_cr_timestamp = block.timestamp;
5     }
```

DESCRIPTION	The contract owner can pause the updating of *target_collateral_ratio* and *effective_collateral_ratio*. This can cause the *effective_collateral_ratio* to drift from the actual collateral ratio.
RECOMMENDATION	Add a separate function to update only the value of *effective_collateral_ratio*. Alternatively, remove the ability to

	pause the collateral ratio.
MITIGATED/COMMENT	The collateralization ratio is now only paused during the initial setup phase and cannot be paused again once the protocol starts. Refer to commit 53fe4408d6771343fe04981aba4a951fb812c8af

Buyback and Recollateralize Restricted by Timelock

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0040
LOCATION	Revised contracts Commit 1e1b2c7420c5d1a3aae5e9273ba541083674705e Treasury.sol -> 289

```
1     // SINGLE POOL STRATEGY
2     // With Treasury v1, we will only utilize collateral
  from a single pool to do rebalancing
3     function buyback(uint256 _collateral_value, uint256
  _min_share_amount) external onlyOwner withOracleUpdates
  notMigrated hasRebalancePool checkRebalanceCooldown {
4         // ...
5     }
6
7     // SINGLE POOL STRATEGY
8     // With Treasury v1, we will only utilize collateral
  from a single pool to do rebalancing
9     function recollateralize(uint256 _share_amount, uint256
  _min_collateral_amount) external onlyOwner
  withOracleUpdates notMigrated hasRebalancePool
  checkRebalanceCooldown {
10        // ...
11    }
```

DESCRIPTION	The contract owner is assumed to be the timelock. As a result, it may be too restrictive to operate the buyback and collateralized mechanisms. For example if it's run from a timelock it would be frontrun when the transaction gets executed.
RECOMMENDATION	Re-add the strategist address and change the modifier of the noted functions to *onlyStrategist* if it's too restrictive under.
MITIGATED/COMMENT	Project team has implemented the recommended fix.

	Refer to commit cfe45e7fb8bf8bc7201f930c5619d3b42873d609

Static Analysis

Missing Check For Zero Address

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0035
LOCATION	Automated Findings

```
CNUSD.constructor(string,string,address)._treasury (contracts/Dollar.sol#53) lacks a zero-check on :
        - treasury = _treasury (contracts/Dollar.sol#56)
CNUSD.initialize(address,address,uint256)._devFund (contracts/Dollar.sol#60) lacks a zero-check on :
        - devFund = _devFund (contracts/Dollar.sol#67)
CNUSD.setTreasuryAddress(address)._treasury (contracts/Dollar.sol#88) lacks a zero-check on :
        - treasury = _treasury (contracts/Dollar.sol#89)
Dollar.constructor(string,string,address)._treasury (contracts/Dollar.sol#159) lacks a zero-check on
:
        - treasury = _treasury (contracts/Dollar.sol#162)
Dollar.setTreasuryAddress(address)._treasury (contracts/Dollar.sol#185) lacks a zero-check on :
        - treasury = _treasury (contracts/Dollar.sol#186)
Foundry.initialize(address,address,address)._collateral (contracts/Foundry.sol#88) lacks a zero-
check on :
        - collateral = _collateral (contracts/Foundry.sol#89)
Foundry.initialize(address,address,address)._share (contracts/Foundry.sol#88) lacks a zero-check on
:
        - share = _share (contracts/Foundry.sol#90)
Foundry.initialize(address,address,address)._treasury (contracts/Foundry.sol#88) lacks a zero-check
on :
        - treasury = _treasury (contracts/Foundry.sol#91)
Foundry.setOracle(address)._oracle (contracts/Foundry.sol#105) lacks a zero-check on :
        - oracle = _oracle (contracts/Foundry.sol#106)
Pool.constructor(address,address,address,address,address,uint256)._dollar (contracts/Pool.sol#78)
lacks a zero-check on :
        - dollar = _dollar (contracts/Pool.sol#85)
Pool.constructor(address,address,address,address,address,uint256)._share (contracts/Pool.sol#79)
lacks a zero-check on :
        - share = _share (contracts/Pool.sol#86)
Pool.constructor(address,address,address,address,address,uint256)._collateral
(contracts/Pool.sol#80) lacks a zero-check on :
        - collateral = _collateral (contracts/Pool.sol#87)
Pool.constructor(address,address,address,address,address,uint256)._governanceToken
(contracts/Pool.sol#81) lacks a zero-check on :
        - governanceToken = _governanceToken (contracts/Pool.sol#88)
Pool.constructor(address,address,address,address,address,uint256)._treasury (contracts/Pool.sol#82)
lacks a zero-check on :
        - treasury = _treasury (contracts/Pool.sol#89)
Pool.setOracle(address)._oracle (contracts/Pool.sol#266) lacks a zero-check on :
        - oracle = _oracle (contracts/Pool.sol#267)
Pool.setTreasury(address)._treasury (contracts/Pool.sol#282) lacks a zero-check on :
        - treasury = _treasury (contracts/Pool.sol#284)
Share.constructor(string,string,address)._treasury (contracts/Share.sol#50) lacks a zero-check on :
        - treasury = _treasury (contracts/Share.sol#53)
Share.initialize(address,address,uint256)._devFund (contracts/Share.sol#57) lacks a zero-check on
:eck on :
        - (success,returnData) = target.call{value: value}(callData) (contracts/Treasury.sol#522)
DollarOracle.constructor(address,address,address)._dollar (contracts/oracle/DollarOracle.sol#145)
lacks a zero-check on :
        - dollar = _dollar (contracts/oracle/DollarOracle.sol#149)
DollarOracle.constructor(address,address,address)._oracleAGOUSDUSDT
(contracts/oracle/DollarOracle.sol#146) lacks a zero-check on :
        - oracleAGOUSDUSDT = _oracleAGOUSDUSDT (contracts/oracle/DollarOracle.sol#150)
DollarOracle.constructor(address,address,address)._chainlinkUsdtUsd
(contracts/oracle/DollarOracle.sol#147) lacks a zero-check on :
        - chainlinkUsdtUsd = _chainlinkUsdtUsd (contracts/oracle/DollarOracle.sol#151)
DollarOracle.setchainlinkUsdtUsd(address)._chainlinkUsdtUsd (contracts/oracle/DollarOracle.sol#167)
lacks a zero-check on :
        - chainlinkUsdtUsd = _chainlinkUsdtUsd (contracts/oracle/DollarOracle.sol#168)
DollarOracle.setOracleAGOUSDUSDT(address)._oracleAGOUSDUSDT (contracts/oracle/DollarOracle.sol#171)
lacks a zero-check on :
        - oracleAGOUSDUSDT = _oracleAGOUSDUSDT (contracts/oracle/DollarOracle.sol#172)
MATICOracle.setChainlinkMATICUsd(address)._chainlinkMATICUsd (contracts/oracle/MATICOracle.sol#138)
lacks a zero-check on :
        - chainlinkMATICUsd = _chainlinkMATICUsd (contracts/oracle/MATICOracle.sol#139)
ShareOracle.constructor(address,address,address)._share (contracts/oracle/ShareOracle.sol#145) lacks
a zero-check on :
        - share = _share (contracts/oracle/ShareOracle.sol#149)
ShareOracle.constructor(address,address,address)._oracleCnusdWMatic
(contracts/oracle/ShareOracle.sol#146) lacks a zero-check on :
        - oracleCnusdWMatic = _oracleCnusdWMatic (contracts/oracle/ShareOracle.sol#150)
ShareOracle.constructor(address,address,address)._chainlinkWMaticUsd
(contracts/oracle/ShareOracle.sol#147) lacks a zero-check on :
        - chainlinkWMaticUsd = _chainlinkWMaticUsd (contracts/oracle/ShareOracle.sol#151)
ShareOracle.setChainlinkWMaticUsd(address)._chainlinkWMaticUsd
(contracts/oracle/ShareOracle.sol#167) lacks a zero-check on :
        - chainlinkWMaticUsd = _chainlinkWMaticUsd (contracts/oracle/ShareOracle.sol#168)
ShareOracle.setOracleCnusdWMatic(address)._oracleCnusdWMatic (contracts/oracle/ShareOracle.sol#171)
lacks a zero-check on :
        - oracleCnusdWMatic = _oracleCnusdWMatic (contracts/oracle/ShareOracle.sol#172)
USDTOracle.setChainlinkUsdtUsd(address)._chainlinkUSDTUsd (contracts/oracle/USDTOracle.sol#138)
lacks a zero-check on :
        - chainlinkUSDTUsd = _chainlinkUSDTUsd (contracts/oracle/USDTOracle.sol#139)

```

```
1
2         - devFund = _devFund (contracts/Share.sol#64)
3 Share.setTreasuryAddress(address)._treasury (contracts/Share.sol#85) lacks a zero-check on :
4         - treasury = _treasury (contracts/Share.sol#86)
5 Timelock.constructor(address,uint256).admin_ (contracts/Timelock.sol#47) lacks a zero-check on :
6         - admin = admin_ (contracts/Timelock.sol#57)
7 Timelock.setPendingAdmin(address).pendingAdmin_ (contracts/Timelock.sol#90) lacks a zero-check on :
8         - pendingAdmin = pendingAdmin_ (contracts/Timelock.sol#95)
9 Timelock.executeTransaction(address,uint256,string,bytes,uint256).target
  (contracts/Timelock.sol#144) lacks a zero-check on :
10        - (success,returnData) = target.call{value: value}(callData) (contracts/Timelock.sol#184-
  185)
11 Treasury.setOracleDollar(address)._oracleDollar (contracts/Treasury.sol#437) lacks a zero-check on :
12        - oracleDollar = _oracleDollar (contracts/Treasury.sol#438)
13 Treasury.setOracleShare(address)._oracleShare (contracts/Treasury.sol#441) lacks a zero-check on :
14        - oracleShare = _oracleShare (contracts/Treasury.sol#442)
15 Treasury.setGovTokenShare(address)._oracleGovToken (contracts/Treasury.sol#445) lacks a zero-check
  on :
16        - oracleGovToken = _oracleGovToken (contracts/Treasury.sol#446)
17 Treasury.setDollarAddress(address)._AGOUSD (contracts/Treasury.sol#449) lacks a zero-check on :
18        - AGOUSD = _AGOUSD (contracts/Treasury.sol#450)
19 Treasury.setShareAddress(address)._CNUSD (contracts/Treasury.sol#453) lacks a zero-check on :
20        - CNUSD = _CNUSD (contracts/Treasury.sol#454)
21 Treasury.setGovTokenAddress(address)._governanceToken (contracts/Treasury.sol#457) lacks a zero-
  check on :
22        - governanceToken = _governanceToken (contracts/Treasury.sol#458)
23 Treasury.setStrategist(address)._strategist (contracts/Treasury.sol#461) lacks a zero-check on :
24        - strategist = _strategist (contracts/Treasury.sol#462)
25 Treasury.setUniswapParams(address,address,address)._uniswap_router (contracts/Treasury.sol#466)
  lacks a zero-check on :
26        - uniswap_router = _uniswap_router (contracts/Treasury.sol#470)
27 Treasury.setUniswapParams(address,address,address)._uniswap_pair_CNUSD_WMATIC
  (contracts/Treasury.sol#467) lacks a zero-check on :
28        - uniswap_pair_SHARE_WMATIC = _uniswap_pair_CNUSD_WMATIC (contracts/Treasury.sol#471)
29 Treasury.setUniswapParams(address,address,address)._uniswap_pair_WMATIC_USDT
  (contracts/Treasury.sol#468) lacks a zero-check on :
30        - uniswap_pair_WMATIC_COLLATERALL = _uniswap_pair_WMATIC_USDT (contracts/Treasury.sol#472)
31 Treasury.setFoundry(address)._foundry (contracts/Treasury.sol#492) lacks a zero-check on :
32        - foundry = _foundry (contracts/Treasury.sol#493)
33 Treasury.executeTransaction(address,uint256,string,bytes).target (contracts/Treasury.sol#507) lacks
  a zero-ch
```

DESCRIPTION	Address contract values can be set to zero address.
RECOMMENDATION	Consider adding a zero address check.
MITIGATED/COMMENT	Zero checks were added to the contracts.

On-Chain Analysis

No Oracles

SEVERITY	High Risk
RESOLVED	**YES**
FINDING ID	#0041
DESCRIPTION	The following contracts do not have any oracles. Most contract interactions will fail as a result. BTC Foundry (1st deployment) 0x2B0D67fbDE5355A55975907077c05d940Cd5fb0b BTC Pool (1st deployment) 0x007B18eD83fF2f1d7171CbDfA804c2631455269f BTC Treasury (1st deployment) 0xb57b2C2a193C2e3fE1D20Bc194306a0149B978C7 USD Foundry (1st deployment) 0xeB26323574eC2C4C0B64554D3C48584467A8D0a8 USD Pool (1st deployment) 0x12ae4cCc50FD48f05B9F1a0332EA60C69C2654ff USD Treasury (1st deployment) 0x35D59b98EB3d5bD309fEea3F27658ff844a9EF5B
RECOMMENDATION	Deploy and attach the expected oracles.
MITIGATED/COMMENT	Contracts were redeployed with oracles. BTC Foundry (2nd deployment) 0xfB59682d4a14F632207B4024F282576a72DC3845 BTC Pool (2nd deployment) 0xf422cb9C7c382c9B0274ee84D75104387c380678 BTC Treasury (2nd deployment) 0x169B8C64CeD077d941BE91Fc89BB62739F38E647

	USD Foundry (2nd deployment) 0x2a5e24D4F9b43A83c77973f0C727deFab0fEB345 USD Pool (2nd deployment) 0x3Ad6C7640DA03716B7b80A8F804C02b63214d03f USD Treasury (2nd deployment) 0xE4F2961c98aD492447d5C7aAdDCec04D72DCB7c0

Share Reward Controller Is EOA

SEVERITY	High Risk
RESOLVED	**NO**
FINDING ID	#0042
DESCRIPTION	The *rewardController* of the Share contracts receives all community allocations. This is currently set to an externally owned address (EOA). Because shares are minted during the redemption process, it is possible that the redeeming process fails as a result of the shares reaching their hard cap. EOA 0x105d083B04dBC78Bd75D999cd3aF34890dBA1c0D BTC Share (1st deployment) 0x2b83b70cD1c683eB52440C886B43Da59cD21973e USD Share (1st deployment) 0xc235A55e1aEe1d3ac1AdC7c40bCb5a760caA5C67 BTC Share (2nd deployment) 0xf8D6FBf91E1d0ef11E1A1d5384758c04d2517fe9 USD Share (2nd deployment) 0x47C44d60A75e2E8009F4Ce8F166b46e6b94b235F
RECOMMENDATION	Ensure that this behaviour is intended. Change the reward controller to an appropriate community controlled contract if not.

MITIGATED/COMMENT	Project team comment: “Community allocation will be minted directly to pools. After defining the reward distribution strategy, the reward controller will be transferred to a special contract to track all token mints. Share tokens will be minted during redeeming. If the hard cap is reached the only way to get back your USDT is just to sell into the pair (if the token price is pegged). We will apply new stimulating pricing strategies in addition to the existing foundry staking.” Obelisk comment: “It may be challenging to change the reward controller. We recommend minting a fixed amount, storing that in a community controlled contract (eg. a DAO), then revoking ownership of the share contracts. While the reward controller is an EOA, the owner of the share contracts is a timelock. This will provide users with advance notice that community rewards are being minted.”

Low Time Delay On Timelock

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0043
DESCRIPTION	The timelocks have a delay of 12 hours. This is a comparatively short duration and may not be sufficient to allow users to respond to changes to protocol values. BTC Timelock (1st deployment) 0x218Df92131eCea7D763E3528222F0236281C86B9 USD Timelock (1nd deployment) 0x716c6894897281b3571bc75633a86a8f873aa27e Timelock (2nd deployment) 0xA41819313D9b0f7680a9cAf1009203a16811b349
RECOMMENDATION	Increase the timelock delay to at least 72 hours.
MITIGATED/COMMENT	Timelock was redeployed with 48 hours of delay.

Community Reward Exceeds Hard Cap

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0044
DESCRIPTION	The hard cap of CNBTC (BTC Shares) is 8000, while the community reward allocation is 8000000. BTC Share (1st deployment) 0x2b83b70cD1c683eB52440C886B43Da59cD21973e
RECOMMENDATION	Ensure that this behaviour is intended.
MITIGATED/COMMENT	BTC Share was redeployed with a reduced community allocation of 8000. BTC Share (2nd deployment) 0xf8D6FBf91E1d0ef11E1A1d5384758c04d2517fe9

Multiple Timelocks

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0045
DESCRIPTION	Multiple timelocks are used. Note: • All BTC related contracts use the BTC Timelock • All USDT related contracts use the USD Timelock • Argano uses the USD Timelock BTC Timelock (1st deployment) 0x218Df92131eCea7D763E3528222F0236281C86B9 USD Timelock (1st deployment) 0x716c6894897281b3571bc75633a86a8f873aa27e
RECOMMENDATION	No change is necessary.
MITIGATED/COMMENT	Contracts were redeployed with a single timelock. Timelock (2nd deployment) 0xA41819313D9b0f7680a9cAf1009203a16811b349

Pair Oracles Are Not Initialized

SEVERITY	High Risk
RESOLVED	**NO**
FINDING ID	#0046
DESCRIPTION	Pair oracles are not initialized and are therefore non-functional. Argano WCoin Pair Oracle (2nd deployment) 0xC3469607D9F30c08a335919F8adDDC3AC8658265 Expected ARGANO/MATIC BTC Dollar Collateral Pair Oracle (2nd deployment) 0x3c824128274ec10666b22aAE687B0bFcE2CDdB4B Expected BTC Dollar/USD Pair BTC Share WCoin Pair Oracle (2nd deployment) 0x4A2001B62c5e1DCaCDa63eA827662F12340A3A50 Expected BTC Share/MATIC USD Dollar Collateral Pair Oracle (2nd deployment) 0x54B9B1389dcdE6E2c7eBA4d392A21B4A871b9E9D Expected USD Dollar/USDT Pair USD Share WCoin Pair Oracle (2nd deployment) 0x04432f360D02E3dAF2a85aAB8F422914FC2fb96A Expected USD Share/MATIC
RECOMMENDATION	Initialize the oracles to the correct pairs.
MITIGATED/COMMENT	Pair oracles will be initialized when the appropriate liquidity pairs will be created. This will be verified once the project launches.

Treasury Uses Share Oracle To Get Governance Token Price

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0047
DESCRIPTION	The treasury contracts have their respective governance token oracles set to the share oracle. As a result, they will get the incorrect price for the governance token. BTC Treasury (2nd deployment) 0x169B8C64CeD077d941BE91Fc89BB62739F38E647 USD Treasury (2nd deployment) 0xE4F2961c98aD492447d5C7aAdDCec04D72DCB7c0
RECOMMENDATION	Deploy and connect an oracle for the governance token.
MITIGATED/COMMENT	The treasuries were updated to use the governance token oracle: Argano Custom Token Oracle (2nd deployment) 0x96EF4e181e1592A44343135Dd1B819030d8CD933

Ownership Of Custom Token Oracles

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0048
DESCRIPTION	The custom token oracle has owner controlled settings which can break its functionality. The contract owner is an externally owned address. EOA 0x105d083b04dbc78bd75d999cd3af34890dba1c0d BTC Dollar Custom Token Oracle (2nd deployment) 0x98c2856706e0D07bE087619B36A64a66dbA8bfe3 BTC Share Custom Token (2nd deployment) 0xB6005e062363a1Da60bD0E39F50a17f61b1017B5 USD Dollar Custom Token (2nd deployment) 0x8b7CAffCc74ee835324F95CB193411Ec32DC8f07 USD Share Custom Token (2nd deployment) 0x79f494526AE8fa543825335b3fB634F575c66b52
RECOMMENDATION	Revoke ownership of the oracle or transfer it to the timelock
MITIGATED/COMMENT	Ownership of custom oracles was transferred to the timelock.

Inconsistent Oracle Precision

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0049
DESCRIPTION	Different types of timelocks use different precisions. This may have unintended consequences in other contracts. The chainlink oracles use 6 digits of precision, while the pair oracles use 18. BTC Collateral Chainlink Oracle (2nd deployment) 0xDc1d3ff4E2bB6145CE02091b1F742dCe203Cfb62 BTC WCoin Chainlink Token (2nd deployment) 0x926a047578a07017AD019AFCB29F92b929315E90 USD Collateral Chainlink Token (2nd deployment) 0xdEe75F81a2D1D2551140c7Fb4D67EAc3231D5026 USD WCoin Chainlink Token (2nd deployment) 0x223637266F0835df686A5C4FC1Efd5a011EB569e
RECOMMENDATION	Ensure that the number of decimals is accounted for in all contracts.
MITIGATED/COMMENT	Project team has verified that oracle precisions work via on chain testing of contracts.

Appendix A - Reviewed Documents

Document	Address
Address.sol	N/A
AggregatorV3Interface.sol	N/A
ARGANO.sol	ARGANO (1st deployment) 0x27484eb4567F50d2a9cE099971Ca4E04fD663470 --- ARGANO (2nd deployment) 0x4e125214Db26128B35c24c66113C63A83029e433
ChainLinkOracle.sol	BTC Collateral Chainlink Oracle (2nd deployment) 0xDc1d3ff4E2bB6145CE02091b1F742dCe203Cfb62 BTC WCoin Chainlink Oracle (2nd deployment) 0x926a047578a07017AD019AFCB29F92b929315E90 USD Collateral Chainlink Oracle (2nd deployment) 0xdEe75F81a2D1D2551140c7Fb4D67EAc3231D5026 USD WCoin Chainlink Oracle (2nd deployment) 0x223637266F0835df686A5C4FC1Efd5a011EB569e
Context.sol	N/A
CustomTokenOracle.sol	BTC Dollar Custom Token Oracle (2nd deployment) 0x98c2856706e0D07bE087619B36A64a66dbA8bfe3 BTC Share Custom Token Oracle (2nd deployment) 0xB6005e062363a1Da60bD0E39F50a17f61b1017B5 Argano Custom Token Oracle (2nd deployment) 0x96EF4e181e1592A44343135Dd1B819030d8CD933 USD Dollar Custom Token Oracle (2nd deployment) 0x8b7CAffCc74ee835324F95CB193411Ec32DC8f07 USD Share Custom Token Oracle (2nd deployment) 0x79f494526AE8fa543825335b3fB634F575c66b52
Dollar.sol	BTC Dollar (1st deployment) 0x7edD6a8f47253f458cb3e13b399231Aa26A25265

	USD Dollar (1st deployment) 0x01BF71B75b2A9a972943d53B692A33a7932c8e59 --- BTC Dollar (2nd deployment) 0xb9f5b73dbF1e2B9b34cd0B76cDB28dE9AD7c95ce USD Dollar (2nd deployment) 0x9d655C916f548cD29E521bCFd3d83Eef8F6DAE59
ERC20.sol	N/A
ERC20Custom.sol	N/A
ERC20Detailed.sol	N/A
Foundry.sol	BTC Foundry (1st deployment) 0x2B0D67fbDE5355A55975907077c05d940Cd5fb0b USD Foundry (1st deployment) 0xeB26323574eC2C4C0B64554D3C48584467A8D0a8 --- BTC Foundry (2nd deployment) 0xfB59682d4a14F632207B4024F282576a72DC3845 USD Foundry (2nd deployment) 0x2a5e24D4F9b43A83c77973f0C727deFab0fEB345
IDollar.sol	N/A
IEpoch.sol	N/A
IERC20.sol	N/A
IFoundry.sol	N/A
IOracle.sol	N/A
IPool.sol	N/A
IShare.sol	N/A
ITreasury.sol	N/A
IUniswapRouter.sol	N/A
Ownable.sol	N/A

PairOracle.sol	Argano WCoin Pair Oracle (2nd deployment) 0xC3469607D9F30c08a335919F8adDDC3AC8658265 BTC Dollar Collateral Pair Oracle (2nd deployment) 0x3c824128274ec10666b22aAE687B0bFcE2CDdB4B BTC Share WCoin Pair Oracle (2nd deployment) 0x4A2001B62c5e1DCaCDa63eA827662F12340A3A50 USD Dollar Collateral Pair Oracle (2nd deployment) 0x54B9B1389dcdE6E2c7eBA4d392A21B4A871b9E9D USD Share WCoin Pair Oracle (2nd deployment) 0x04432f360D02E3dAF2a85aAB8F422914FC2fb96A
Pool.sol	BTC Pool (1st deployment) 0x007B18eD83fF2f1d7171CbDfA804c2631455269f USD Pool (1st deployment) 0x12ae4cCc50FD48f05B9F1a0332EA60C69C2654ff --- BTC Pool (2nd deployment) 0xf422cb9C7c382c9B0274ee84D75104387c380678 USD Pool (2nd deployment) 0x3Ad6C7640DA03716B7b80A8F804C02b63214d03f
ReentrancyGuard.sol	N/A
SafeERC20.sol	N/A
SafeMath.sol	N/A
Share.sol	BTC Share (1st deployment) 0x2b83b70cD1c683eB52440C886B43Da59cD21973e USD Share (1st deployment) 0xc235A55e1aEe1d3ac1AdC7c40bCb5a760caA5C67 --- BTC Share (2nd deployment) 0xf8D6FBf91E1d0ef11E1A1d5384758c04d2517fe9 USD Share (2nd deployment) 0x47C44d60A75e2E8009F4Ce8F166b46e6b94b235F

ShareWrapper.sol	N/A
Timelock.sol	BTC Timelock (1st deployment) 0x218Df92131eCea7D763E3528222F0236281C86B9 USD Timelock (1st deployment) 0x716c6894897281b3571bc75633a86a8f873aa27e --- Timelock (2nd deployment) 0xA41819313D9b0f7680a9cAf1009203a16811b349
Treasury.sol	BTC Treasury (1st deployment) 0xb57b2C2a193C2e3fE1D20Bc194306a0149B978C7 USD Treasury (1st deployment) 0x35D59b98EB3d5bD309fEea3F27658ff844a9EF5B --- BTC Treasury (2nd deployment) 0x169B8C64CeD077d941BE91Fc89BB62739F38E647 USD Treasury (2nd deployment) 0xE4F2961c98aD492447d5C7aAdDCec04D72DCB7c0

Appendix B - Risk Ratings

Risk	Description
High Risk	A fatal vulnerability that can cause immediate loss of Tokens / Funds
Medium Risk	A vulnerability that can cause some loss of Tokens / Funds
Low Risk	A vulnerability that can be mitigated
Informational	No vulnerability

Appendix C - Icons

Icon	Explanation
	Solved by Project Team
	Under Investigation of Project Team
	Unsolved

Appendix D - Testing Standard

An ordinary audit is conducted using these steps.

1. Gather all information
2. Conduct a first visual inspection of documents and contracts
3. Go through all functions of the contract manually (2 independent auditors)
    a. Discuss findings
4. Use specialized tools to find security flaws
    a. Discuss findings
5. Follow up with project lead of findings
6. If there are flaws, and they are corrected, restart from step 2
7. Write and publish a report

During our audit, a thorough investigation has been conducted employing both automated analysis and manual inspection techniques. Our auditing method lays a particular focus on the following important concepts:

- Ensuring that the code and codebase use best practices, industry standards, and available libraries.
- Testing the contract from different angles ensuring that it works under a multitude of circumstances.
- Analyzing the contracts through databases of common security flaws.

OBELISK

Part of Tibereum Group