Ubiquity

Staking

Security Assessment & Formal Verification

Sep 18th, 2025

ryzhak.com

Ubiquity Staking - Security Assessment & Formal Verification

Project Overview

Project Summary

Project Name	**Ubiquity Staking**
Language	Solidity
Codebase	https://github.com/ubiquity/ubiquity-dollar

Project Description

Ubiquity Staking is a set of smart contracts using ERC-2535 Diamond proxy pattern where users are able to stake Curve's UUSD/LUSD LP tokens and get UBQ (Ubiquity Governance) tokens as rewards.

Project Roles

Role	Method	Description
Anybody	massUpdateStakingPools	Refreshes rewards for selected pools
Anybody	stake	Stakes LUSD/UUSD LP tokens to get UBQ rewards later
Anybody	unstake	Unstakes LUSD/UUSD LP tokens and harvests rewards
Anybody	updateStakingPool	Mints rewards to the contract and treasury, updates pool reward parameters
`DEFAULT_ADMIN_ROLE`	createStakingPool	Creates a new staking pool
`DEFAULT_ADMIN_ROLE`	setGovernanceBonusEndBlock	Sets the end block when bonus multiplier is applied
`DEFAULT_ADMIN_ROLE`	setGovernanceBonusMultiplier	Sets amount of bonus tokens minted each block
`DEFAULT_ADMIN_ROLE`	setGovernancePerBlock	Sets amount of UBQ tokens minted each block
`DEFAULT_ADMIN_ROLE`	setGovernanceTreasuryDivider	Sets divider for additional UBQ allocation to the treasury
`DEFAULT_ADMIN_ROLE`	setStakingRewardToken	Sets address of the staking reward token
`DEFAULT_ADMIN_ROLE`	setStakingStartBlock	Sets block number when the staking starts
`DEFAULT_ADMIN_ROLE`	updateStakingPool	Sets allocation points for a single pool

Audit Overview

Audit Summary

Delivery Date	**Sep 18, 2025**
Audit Methodology	Manual Review, Static Analysis, Contract Fuzzing, Formal Verification
Author(s)	Vladimir Ryzhak
Initial Commit	8fbfbe92b5403be031e6d092fd699b746fa13406
Final Commit	7eb880f4fba21494d313924cfb57f6e8dfbc5078
Formal Verification Report	Report URL
Formal Verification CI Setup	PR URL

Audit Scope

Filename	**URL**
`StakingFacet.sol`	https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/facets/StakingFacet.sol
`LibStaking.sol`	https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibStaking.sol

Severity Matrix

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- **High** - results in a considerable risk that may jeopardize the protocol's overall integrity, impacting all or the majority of users.
- **Medium** - results in a non-critical risk for the protocol, impacting either all users or a specific subset, yet remaining unequivocally unacceptable.
- **Low** - losses incurred will be within acceptable limits, attack vectors can be fixed with relative ease.

Likelihood

- **High** - highly probable, presenting significant financial opportunities for exploitation by malicious actors.
- **Medium** - still relatively probable, although contingent upon certain conditions.
- **Low** - requires a unique set of conditions and presents a cost of execution that does not yield a favorable ratio of rewards for the individual involved.

Findings Summary

Severity	**Discovered**	**Confirmed**	**Fixed**
Critical	-	-	-
High	1	1	1
Medium	8	8	3
Low	5	5	4

Severity	Discovered	Confirmed	Fixed
Total	14	14	8

Findings & Resolutions

ID	Title	Severity	Status
H-01	Missing "mass pool update" on creating/updating a pool affects users' rewards	High	Fixed
M-01	Staking and reward tokens overlapping with `LibUbiquityPool` 's collateral skews calculations	Medium	Fixed
M-02	Possible reentrancy	Medium	Fixed
M-03	`setStakingRewardToken` causes DoS of `stake` / `unstake` methods	Medium	Acknowledged
M-04	Staked tokens are stuck in the contract	Medium	Acknowledged
M-05	Staking DoS if `UBQ_MINTER_ROLE` is revoked from the `Diamond` contract	Medium	Acknowledged
M-06	User can get 0 rewards on high stake amounts	Medium	Fixed
M-07	Dust reward tokens are stuck in the contract	Medium	Acknowledged
M-08	Some "weird" ERC20 tokens are not supported	Medium	Acknowledged
L-01	Unused imports	Low	Fixed
L-02	`whenNotPaused` modifier not used	Low	Fixed
L-03	Treasury griefing	Low	Fixed
L-04	`governanceTreasuryDivider` can't be set to 0	Low	Fixed
L-05	DoS when `treasuryAddress` is 0	Low	Acknowledged

Certora Formal Verification Mitigation Review

Property Description	Type	Passed	URL
`pool.accumulatedGovernancePerShare` only increases	High Level	✅	Link
`massUpdateStakingPools()` for a single pool must have the same effect on storage as calling `updateStakingPool`	Unit	✅	Link
`stake()` increases user rewards	Unit	✅	Link
`stake()` transfers staked tokens	Unit	✅	Link
`stake()` updates storage as expected	Unit	✅	Link
`stake()` must not affect other users	Unit	✅	Link
`stake()` must not affect other pools	Unit	✅	Link
`unstake()` increases user rewards	Unit	✅	Link
`unstake()` updates storage as expected	Unit	✅	Link
`unstake()` transfers staked tokens	Unit	✅	Link
`unstake()` must not affect other users	Unit	✅	Link
`unstake()` must not affect other pools	Unit	✅	Link
`unstake()` must not transfer more staked tokens than expected	Unit	✅	Link
`updateStakingPool()` does not update a pool if the pool has already been updated in the current block or pool's LP supply is 0	Unit	✅	Link
`updateStakingPool()` mints rewards to diamond	Unit	✅	Link
`updateStakingPool()` mints rewards to treasury	Unit	✅	Link
`updateStakingPool()` updates storage as expected	Unit	✅	Link
`updateStakingPool()` must not revert unexpectedly	Unit	✅	Link
`updateStakingPool()` does not affect other pools	Unit	✅	Link
`createStakingPool()` updates storage as expected	Unit	✅	Link
`createStakingPool()` must not revert unexpectedly	Unit	✅	Link
`createStakingPool()` does not affect other pools	Unit	✅	Link
`setGovernanceBonusEndBlock()` updates storage as expected	Unit	✅	Link
`setGovernanceBonusEndBlock()` must not revert unexpectedly	Unit	✅	Link
`setGovernanceBonusMultiplier()` updates storage as expected	Unit	✅	Link
`setGovernanceBonusMultiplier()` must not revert unexpectedly	Unit	✅	Link
`setGovernancePerBlock()` updates storage as expected	Unit	✅	Link
`setGovernancePerBlock()` must not revert unexpectedly	Unit	✅	Link
`setGovernanceTreasuryDivider()` updates storage as expected	Unit	✅	Link
`setGovernanceTreasuryDivider()` must not revert unexpectedly	Unit	✅	Link
`setStakingRewardToken()` updates storage as expected	Unit	✅	Link
`setStakingRewardToken()` must not revert unexpectedly	Unit	✅	Link
`setStakingStartBlock()` updates storage as expected	Unit	✅	Link

Property Description	Type	Passed	URL
`setStakingStartBlock()` must not revert unexpectedly	Unit	✅	Link
`updateStakingPool()` updates storage as expected	Unit	✅	Link
`updateStakingPool()` must not revert unexpectedly	Unit	✅	Link
`updateStakingPool()` does not affect other pools	Unit	✅	Link

Findings

[H-01] Missing "mass pool update" on creating/updating a pool affects users' rewards

Description

There're 2 methods which accept array of pool ids where rewards must be updated:

1. https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibStaking.sol#L399
2. https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibStaking.sol#L515

The issue is that if pools' rewards are not updated on creating a new pool or updating an existing one (i.e. empty array is passed here or here) then users' rewards are affected.

Consider an example when update is triggered:

1. Staking pool is created with 100 allocation points & `governancePerBlock == 1 ether`
2. User stakes 1 ether
3. 10 blocks pass
4. The 2nd staking pool is created with 300 allocation points (`poolIdsToUpdate = [0]`)
5. User unstakes 1 ether and gets 10 UBQ rewards (as expected)

Now consider an example when update is NOT triggered:

1. Staking pool is created with 100 allocation points & `governancePerBlock == 1 ether`
2. User stakes 1 ether
3. 10 blocks pass
4. The 2nd staking pool is created with 300 allocation points (`poolIdsToUpdate = []`)
5. User unstakes 1 ether and gets only 2.5 UBQ rewards (while expected to get 10 UBQ rewards)

PoC

```
contract ProtocolTest is DiamondTestSetup {
    UbiquityAlgorithmicDollarManager dollarManager;
    UbiquityGovernance rewardToken;
    MockERC20 rewardToken2;
    MockERC20 stakeToken;
    MockERC20 stakeToken2;

    address user = makeAddr("user");
    address user2 = makeAddr("user2");

    function setUp() public override {
        super.setUp();

        vm.prank(owner);
        dollarManager = new UbiquityAlgorithmicDollarManager(owner);

        vm.prank(owner);
        rewardToken = new UbiquityGovernance(address(dollarManager));
        stakeToken = new MockERC20("STK", "STK", 18);
        stakeToken2 = new MockERC20("STK2", "STK2", 6);
        rewardToken2 = new MockERC20("RWD2", "RWD2", 18);

        // staking setup
        vm.startPrank(admin);
        stakingFacet.setGovernancePerBlock(1 ether);
        stakingFacet.setGovernanceTreasuryDivider(5);
        stakingFacet.setStakingRewardToken(address(rewardToken));
        stakingFacet.setStakingStartBlock(block.number);
        vm.stopPrank();

        // owner grants diamond the "UBQ_MINTER_ROLE"
        // NOTICE: in production environment the diamond contract already has the "UBQ_MINTER_ROLE" role
        vm.prank(owner);
        dollarManager.grantRole(keccak256("UBQ_MINTER_ROLE"), address(diamond));

        // admin creates a new staking pool
        vm.startPrank(admin);
        stakingFacet.createStakingPool(
            100, // allocation points
            stakeToken,
            getAvailablePoolIds() // array of pool ids to update
        );
        vm.stopPrank();

        // mint 100 STK tokens to user
        stakeToken.mint(user, 100 ether);
        // mint 100 STK2 tokens to user
        stakeToken2.mint(user, 100 ether);
        // user approves diamond to spend STK tokens
        vm.prank(user);
        stakeToken.approve(address(diamond), type(uint256).max);
        // user approves diamond to spend STK2 tokens
        vm.prank(user);
        stakeToken2.approve(address(diamond), type(uint256).max);

        // mint 100 STK tokens to user2
        stakeToken.mint(user2, 100 ether);
        // mint 100 STK2 tokens to user2
        stakeToken2.mint(user2, 100 ether);
        // user2 approves diamond to spend STK tokens
        vm.prank(user2);
        stakeToken.approve(address(diamond), type(uint256).max);
        // user2 approves diamond to spend STK2 tokens
        vm.prank(user2);
        stakeToken2.approve(address(diamond), type(uint256).max);
    }

    function testCreateStakingPool_AffectsCalculations_IfMassUpdateIsNotCalled() public {
        vm.prank(user);
        stakingFacet.stake(0, 1 ether);

        // 10 blocks pass
        vm.roll(block.number + 10);

        vm.startPrank(admin);
        stakingFacet.createStakingPool(
            300, // allocation points
            stakeToken,
            getEmptyPoolIds() // <=== HERE UPDATE MUST BE CALLED FOR ALL EXISTING POOLS
        );
        vm.stopPrank();

        vm.prank(user);
        stakingFacet.unstake(0, 1 ether);
        // If update is not called on creating the 2nd pool then user gets 2.5 UBQ rewards
        // while the expected amount is 10 UBQ rewards
        console2.log("User balance (UBQ):", rewardToken.balanceOf(user));
    }

    /**
     * Returns array of available pool ids
     */
    function getAvailablePoolIds() public view returns (uint256[] memory) {
        uint256 poolsLength = stakingFacet.getStakingPoolsLength();
        uint256[] memory availablePoolIds = new uint256[](poolsLength);
        for (uint256 i = 0; i < poolsLength; ++i) {
            availablePoolIds[i] = i;
        }
        return availablePoolIds;
    }

    /**
     * Returns array with empty pool ids
     */
    function getEmptyPoolIds() public view returns (uint256[] memory) {
        uint256[] memory availablePoolIds = new uint256[](0);
        return availablePoolIds;
    }
}
```

Recommendation

Although passing an array here is a safety measure against exceeding block gas limit in case there're too many staking pools consider updating the code to always force pool updates in case you don't plan to use many staking pools.

[M-01] Staking and reward tokens overlapping with `LibUbiquityPool` 's collateral skews calculations

Description

If one of the tokens used as collateral in the LibUbiquityPool is also used as a staking or reward token in the LibStaking then calculations in the LibUbiquityPool may be skewed.

It happens because LibUbiquityPool relies on `balanceOf()` in the following places:

- https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibUbiquityPool.sol#L354
- https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibUbiquityPool.sol#L717

Possible impacts could be:

- users depositing more than the pool's ceiling in the LibUbiquityPool
- staking insolvency when users redeem UUSD for tokens hold in the staking contract because of bypassing this check

Recommendation

In the following methods check that reward and staking tokens do not overlap with collateral tokens from LibUbiquityPool:

- https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibStaking.sol#L399
- https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibStaking.sol#L491

[M-02] Possible reentrancy

Description

There are a couple of calls to external contracts executed before storage updates which violates the Check-Effects-Interactions pattern:

- https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibStaking.sol#L305
- https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibStaking.sol#L307
- https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibStaking.sol#L338
- https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibStaking.sol#L345
- https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibStaking.sol#L374
- https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibStaking.sol#L378

If staking or reward tokens are malicious then it would allow them to reenter the StakingFacet.

In particular, if reward token is malicious then it would allow to drain the whole amount of staked tokens on reentrant call to unstake because `user.amount` is updated after the external call.

See the similar issue at https://solodit.cyfrin.io/issues/m-17-convexmasterchefs-deposit-and-withdraw-can-be-reentered-drawing-all-reward-funds-from-the-contract-if-reward-token-allows-for-transfer-flow-control-code4rena-aura-finance-aura-finance-git.

Recommendation

Use the nonReentrant modifier for all public state changing methods.

[M-03] `setStakingRewardToken` causes DoS of `stake` / `unstake` methods

Description

There is the setStakingRewardToken method which is responsible for updating the staking reward token.

The issue is that when the setStakingRewardToken method is called when there're already a couple of staked amounts it causes the stake and unstake methods to revert with the `ERC20: transfer amount exceeds balance` error on transferring rewards.

Consider an example:

1. `governancePerBlock == 1 ether`
2. User stakes 1 ether
3. 10 blocks pass
4. `updateStakingPool()` is called (at this point user is eligible for 10 UBQ rewards)
5. `setStakingRewardToken()` is called (let's call it RWD_NEW)
6. 10 blocks pass
7. User tries to unstake 1 ether

In the example above the step 7 will revert with the `ERC20: transfer amount exceeds balance` error because user's pending rewards are 20 RWD_NEW tokens while the contract has 10 UBQ and only 10 RWD_NEW tokens.

So admin will have to manually mint additional newly set reward tokens in order to make the staking solvent.

PoC

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.19;

import "forge-std/console2.sol";
```

```solidity
import {DiamondTestSetup} from "../diamond/DiamondTestSetup.sol";
import {UbiquityAlgorithmicDollarManager} from "../../src/deprecated/UbiquityAlgorithmicDollarManager.sol";
import {UbiquityGovernance} from "../../src/deprecated/UbiquityGovernance.sol";
import {MockERC20} from "../../src/dollar/mocks/MockERC20.sol";
import {LibStaking} from "../../src/dollar/libraries/LibStaking.sol";

contract ProtocolTest is DiamondTestSetup {
    UbiquityAlgorithmicDollarManager dollarManager;
    UbiquityGovernance rewardToken;
    MockERC20 rewardToken2;
    MockERC20 stakeToken;
    MockERC20 stakeToken2;

    address user = makeAddr("user");
    address user2 = makeAddr("user2");

    function setUp() public override {
        super.setUp();

        vm.prank(owner);
        dollarManager = new UbiquityAlgorithmicDollarManager(owner);

        vm.prank(owner);
        rewardToken = new UbiquityGovernance(address(dollarManager));

        stakeToken = new MockERC20("STK", "STK", 18);
        stakeToken2 = new MockERC20("STK2", "STK2", 6);
        rewardToken2 = new MockERC20("RWD2", "RWD2", 18);

        // staking setup
        vm.startPrank(admin);
        stakingFacet.setGovernancePerBlock(1 ether);
        stakingFacet.setGovernanceTreasuryDivider(5);
        stakingFacet.setStakingRewardToken(address(rewardToken));
        stakingFacet.setStakingStartBlock(block.number);
        vm.stopPrank();

        // owner grants diamond the "UBQ_MINTER_ROLE"
        // NOTICE: in production environment the diamond contract already has the "UBQ_MINTER_ROLE" role
        vm.prank(owner);
        dollarManager.grantRole(keccak256("UBQ_MINTER_ROLE"), address(diamond));

        // admin creates a new staking pool
        vm.startPrank(admin);
        stakingFacet.createStakingPool(
            100, // allocation points
            stakeToken,
            getAvailablePoolIds() // array of pool ids to update
        );
        vm.stopPrank();

        // mint 100 STK tokens to user
        stakeToken.mint(user, 100 ether);
        // mint 100 STK2 tokens to user
        stakeToken2.mint(user, 100 ether);
        // user approves diamond to spend STK tokens
        vm.prank(user);
        stakeToken.approve(address(diamond), type(uint256).max);
        // user approves diamond to spend STK2 tokens
        vm.prank(user);
        stakeToken2.approve(address(diamond), type(uint256).max);

        // mint 100 STK tokens to user2
        stakeToken.mint(user2, 100 ether);
        // mint 100 STK2 tokens to user2
        stakeToken2.mint(user2, 100 ether);
        // user2 approves diamond to spend STK tokens
        vm.prank(user2);
        stakeToken.approve(address(diamond), type(uint256).max);
        // user2 approves diamond to spend STK2 tokens
        vm.prank(user2);
        stakeToken2.approve(address(diamond), type(uint256).max);
    }

    function testSetStakingRewardToken_MustNotAffectCalculations() public {
        vm.prank(user);
        stakingFacet.stake(0, 1 ether);

        // 10 blocks pass
        vm.roll(block.number + 10);

        // mints 10 UBQ as rewards
        stakingFacet.updateStakingPool(0);

        vm.prank(admin);
        stakingFacet.setStakingRewardToken(address(rewardToken2));

        // 10 blocks pass
        vm.roll(block.number + 10);

        // reverts because it tries to transfer 20 RWD2 tokens while current contract rewards are
        // - 10 UBQ
        // - 10 RWD2
        vm.prank(user);
        stakingFacet.unstake(0, 1 ether);
    }
}
```

Recommendation

Implement a mechanism that would allow users to withdraw both old and newly set reward tokens.

[M-04] Staked tokens are stuck in the contract

Description

There may be a case when users' staked tokens are stuck in the contract because unstake reverts with the `panic: division or modulo by zero` error.

The root cause is that if there exists a pool with 0 allocation points and there're staked tokens then this line tries a division by 0 since `totalAllocationPoints == 0` leading to a DoS of the unstake method.

Consider an example:

1. A new pool with 100 allocation points is created
2. User stakes X tokens in that pool
3. Admin sets allocation points of that pool from 100 to 0
4. 10 blocks pass
5. User tries to unstake X tokens which would revert because of division by 0 here

PoC

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.19;

import "forge-std/console2.sol";
import {DiamondTestSetup} from "../diamond/DiamondTestSetup.sol";
import {UbiquityAlgorithmicDollarManager} from "../../src/deprecated/UbiquityAlgorithmicDollarManager.sol";
import {UbiquityGovernance} from "../../src/deprecated/UbiquityGovernance.sol";
import {MockERC20} from "../../src/dollar/mocks/MockERC20.sol";
import {LibStaking} from "../../src/dollar/libraries/LibStaking.sol";

contract ProtocolTest is DiamondTestSetup {
    UbiquityAlgorithmicDollarManager dollarManager;
    UbiquityGovernance rewardToken;
    MockERC20 rewardToken2;
    MockERC20 stakeToken;
    MockERC20 stakeToken2;

    address user = makeAddr("user");
    address user2 = makeAddr("user2");

    function setUp() public override {
        super.setUp();

        vm.prank(owner);
        dollarManager = new UbiquityAlgorithmicDollarManager(owner);

        vm.prank(owner);
        rewardToken = new UbiquityGovernance(address(dollarManager));

        stakeToken = new MockERC20("STK", "STK", 18);
        stakeToken2 = new MockERC20("STK2", "STK2", 6);
        rewardToken2 = new MockERC20("RWD2", "RWD2", 18);

        // staking setup
        vm.startPrank(admin);
        stakingFacet.setGovernancePerBlock(1 ether);
        stakingFacet.setGovernanceTreasuryDivider(5);
        stakingFacet.setStakingRewardToken(address(rewardToken));
        stakingFacet.setStakingStartBlock(block.number);
        vm.stopPrank();

        // owner grants diamond the "UBQ_MINTER_ROLE"
        // NOTICE: in production environment the diamond contract already has the "UBQ_MINTER_ROLE" role
        vm.prank(owner);
        dollarManager.grantRole(keccak256("UBQ_MINTER_ROLE"), address(diamond));

        // admin creates a new staking pool
        vm.startPrank(admin);
        stakingFacet.createStakingPool(
            100, // allocation points
            stakeToken,
            getAvailablePoolIds() // array of pool ids to update
        );
        vm.stopPrank();

        // mint 100 STK tokens to user
        stakeToken.mint(user, 100 ether);
        // mint 100 STK2 tokens to user
        stakeToken2.mint(user, 100 ether);
        // user approves diamond to spend STK tokens
        vm.prank(user);
        stakeToken.approve(address(diamond), type(uint256).max);
        // user approves diamond to spend STK2 tokens
        vm.prank(user);
        stakeToken2.approve(address(diamond), type(uint256).max);

        // mint 100 STK tokens to user2
        stakeToken.mint(user2, 100 ether);
        // mint 100 STK2 tokens to user2
        stakeToken2.mint(user2, 100 ether);
        // user2 approves diamond to spend STK tokens
        vm.prank(user2);
        stakeToken.approve(address(diamond), type(uint256).max);
        // user2 approves diamond to spend STK2 tokens
        vm.prank(user2);
        stakeToken2.approve(address(diamond), type(uint256).max);
    }

    function test_dosAllocationPoints() public {
        vm.prank(user);
        stakingFacet.stake(0, 1 ether);

        vm.startPrank(admin);
        stakingFacet.updateStakingPool(
            0, // pool id
            0, // allocation points
            getAvailablePoolIds()
        );
        vm.stopPrank();

        // 10 blocks pass
        vm.roll(block.number + 10);

        // reverts while expected behavior is to unstake tokens
        vm.prank(user);
        stakingFacet.unstake(0, 1 ether);
    }

    function getAvailablePoolIds() public view returns (uint256[] memory) {
        uint256 poolsLength = stakingFacet.getStakingPoolsLength();
        uint256[] memory availablePoolIds = new uint256[](poolsLength);
        for (uint256 i = 0; i < poolsLength; ++i) {
            availablePoolIds[i] = i;
        }
        return availablePoolIds;
    }
}
```

Recommendation

Refactor the code to allow users to unstake stucked tokens.

[M-05] Staking DoS if `UBQ_MINTER_ROLE` is revoked from the `Diamond` contract

Description

The Diamond contract must have the UBQ_MINTER_ROLE in order to mint UBQ rewards tokens in the following places:

- https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibStaking.sol#L374
- https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibStaking.sol#L378

The issue is that if the `UBQ_MINTER_ROLE` is revoked from the `Diamond` then users won't be able to unstake their tokens.

Recommendation

Refactor the code to allow users to unstake tokens if `UBQ_MINTER_ROLE` is revoked from the `Diamond` .

[M-06] User can get 0 rewards on high stake amounts

Description

There may be a case when `governancePerBlock` is small and the stake amount is high which may cause this line to return 0 thus making a user to get 0 reward tokens.

Consider an example:

1. `governancePerBlock == 0.0000001 ether`
2. User stakes 10_000_000 LP tokens
3. 10 blocks pass
4. User unstakes 10_000_000 LP tokens (at this point user get 0 UBQ rewards while the expected amount is 0.000001 UBQ rewards)

PoC

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.19;

import "forge-std/console2.sol";
import {DiamondTestSetup} from "../diamond/DiamondTestSetup.sol";
import {UbiquityAlgorithmicDollarManager} from "../../src/deprecated/UbiquityAlgorithmicDollarManager.sol";
import {UbiquityGovernance} from "../../src/deprecated/UbiquityGovernance.sol";
import {MockERC20} from "../../src/dollar/mocks/MockERC20.sol";
import {LibStaking} from "../../src/dollar/libraries/LibStaking.sol";

contract ProtocolTest is DiamondTestSetup {
    UbiquityAlgorithmicDollarManager dollarManager;
    UbiquityGovernance rewardToken;
    MockERC20 rewardToken2;
    MockERC20 stakeToken;
    MockERC20 stakeToken2;

    address user = makeAddr("user");
    address user2 = makeAddr("user2");

    function setUp() public override {
        super.setUp();

        vm.prank(owner);
        dollarManager = new UbiquityAlgorithmicDollarManager(owner);

        vm.prank(owner);
        rewardToken = new UbiquityGovernance(address(dollarManager));

        stakeToken = new MockERC20("STK", "STK", 18);
        stakeToken2 = new MockERC20("STK2", "STK2", 6);
        rewardToken2 = new MockERC20("RWD2", "RWD2", 18);

        // staking setup
        vm.startPrank(admin);
        stakingFacet.setGovernancePerBlock(1 ether);
        stakingFacet.setGovernanceTreasuryDivider(5);
        stakingFacet.setStakingRewardToken(address(rewardToken));
        stakingFacet.setStakingStartBlock(block.number);
        vm.stopPrank();

        // owner grants diamond the "UBQ_MINTER_ROLE"
        // NOTICE: in production environment the diamond contract already has the "UBQ_MINTER_ROLE" role
        vm.prank(owner);
        dollarManager.grantRole(keccak256("UBQ_MINTER_ROLE"), address(diamond));

        // admin creates a new staking pool
        vm.startPrank(admin);
        stakingFacet.createStakingPool(
            100, // allocation points
            stakeToken,
            getAvailablePoolIds() // array of pool ids to update
        );
        vm.stopPrank();

        // mint 100 STK tokens to user
        stakeToken.mint(user, 100 ether);
        // mint 100 STK2 tokens to user
        stakeToken2.mint(user, 100 ether);
        // user approves diamond to spend STK tokens
        vm.prank(user);
        stakeToken.approve(address(diamond), type(uint256).max);
        // user approves diamond to spend STK2 tokens
        vm.prank(user);
        stakeToken2.approve(address(diamond), type(uint256).max);

        // mint 100 STK tokens to user2
        stakeToken.mint(user2, 100 ether);
        // mint 100 STK2 tokens to user2
        stakeToken2.mint(user2, 100 ether);
        // user2 approves diamond to spend STK tokens
        vm.prank(user2);
        stakeToken.approve(address(diamond), type(uint256).max);
        // user2 approves diamond to spend STK2 tokens
        vm.prank(user2);
        stakeToken2.approve(address(diamond), type(uint256).max);
    }

    function test_zeroRewardsDueToPrecisionLoss() public {
        vm.prank(admin);
        stakingFacet.setGovernancePerBlock(0.0000001 ether);

        uint256 amount = 10_000_000 ether;
        stakeToken.mint(user, amount);

        vm.prank(user);
        stakingFacet.stake(0, amount);

        // 10 blocks pass
        vm.roll(block.number + 10);

        vm.prank(user);
        stakingFacet.unstake(0, amount);

        // UBQ balance is 0 while the expected value is 0.000001 UBQ rewards
        console2.log("User balance (UBQ):", rewardToken.balanceOf(user));
    }
}
```

Recommendation

Add a validation for a miminum amount of the `governancePerBlock` parameter.

[M-07] Dust reward tokens are stuck in the contract

Description

There're some reward tokens that may be stuck in the contract due to precision losses and there's no way for admin to retrieve them.

PoC

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.19;

import "forge-std/console2.sol";
import {DiamondTestSetup} from "../diamond/DiamondTestSetup.sol";
import {UbiquityAlgorithmicDollarManager} from "../../src/deprecated/UbiquityAlgorithmicDollarManager.sol";
import {UbiquityGovernance} from "../../src/deprecated/UbiquityGovernance.sol";
import {MockERC20} from "../../src/dollar/mocks/MockERC20.sol";
import {LibStaking} from "../../src/dollar/libraries/LibStaking.sol";

contract ProtocolTest is DiamondTestSetup {
    UbiquityAlgorithmicDollarManager dollarManager;
    UbiquityGovernance rewardToken;
    MockERC20 rewardToken2;
    MockERC20 stakeToken;
    MockERC20 stakeToken2;

    address user = makeAddr("user");
    address user2 = makeAddr("user2");

    function setUp() public override {
        super.setUp();

        vm.prank(owner);
        dollarManager = new UbiquityAlgorithmicDollarManager(owner);

        vm.prank(owner);
        rewardToken = new UbiquityGovernance(address(dollarManager));

        stakeToken = new MockERC20("STK", "STK", 18);
        stakeToken2 = new MockERC20("STK2", "STK2", 6);
        rewardToken2 = new MockERC20("RWD2", "RWD2", 18);

        // staking setup
        vm.startPrank(admin);
        stakingFacet.setGovernancePerBlock(1 ether);
        stakingFacet.setGovernanceTreasuryDivider(5);
        stakingFacet.setStakingRewardToken(address(rewardToken));
        stakingFacet.setStakingStartBlock(block.number);
        vm.stopPrank();

        // owner grants diamond the "UBQ_MINTER_ROLE"
        // NOTICE: in production environment the diamond contract already has the "UBQ_MINTER_ROLE" role
        vm.prank(owner);
        dollarManager.grantRole(keccak256("UBQ_MINTER_ROLE"), address(diamond));

        // admin creates a new staking pool
        vm.startPrank(admin);
        stakingFacet.createStakingPool(
            100, // allocation points
            stakeToken,
            getAvailablePoolIds() // array of pool ids to update
        );
        vm.stopPrank();

        // mint 100 STK tokens to user
        stakeToken.mint(user, 100 ether);
        // mint 100 STK2 tokens to user
        stakeToken2.mint(user, 100 ether);
        // user approves diamond to spend STK tokens
        vm.prank(user);
        stakeToken.approve(address(diamond), type(uint256).max);
        // user approves diamond to spend STK2 tokens
        vm.prank(user);
        stakeToken2.approve(address(diamond), type(uint256).max);

        // mint 100 STK tokens to user2
        stakeToken.mint(user2, 100 ether);
        // mint 100 STK2 tokens to user2
        stakeToken2.mint(user2, 100 ether);
        // user2 approves diamond to spend STK tokens
        vm.prank(user2);
        stakeToken.approve(address(diamond), type(uint256).max);
        // user2 approves diamond to spend STK2 tokens
        vm.prank(user2);
        stakeToken2.approve(address(diamond), type(uint256).max);
    }

    /// forge-config: default.fuzz.runs = 5120
    function testFuzz_RewardsStuckInTheContract(
        uint allocationPointsPool2,
        uint amount,
        uint blocksPassed
    ) public {
        allocationPointsPool2 = bound(allocationPointsPool2, 0, 100_000);
        blocksPassed = bound(blocksPassed, 1, 2628000 * 50); // 50 years
        amount = bound(amount, 1, 1e26); // change to 1e30 to get counterexample

        // mint stake tokens to users
        deal(address(stakeToken), user, amount);
        deal(address(stakeToken2), user2, amount);

        // admin creates 2nd staking pool
        vm.startPrank(admin);
        stakingFacet.createStakingPool(
            allocationPointsPool2, // allocation points
            stakeToken2,
            getAvailablePoolIds() // array of pool ids to update
        );
        vm.stopPrank();

        // users stake tokens
        vm.prank(user);
        stakingFacet.stake(0, amount);
        vm.prank(user2);
        stakingFacet.stake(1, amount);

        vm.roll(block.number + blocksPassed);

        // users unstake tokens
        vm.prank(user);
        stakingFacet.unstake(0, amount);
        vm.prank(user2);
        stakingFacet.unstake(1, amount);

        // finds a counterexample because there're stucked UBQ rewards in the contract
        assertApproxEqAbsDecimal(rewardToken.balanceOf(address(stakingFacet)), 0, 1e14, 18);
    }

    function getAvailablePoolIds() public view returns (uint256[] memory) {
        uint256 poolsLength = stakingFacet.getStakingPoolsLength();
        uint256[] memory availablePoolIds = new uint256[](poolsLength);
        for (uint256 i = 0; i < poolsLength; ++i) {
            availablePoolIds[i] = i;
        }
        return availablePoolIds;
    }
}
```

Recommendation

Add an admin method for retrieving dust reward tokens.

[M-08] Some "weird" ERC20 tokens are not supported

Description

There are many "weird" ERC20 tokens which behave differently from "standard" ERC20 tokens. For example, some have fees on transfer while others are able to rebase token balances.

Some of those tokens must not be used as a staking or reward token because it breaks the staking leading to DoS or incorrect calculations.

For example, if there're fees on staking token transfer then there will be a difference between the staked amount in storage and the actual amount in contract here.

Another example, if staking token is pausable and is actually paused then users won't be able to call stake and unstake methods.

Here's the list of compatibility between the staking contract and "weird" ERC20 tokens:

	Staking Token	Reward Token
Reentrant Calls	✅	✅
Missing Return Values	✅	✅
Fee on Transfer	❌	✅
Rebasing	❌	❌
Upgradable Tokens	✅	✅
Flash Mintable Tokens	✅	✅
Tokens with Blocklists	✅	✅
Pausable Tokens	❌	❌
Approval Race Protections	✅	✅
Revert on Approval To Zero Address	✅	✅
Revert on Zero Value Approvals	✅	✅
Revert on Zero Value Transfers	✅	✅
Multiple Token Addresses	✅	✅
Low Decimals	✅	✅
High Decimals	✅	✅
transferFrom with src == msg.sender	✅	✅
Non string metadata	✅	✅
Revert on Transfer to the Zero Address	✅	✅
No Revert on Failure	✅	✅
Revert on Large Approvals & Transfers	✅	✅
Code Injection Via Token Name	✅	✅
Unusual Permit Function	✅	✅
Transfer of less than amount	❌	❌
ERC-20 Representation of Native Currency	✅	✅

Recommendation

If you plan to add a new staking or reward token make sure it's compatible with the current staking contract.

[L-01] Unused imports

Description

There are a couple of unused imports which could be removed for better code clarity:

- https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/facets/StakingFacet.sol#L5
- https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/interfaces/IStaking.sol#L6
- https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibStaking.sol#L6
- https://github.com/ubiquity/ubiquity-dollar/blob/8fbfbe92b5403be031e6d092fd699b746fa13406/packages/contracts/src/dollar/libraries/LibStaking.sol#L8

Recommendation

Remove unused imports.

[L-02] `whenNotPaused` modifier not used

Description

There's the whenNotPaused modifier which is not used anywhere in the StakingFacet contract although there're places in the codebase where it is applied.

It makes sense to utilize the whenNotPaused modifier for all public state changing methods in the StakingFacet contract in order to mitigate losses in case of emergency.

Recommendation

Use the whenNotPaused modifier for all public state changing methods.

[L-03] Treasury griefing

Description

Malicious user can grief treasury to get 0 rewards if rewards allocation is triggered every block due to precision loss here.

Consider an example:

1. `governancePerBlock == 0.0000000001 ether && governanceTreasuryDivider == 1000000000`
2. User stakes 1 ether
3. 10 blocks pass
4. User unstakes 1 ether
5. At this point treasury has 1 UBQ token (which is expected)

Now consider an example when rewards are harvested each block:

1. `governancePerBlock == 0.0000000001 ether && governanceTreasuryDivider == 1000000000`
2. User stakes 1 ether
3. User collects rewards in 10 blocks in a row
4. User unstakes 1 ether
5. At this point treasury has 0 UBQ token (while expected value is 1)

PoC

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.19;

import "forge-std/console2.sol";
import {DiamondTestSetup} from "../diamond/DiamondTestSetup.sol";
import {UbiquityAlgorithmicDollarManager} from "../../src/deprecated/UbiquityAlgorithmicDollarManager.sol";
import {UbiquityGovernance} from "../../src/deprecated/UbiquityGovernance.sol";
import {MockERC20} from "../../src/dollar/mocks/MockERC20.sol";
import {LibStaking} from "../../src/dollar/libraries/LibStaking.sol";

contract ProtocolTest is DiamondTestSetup {
    UbiquityAlgorithmicDollarManager dollarManager;
    UbiquityGovernance rewardToken;
    MockERC20 rewardToken2;
    MockERC20 stakeToken;
    MockERC20 stakeToken2;

    address user = makeAddr("user");
    address user2 = makeAddr("user2");

    function setUp() public override {
        super.setUp();

        vm.prank(owner);
        dollarManager = new UbiquityAlgorithmicDollarManager(owner);

        vm.prank(owner);
        rewardToken = new UbiquityGovernance(address(dollarManager));

        stakeToken = new MockERC20("STK", "STK", 18);
        stakeToken2 = new MockERC20("STK2", "STK2", 6);
        rewardToken2 = new MockERC20("RWD2", "RWD2", 18);

        // staking setup
        vm.startPrank(admin);
        stakingFacet.setGovernancePerBlock(1 ether);
        stakingFacet.setGovernanceTreasuryDivider(5);
        stakingFacet.setStakingRewardToken(address(rewardToken));
        stakingFacet.setStakingStartBlock(block.number);
        vm.stopPrank();

        // owner grants diamond the "UBQ_MINTER_ROLE"
        // NOTICE: in production environment the diamond contract already has the "UBQ_MINTER_ROLE" role
        vm.prank(owner);
        dollarManager.grantRole(keccak256("UBQ_MINTER_ROLE"), address(diamond));

        // admin creates a new staking pool
        vm.startPrank(admin);
        stakingFacet.createStakingPool(
            100, // allocation points
            stakeToken,
            getAvailablePoolIds() // array of pool ids to update
        );
        vm.stopPrank();

        // mint 100 STK tokens to user
        stakeToken.mint(user, 100 ether);
        // mint 100 STK2 tokens to user
        stakeToken2.mint(user, 100 ether);
        // user approves diamond to spend STK tokens
        vm.prank(user);
        stakeToken.approve(address(diamond), type(uint256).max);
        // user approves diamond to spend STK2 tokens
        vm.prank(user);
        stakeToken2.approve(address(diamond), type(uint256).max);

        // mint 100 STK tokens to user2
        stakeToken.mint(user2, 100 ether);
        // mint 100 STK2 tokens to user2
        stakeToken2.mint(user2, 100 ether);
        // user2 approves diamond to spend STK tokens
        vm.prank(user2);
        stakeToken.approve(address(diamond), type(uint256).max);
        // user2 approves diamond to spend STK2 tokens
        vm.prank(user2);
        stakeToken2.approve(address(diamond), type(uint256).max);
    }

    function test_treasuryGrief() public {
        vm.startPrank(admin);
        stakingFacet.setGovernancePerBlock(0.0000000001 ether);
        stakingFacet.setGovernanceTreasuryDivider(1000000000);
        vm.stopPrank();

        vm.prank(user);
        stakingFacet.stake(0, 1 ether);

        // user collects rewards each block
        for(uint i = 0; i < 10; ++i) {
            vm.roll(block.number + 1);
            vm.prank(user);
            stakingFacet.unstake(0, 0);
        }

        vm.prank(user);
        stakingFacet.unstake(0, 1 ether);

        // treasury balance == 0 while the expected value should be 1 if user did't collect rewards each block
        console2.log("Treasury balance (UBQ):", rewardToken.balanceOf(admin));
    }
}
```

Recommendation

Increase precision on minting tokens to the treasury.

[L-04] `governanceTreasuryDivider` can't be set to 0

Description

There may be a situation when there's no more need in minting additional tokens to the treasury. In this case `governanceTreasuryDivider` should be set to 0.

There are a couple of issues:

1. `governanceTreasuryDivider` must always be greater than 0
2. Even if `governanceTreasuryDivider` was set to 0 then updateStakingPool would revert due to division by 0 here

Recommendation

Refactor the code and allow `governanceTreasuryDivider` to be 0.

[L-05] DoS when `treasuryAddress` is 0

Description

There may be a situation when the protocol decides to stop minting additional UBQ tokens to the treasury. One of the possible solutions could be to set `governanceTreasuryDivider` to 0 but it is not allowed (see `[L-04] governanceTreasuryDivider can't be set to 0`). Another option could be to set `treasuryAddress` to 0 address.

The issue with setting `treasuryAddress` to 0 address is that it leads to DoS of staking methods because UBQ token reverts on minting to 0 address.

Recommendation

Refactor the code and don't mint tokens to the treasury if its address is not set (i.e. equals to `address(0)`).

Disclaimer

About the author

Prepared by ryzhak.com.

Perform a comprehensive smart contract audit using advanced formal verification tools that identify even the most elusive and intricate bugs within smart contracts and mathematically prove the absence of security issues. All formal verification properties will be integrated into your standard deployment CI/CD pipelines, which helps reduce the number of bugs in already audited code, thereby lowering costs for future security assessments.