Competitive Security Assessment

Lumoz

Dec 21st, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	Lumoz
Platform & Language	Solidity & Go

Codebase	• https://github.com/ZKFair/zkfair-transaction-mining-contract
	  ◦ audit commit - bde8bbe769229e9cc3956b639e4082165f61ee22
	  ◦ final commit - c9c657a26b91db80ab41846faf7257a29ad68c75
	• https://github.com/ZKFair/zkfair-staking-contracts
	  ◦ audit commit - 64c6072f9c660f789e838dea4bc7c2c078d6bf1d
	  ◦ final commit - c017866f1c713968d12e8040431e740881b79ee8
	• https://github.com/ZKFair/zkfair-cdk-validium-contracts
	  ◦ audit commit - cecd53e0b1e39cd9df1a79215eedbbb636b4e0a7
	  ◦ final commit - cecd53e0b1e39cd9df1a79215eedbbb636b4e0a7
	• https://github.com/ZKFair/zkfair-cdk-validium-bridge-service
	  ◦ audit commit - 7d3d3bbe7d0c72bffdb19a129e4d6ec817d62819
	  ◦ final commit - 7d3d3bbe7d0c72bffdb19a129e4d6ec817d62819

Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
zkfair-transaction-mining-contract/contracts/RewardDistribution.sol	96728edbab51d4b21fff2bc0af1307a934700a39c1e3e95fb8bb02f94df050d6
zkfair-staking-contracts/contracts/ZKFStaking.sol	9cb361f40e8328c67943d2a8846c9732d75c2a575715c3c042a6452e7915f030
zkfair-staking-contracts/contracts/ZKFRewardContract.sol	dd24f4606bfcddb56a3b2b6d4b06b15c51efeb1507da45b0692bb76fcec64450
zkfair-cdk-validium-contracts/contracts/CDKValidium.sol	708c11b6182ff0e74ecc820a938f7792b7df6d67c2f3ed772ec3651d3aa753fb
zkfair-cdk-validium-contracts/contracts/verifiers/FflonkVerifier.sol	7b3d7f5eb4dad7c35a3673ec3e0059509918dd827b7120ea3d669a118bd692bc
zkfair-cdk-validium-contracts/contracts/PolygonZkEVMBridge.sol	cfa3867862f7630579168afee6b5ace567bc819c1f346d2437b2cecaf4de4440
zkfair-cdk-validium-contracts/contracts/CDKDataCommittee.sol	5e344883976750692c04d92f05f6a984262902d879db45478a0ef9507771f88d
zkfair-cdk-validium-contracts/contracts/lib/TokenWrapped.sol	421a434bb0b24efa710e151aeb60623f4482369562933e7beaedd395f9216bdf
zkfair-cdk-validium-contracts/contracts/lib/DepositContract.sol	a807f752e1297a2e2b7ade217975584e116f30acaba794980ebb47afdf428ea2
zkfair-cdk-validium-contracts/contracts/interfaces/IPolygonZkEVMBridge.sol	facf45a9d8ab6471abb8bb0d2b2841c4734651e77575b0f3267cddeb5eedd605
zkfair-cdk-validium-contracts/contracts/PolygonZkEVMGlobalExitRoot.sol	8b002ff5177c31dc39ca9e3daffaf818163d17d57f8abdeccd847e35d401a48a
zkfair-cdk-validium-contracts/contracts/interfaces/ICDKValidiumErrors.sol	942f19357eb0590991b3be1694fd9f878e823629975fae1770c87d662f123c31
zkfair-cdk-validium-contracts/contracts/deployment/CDKValidiumDeployer.sol	58914a665778cdd97cfc4f7fc6f562a536a9f88ac8fba70de40652c243996630
zkfair-cdk-validium-contracts/contracts/lib/EmergencyManager.sol	0d30c56c0f7a27f5f8f69fe40322c2f25e896b00159153682a8fc75a509dcd89

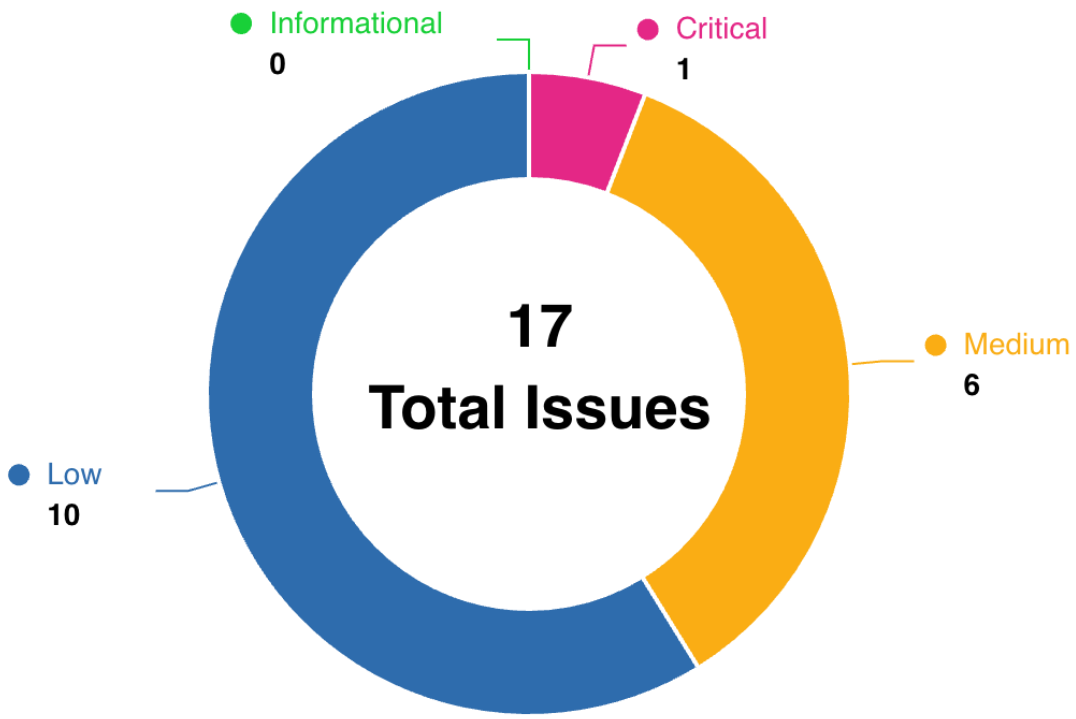
zkfair-cdk-validium-contracts/contracts/CDKValidiumTimelock.sol	b94238851a67a493f8367fa16d421d7ea8071f75c3de0a98193f733ce08a431f
zkfair-cdk-validium-contracts/contracts/PolygonZkEVMGlobalExitRootL2.sol	aa1c6879c6ff53b654c8400c7198efc8a60efd9a3b041fb71fd9c11cd892f123
zkfair-cdk-validium-contracts/contracts/interfaces/ICDKDataCommitteeErrors.sol	6db7ea096943fbd27589d2027c3944b3d3d527fb9c9fe4807d9e21b987b32e33
zkfair-cdk-validium-contracts/contracts/lib/GlobalExitRootLib.sol	6f880c1ffeab850e046488ab7fd45379ca628367b335c699a5c0906d01b6c9d1
zkfair-cdk-validium-contracts/contracts/interfaces/IBasePolygonZkEVMGlobalExitRoot.sol	68e6ee83953fb7eb6df407e9ddeddf3685af6e456e842934cbc373e7d33bc746
zkfair-cdk-validium-contracts/contracts/interfaces/IBridgeMessageReceiver.sol	55d499a259adf7778e04dbe18161a30213e33fd18d38ff75db8dbd7df9d58e1f
zkfair-cdk-validium-contracts/contracts/interfaces/IVerifierRollup.sol	ab4030bc19da8b28e581bfbdf8ab7ed4af98b87c7121af081a8922ce664d9f3c
zkfair-cdk-validium-contracts/contracts/interfaces/IPolygonZkEVMGlobalExitRoot.sol	98f8432bca9b822701b4993a6866f132a2c5a0c72ebf8a008db4e29cf7f854fc
zkfair-cdk-validium-contracts/contracts/interfaces/ICDKDataCommittee.sol	fdc3fb3dbfb5ef7797095535b0e0b97635c7effc5732d2e0bfdeb43dde28d237
zkfair-cdk-validium-bridge-service/bridgectrl/pb/query.pb.go	e874e09cc6f9ff949745ef60258184f7384f194730651faa23296f1b88476fc5
zkfair-cdk-validium-bridge-service/bridgectrl/pb/query.pb.gw.go	ec998c1184d5da849c582ef9c6e84be4a33b599643814a8c3f4aa01ff18ac593
zkfair-cdk-validium-bridge-service/synchronizer/synchronizer.go	3146f8e24e3345c838b0c907cf8c60f8e9a86b73ff8d870efe7873efe990768c
zkfair-cdk-validium-bridge-service/claimtxman/claimtxman.go	488c910a4cc312e3f00429e2bb3a51bc903462208b6c4b0b7005cbeb8ae68141
zkfair-cdk-validium-bridge-service/db/pgstorage/pgstorage.go	1af2fb1b6435648e41036ee2750676b5832f3ff09a7f3901fe7fbc389eef1589

zkfair-cdk-validium-bridge-service/etherman/etherman.go	c258cef458ffcfc9ba818824ee1718de5844aedc8a1c254602eb621297ac7e22
zkfair-cdk-validium-bridge-service/server/service.go	dbf4655d4619bb4224e1d7f602ea347741050320b86a9f30cc5daef1c0005b35
zkfair-cdk-validium-bridge-service/bridgectrl/pb/query_grpc.pb.go	47d756c9545bedbab23f8c9208c816bce45805580934c42cb6258e7a17d49457
zkfair-cdk-validium-bridge-service/synchronizer/mock_storage.go	5fe30fc39dd4a0992460c2738699cdd71b10402ecf31b59abce00fbb514f70f0
zkfair-cdk-validium-bridge-service/synchronizer/mock_dbtx.go	e96ba92e409351c8df61b2ce99013c4aa2724f3c709ecace7eb460373e86a6ca
zkfair-cdk-validium-bridge-service/utils/client.go	5c63adb6bd05a92ca39ad4d64891785958b92f9d9dc10b1a9d12fa332fe5e13b
zkfair-cdk-validium-bridge-service/cmd/run.go	6dc6bfacfc762643531c2874037ca40bf4f07aa46288dd5c97c1b2614e51ca3a
zkfair-cdk-validium-bridge-service/server/server.go	7ab59403ba476ecc69bc4cb3451830e7b58b30e7167d104e676affbc49577128
zkfair-cdk-validium-bridge-service/bridgectrl/merkletree.go	828eb5c089b27dd02e4e98029b5b1d97e770e0af7724f2c40eea68100806d7b8
zkfair-cdk-validium-bridge-service/synchronizer/mock_etherman.go	c752e09c53ef6ff24f7bfc64e7ea04044e31128032b5b4c15b1b985e1d98def5
zkfair-cdk-validium-bridge-service/etherman/simulated.go	5cdb42dadbf7265ef7a4f7e15052ab59ef6695411d18b9f19e812448f92375de
zkfair-cdk-validium-bridge-service/db/pgstorage/utils.go	ce94ea1098d888769aaf9eededad8748c74ebbb6047aa439b7d38250af0e060c
zkfair-cdk-validium-bridge-service/config/config.go	df10fc4189a7306c5ccbf7bcfeeb0745e8de6dc34f1ac604c922b72dcf026748
zkfair-cdk-validium-bridge-service/etherman/types.go	977effe2314d7abc2eeb9fd4875a7c235286f7c8e8bbf3ce2cb0e811b750a08c
zkfair-cdk-validium-bridge-service/claimtxman/types/monitoredtx.go	18905b60624a0100fbdd957a2df297a5ba2ac5fd16668fbc8a8cb688fd7d2941
zkfair-cdk-validium-bridge-service/synchronizer/mock_zkevmclient.go	5e17dea0c5e3c12125d91ea8728f9cbaddd2a57123967a72adbe27d146d651c0
zkfair-cdk-validium-bridge-service/bridgectrl/bridgectrl.go	72304bedaab7bc5bfab9c7d4a6995afd421ef555202ded765cd5b04883837260

zkfair-cdk-validium-bridge-service/cmd/main.go	a218b7b6dd5a4627b605b7c75b4f19721d4244a5030647804be127cb6f010f2e
zkfair-cdk-validium-bridge-service/config/network.go	efa074a4bcb956c21b481396ef9ad4994569961fc841906e5ad9c71ac1e3d241
zkfair-cdk-validium-bridge-service/config/default.go	eb55eae6fbb76c8b7f1a0ed13d675e34e3dc7a5ade192ad76ab720f01e8b68c4
zkfair-cdk-validium-bridge-service/synchronizer/interfaces.go	937f4d4feb4b18ec0debe45bb22e072088ad321a43c284e53eb740f58fffc201
zkfair-cdk-validium-bridge-service/synchronizer/mock_bridgectrl.go	061ec30a927b818bf8fa04a5a370ef221d7640227e8fa438ed713c30acd267fa
zkfair-cdk-validium-bridge-service/bridgectrl/hash.go	ba8c97bb72ba4cc1f100f97915a2b54911eeeac6e3f7f17cc348f62c48741220
zkfair-cdk-validium-bridge-service/db/storage.go	39aef4b134737c290f94e2bfb443ae22970d686e3a58060fb79f02231710f27e
zkfair-cdk-validium-bridge-service/claimtxman/interfaces.go	19b3916ae5cd63c6b08ea9569358989181b5c92b619ec8147b100f2ab7e8ba22
zkfair-cdk-validium-bridge-service/scripts/cmd/main.go	8438e24f1bd7d7c4905ffe0e36b0c6032bcdfe83fbc074ef4f7c0c65c62dbd6d
zkfair-cdk-validium-bridge-service/scripts/cmd/dependencies.go	1e0f6abc8641e57a8a9db99a55d60a35f25216a206ef1b2505447e67c0b9bfc7
zkfair-cdk-validium-bridge-service/server/interfaces.go	c126ae15f654a82c5fcd858ae134e5f27627bdcc6ea03cb39faf56b0096fc7da
zkfair-cdk-validium-bridge-service/version.go	f1d3795b88a7d9f85821f9fd09db46e836a52299d752feb8e476e02965f1c51f
zkfair-cdk-validium-bridge-service/utils/helpers.go	1395ecc56dd37f6afc3f6ffb1cfbf72986e73731a4cfff07320599d464ac10d7
zkfair-cdk-validium-bridge-service/claimtxman/config.go	033c299176ac1517392a9e84dc86adada86478a9b3344184e4efd9eb1ca0c521
zkfair-cdk-validium-bridge-service/bridgectrl/interfaces.go	49eb6c4c48876c3a24eaec0c43f993fba5c4a87c0c1588fadb3433269390408a
zkfair-cdk-validium-bridge-service/db/pgstorage/interfaces.go	c9fac1c39b61cea9329bab700ac38c6ef78ff43aebda6d69237c87cdafdd3373
zkfair-cdk-validium-bridge-service/server/config.go	b946ac2b5ce915b2ddb79ed24a44181c23b8cb09e8440ea3a13078536eded640

zkfair-cdk-validium-bridge-service/cmd/version.go	bdf6e83ec37e06604855b98a59e9e061dc4a780bd2060364d37b8b450cb4d931
zkfair-cdk-validium-bridge-service/db/config.go	5b3f016455849dfa20864028e805a828a985ef2cd4657237d702bf1064afaa28
zkfair-cdk-validium-bridge-service/utils/gerror/error.go	bde93816deb25e90798a33ff2d2553724d21bebda5ec7037771b2a54155227ff
zkfair-cdk-validium-bridge-service/db/pgstorage/config.go	79a60ad041803f91f7986b98e4633ee728dada49b495b0034786b5e079b4a159
zkfair-cdk-validium-bridge-service/synchronizer/config.go	215af41b6966997935269543b3ac64e272045cc9bf72cb268a1357437aac6de7
zkfair-cdk-validium-bridge-service/bridgectrl/config.go	1de0a55bba058d0486c2516def337658d2aed9db7438af7046a8dfb8d7b65769
zkfair-cdk-validium-bridge-service/etherman/config.go	ea46ebfe4a5bee1e92764e9d37ca152debe5d92e4c4a710933d41041bffd20ce

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
LMZ-1	`proposewMerkleRoot` can only be called once	Logical	Critical	Acknowledged	8olidity, 0xffchain
LMZ-2	Flawed Weight Update Mechanism in StakingContract's Deposit and Withdrawal Functions	Logical	Medium	Fixed	BradMoonUESTC
LMZ-3	Second PreImage Attack	Language Specific	Medium	Fixed	defsec

LMZ-4	Vulnerability in Withdraw Function Leading to Potential Over-withdrawal of Funds	Logical	Medium	Fixed	BradMoonUESTC
LMZ-5	Detection of Duplicate Signature Vulnerability in Smart Contract Multi-Signature Verification	Logical	Medium	Acknowledged	BradMoonUESTC
LMZ-6	Risk with Proposal and Review Authority Management in RewardDistribution Contract	Logical	Medium	Acknowledged	defsec
LMZ-7	Inconsistency in Merkle Tree State due to Post-Increment in Deposit Count	Logical	Medium	Acknowledged	BradMoonUESTC
LMZ-8	Calling deleted Values	Logical	Low	Fixed	0xffchain
LMZ-9	Vulnerability in Merkle Root Update Mechanism	Logical	Low	Acknowledged	BradMoonUESTC
LMZ-10	Incompatible with defliationary token	Logical	Low	Acknowledged	defsec, danielt
LMZ-11	Vulnerability in ERC20 Token Handling in `bridgeAsset` Function	Logical	Low	Acknowledged	BradMoonUESTC
LMZ-12	Users loose claim due to poor timing of sent rewards	Logical	Low	Acknowledged	0xffchain
LMZ-13	Use of transfer instead of call can run out of gas in some multi-sig wallets	Language Specific	Low	Acknowledged	defsec
LMZ-14	Use `disableInitializers`	Language Specific	Low	Acknowledged	defsec
LMZ-15	Use `call` instead of `transfer` to send ether	Code Style	Low	Acknowledged	0xffchain, 8olidity
LMZ-16	Vulnerability in Claim Function due to Unchecked Merkle Root	Privilege Related	Low	Fixed	BradMoonUESTC
LMZ-17	Use `safeTransfer` instead of `transfer`	Code Style	Low	Acknowledged	8olidity

LMZ-1: `proposewMerkleRoot` can only be called once

Category	Severity	Client Response	Contributor
Logical	Critical	Acknowledged	8olidity, 0xffchain

Code Reference

- code/zkfair-transaction-mining-contract/contracts/RewardDistribution.sol#L76-L81

```
76:reviewAuthority = _account;
77:     }
78:
79:     // Each week, the proposal authority calls to submit the merkle root for a new airdrop.
80:     function proposewMerkleRoot(bytes32 _merkleRoot) public {
81:         require(msg.sender == proposalAuthority);
```

Description

8olidity : In the RewardDistribution contract

- `proposewMerkleRoot(bytes32 _merkleRoot)`:
  The purpose of this function is to allow authorized users to propose a new Merkle root for subsequent reward distribution. The proposed Merkle root is stored in the `pendingMerkleRoot` variable, pending review.
- `reviewPendingMerkleRoot()`:
  This function is called by a privileged reviewer to review the currently pending Merkle root and, upon passing review, updates the stored master merkleRoot . This operation means that new reward distributions can be made based on this new root.

There is a major logic flaw in the existing contract implementation. Once the `reviewPendingMerkleRoot` function is executed and `pendingMerkleRoot` is successfully updated to `merkleRoot`, according to the contract logic, if `merkleRoot` is not the initial value `0x00`, the `proposewMerkleRoot` function will not be executed again. This will cause the contract to be unable to accept new Merkle root proposals after the first update, hindering the normal operation of the contract and the update of reward distribution.
0xffchain : Each week a new merkle root is needed for accounts to be able to claim rewards, but the mechanism to propose roots only allows the proposal of a new root once in the life time of the contract, after which a new proposal is impossible.

```
  function proposewMerkleRoot(bytes32 _merkleRoot) public {
        require(msg.sender == proposalAuthority);
        require(pendingMerkleRoot == 0x00);
        require(merkleRoot == 0x00);
        // @audit-issue since a merkle root has no delete funtionality , it means that the merkle ro
ot
        // can infact be only set once, which is from an empty state to a non empty state,
        // then after then all attempts at change is reverted.
        pendingMerkleRoot = _merkleRoot;
    }
```

as is seen in the `proposewMerkleRoot` function, the requirement `require(merkleRoot == 0x00);` requires that the merkleroot be empty at proposal. This only be the case after the first root is set. There is no delete functionality attached to the contract that allows for the deletion of a root value, only for the oveririding of the value when a new proposal is accepted, so this simply means that the root can only be set once in its lifetime which is its first value, and after that any other attemp to propose a root will fail since the merkle root is not empty.

Recommendation

8olidity :

```
  function proposewMerkleRoot(bytes32 _merkleRoot) public {
    require(msg.sender == proposalAuthority);
    require(pendingMerkleRoot == 0x00);
-   require(merkleRoot == 0x00);
    pendingMerkleRoot = _merkleRoot;
  }
```

0xffchain : Merkle root does not have to be empty to set a new value.

Client Response

Acknowledged,The business design is like this, the activity is only once

LMZ-2:Flawed Weight Update Mechanism in StakingContract's Deposit and Withdrawal Functions

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	BradMoonUESTC

Code Reference

- code/zkfair-staking-contracts/contracts/ZKFStaking.sol#L118

```
118:DepositInfo memory depositInfo = deposits[msg.sender][durations[_duration].index];
```

Description

BradMoonUESTC : The StakingContract smart contract demonstrates a significant logical vulnerability in its `deposit` and `withdraw` functions, specifically in the way it updates the `Weight` structure for users. This flaw stems from the contract's failure to accurately adjust the total weight (`totalWeight`) when users make new deposits or withdrawals.

In the `deposit` function, when a user adds funds to an existing duration, the contract recalculates the user's weight for that duration. However, it incorrectly updates the total weight by merely subtracting the old weight and adding the new one. This simplistic approach overlooks the fact that the weight of ongoing deposits should increase over time, leading to an underestimation of the total weight.

Similarly, in the `withdraw` function, the total weight is reduced by the weight of the withdrawn deposit. This does not take into account the potential increase in weight of other deposits due to the passage of time, potentially leading to an overestimation of the reduced weight.

The functions:

```
totalWeight.accountWeight -= weight.accountWeight;
...
totalWeight.accountWeight += weight.accountWeight;
```

and

```
weights[msg.sender][0].accountWeight -= _calculateWeight(depositInfo.amount, depositInfo.duration);
```

are indicative of this issue.

Recommendation

BradMoonUESTC : To rectify this vulnerability, a more dynamic and accurate weight calculation system is necessary. The following steps are recommended:

1. **Refine Weight Update Logic:** Modify the `deposit` and `withdraw` functions to include a more sophisticated method of calculating the total weight. This should involve recalculating the weights of all ongoing deposits whenever a new deposit or withdrawal occurs.
2. **Introduce Time-Dependent Weight Adjustments:** Implement logic to update the weights of ongoing deposits to reflect the passage of time, ensuring that the total weight is always an accurate representation of the user's current stake.

Client Response

Fixed,Weight and Deposit will not increase over time

LMZ-3:Second PreImage Attack

Category	Severity	Client Response	Contributor
Language Specific	Medium	Fixed	defsec

Code Reference

- code/zkfair-transaction-mining-contract/contracts/RewardDistribution.sol#L118

```
118:bytes32 node = keccak256(abi.encodePacked(index, msg.sender, amount));
```

Description

defsec : When using MerkleProof, it should avoid using leaf values that are 64 bytes long prior to hashing.

Recommendation

defsec : Use the following leaf encoding(https://github.com/OpenZeppelin/merkle-tree#leaf-hash) or which is the same, use keccak256(bytes.concat(keccak256(abi.encode(x,y,z))) instead of keccak256(abi.encodePacked(x,y,z)).

Client Response

Fixed,: https://github.com/ZKFair/zkfair-transaction-mining-contract/commit/2c1b234349c73d3cbdd11f0cfdd3e59145b3b71b

LMZ-4:Vulnerability in Withdraw Function Leading to Potential Over-withdrawal of Funds

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	BradMoonUESTC

Code Reference

- code/zkfair-staking-contracts/contracts/ZKFStaking.sol#L116-L134

```
116:function withdraw(uint256 _duration) external {
117:        require(durations[_duration].index != 0, "Invalid duration");
118:        DepositInfo memory depositInfo = deposits[msg.sender][durations[_duration].index];
119:        require(depositInfo.depositor == msg.sender, "Unauthorized withdrawal");
120:        require(depositInfo.amount > 0, "empty amount");
121:        require(block.timestamp >= depositInfo.timestamp + (depositInfo.duration * period), "Dep
osit is not matured yet");
122:
123:        deposits[msg.sender][0].amount -= depositInfo.amount;
124:        deposits[msg.sender][0].timestamp = block.timestamp;
125:        weights[msg.sender][0].accountWeight -= _calculateWeight(depositInfo.amount, depositInf
o.duration);
126:
127:        delete deposits[msg.sender][durations[_duration].index];
128:        delete weights[msg.sender][durations[_duration].index];
129:        uint256 unaffectedWeight = calculateDepositorWeight(msg.sender);
130:        bool result = token.transfer(msg.sender, depositInfo.amount);
131:        require(result, 'ZKFStaking: ZKF transfer failed.');
132:        emit Withdraw(depositInfo.depositor, _duration, depositInfo.amount, depositInfo.nonce);
133:        emit UpdateWeight(msg.sender, unaffectedWeight, weights[msg.sender][0].accountWeight, bl
ock.timestamp);
134:    }
```

Description

BradMoonUESTC : The smart contract contains a critical logical vulnerability in the `withdraw` function. This flaw allows a user to potentially withdraw more funds than they deposited, under certain conditions. Specifically, the contract fails to correctly correlate the total deposit amounts (`deposits[msg.sender][0]`) with the specific duration-based

deposits (`deposits[msg.sender][durations[_duration].index]`). When a user withdraws funds, the contract reduces the total deposit amount before deleting the specific duration deposit, without verifying if the withdrawn amount corresponds only to the specific duration deposit. This oversight allows a user to make withdrawals that cumulatively exceed their total deposited funds.

Steps to Exploit:

1. User A deposits tokens into the contract with a specific duration (e.g., 3 months) using the `deposit` function.
2. User A makes another deposit with a different duration (e.g., 1 month).
3. After the first deposit duration (3 months) matures, User A initiates a withdrawal.
4. The `withdraw` function deducts the amount from the total deposits (`deposits[msg.sender][0]`) without verifying its link to the specific duration deposit.
5. User A can then withdraw the remaining amount for the 1-month duration deposit, potentially withdrawing more than their total deposited amount.

Recommendation

BradMoonUESTC : To mitigate this vulnerability, it is recommended to add a validation step in the `withdraw` function that ensures the withdrawal amount is strictly correlated with the specific duration deposit. This can be achieved by implementing a check to confirm that the withdrawal amount does not exceed the amount in the specified duration deposit. Additionally, the total deposit amount should only be updated after this validation is passed. This adjustment will prevent users from withdrawing more than their actual deposited amounts for specific durations and safeguard against potential fund exploitation in the contract.

Client Response

fixed and commit
https://github.com/ZKFair/zkfair-staking-contracts/commit/c017866f1c713968d12e8040431e740881b79ee8

LMZ-5:Detection of Duplicate Signature Vulnerability in Smart Contract Multi-Signature Verification

Category	Severity	Client Response	Contributor
Logical	Medium	Acknowledged	BradMoonUESTC

Code Reference

- code/zkfair-cdk-validium-contracts/contracts/CDKDataCommittee.sol#L103-L149

```
103:function verifySignatures(
104:        bytes32 signedHash,
105:        bytes calldata signaturesAndAddrs
106:    ) external view {
107:        // pre-check: byte array size
108:        uint splitByte = _SIGNATURE_SIZE * requiredAmountOfSignatures;
109:        if(
110:            signaturesAndAddrs.length < splitByte ||
111:            (signaturesAndAddrs.length - splitByte) % _ADDR_SIZE != 0
112:        ) {
113:            revert UnexpectedAddrsAndSignaturesSize();
114:        }
115:
116:        // hash the addresses part of the byte array and check that it's equal to committe hash
117:        if (
118:            keccak256(signaturesAndAddrs[splitByte:]) !=
119:            committeeHash
120:        ) {
121:            revert UnexpectedCommitteeHash();
122:        }
123:
124:        // recover addresses from signatures and check that are part of the committee
125:        uint lastAddrIndexUsed;
126:        uint addrsLen = (signaturesAndAddrs.length - splitByte) / _ADDR_SIZE;
127:        for (uint i = 0; i < requiredAmountOfSignatures; i++) {
128:            address currentSigner = ECDSA.recover(
129:                signedHash,
130:                signaturesAndAddrs[i*_SIGNATURE_SIZE : i*_SIGNATURE_SIZE + _SIGNATURE_SIZE]
131:            );
132:            bool currentSignerIsPartOfCommittee = false;
133:            for (uint j = lastAddrIndexUsed; j < addrsLen; j++) {
134:                uint currentAddresStartingByte = splitByte + j*_ADDR_SIZE;
135:                address committeeAddr = address(bytes20(signaturesAndAddrs[
136:                    currentAddresStartingByte :
137:                    currentAddresStartingByte + _ADDR_SIZE
138:                ]));
139:                if (committeeAddr == currentSigner) {
140:                    lastAddrIndexUsed = j+1;
141:                    currentSignerIsPartOfCommittee = true;
142:                    break;
```

```
143:                 }
144:             }
145:             if (!currentSignerIsPartOfCommittee) {
146:                 revert CommitteeAddressDoesntExist();
147:             }
148:         }
149:     }
```

Description

BradMoonUESTC : The smart contract code provided for the `verifySignatures` function contains a critical vulnerability related to the handling of signatures in a multi-signature verification process. The original implementation does not incorporate a mechanism to check for duplicate signers. This oversight allows a single committee member to submit multiple signatures, potentially leading to the manipulation of governance voting results or decisions requiring multiple authentications.

Furthermore, the code incorrectly handles the recovery of committee addresses from the `signaturesAndAddrs` byte array. This flaw can result in incorrect address recovery due to improper slicing of bytes and casting them directly to an `address`. These vulnerabilities can significantly undermine the integrity and intended decentralization of the decision-making process, potentially leading to unilateral control or protocol insolvency if used in critical decision-making or financial transactions.

Recommendation

BradMoonUESTC : 1. **Implement Duplicate Signer Check**: Introduce a temporary `signers` array to store addresses recovered from the signatures. Prior to adding a recovered address to this array, the code should check if the address is already present. If a duplicate is found, the function should revert the transaction with a `DuplicateSignerDetected` error, ensuring each signature corresponds to a unique committee member.

Client Response

Acknowledged,Problem confirmed, please provide specific modification code

LMZ-6:Risk with Proposal and Review Authority Management in RewardDistribution Contract

Category	Severity	Client Response	Contributor
Logical	Medium	Acknowledged	defsec

Code Reference

- code/zkfair-transaction-mining-contract/contracts/RewardDistribution.sol#L7-L74

```
7:contract RewardDistribution is OwnableUpgradeable {
8:    uint public totalOutput; //Total Mining.
9:    uint public firstStartTime; // first start time
10:    uint public claimStartTime; // claim start time
11:    uint public claimEndInterval; // claim end interval
12:
13:    address public zkfTokenAddress;
14:    uint public totalDistributedReward; // total Distributed Reward
15:
16:    // history
17:    address[] public allRewardsAddress;
18:    mapping(address => uint) public rewardHistory;
19:
20:    // merkleRoot
21:    bytes32 public merkleRoot;
22:    bytes32 public pendingMerkleRoot;
23:
24:    // admin address which can propose adding a new merkle root
25:    address public proposalAuthority;
26:    // admin address which approves or rejects a proposed merkle root
27:    address public reviewAuthority;
28:
29:    event Claimed(
30:        uint256 index,
31:        address account,
32:        uint256 amount
33:    );
34:
35:    event NewAddFeeRecordEvent(
36:        address indexed receiveAddress
37:    );
38:
39:    // This is a packed array of booleans.
40:    mapping(uint256 => uint256) private claimedBitMap;
41:
42:    modifier onlyValidAddress(address addr) {
43:        require(addr != address(0), "Illegal address");
44:        _;
45:    }
46:
47:    function initialize(
48:        address _initialOwner,
```

```
49:         address _zkfTokenAddress,
50:         address _proposalAuthority,
51:         address _reviewAuthority,
52:         uint256 _totalOutput,
53:         uint256 _claimEndInterval
54:     ) external onlyValidAddress(_initialOwner)
55:     onlyValidAddress(_zkfTokenAddress)
56:     onlyValidAddress(_proposalAuthority)
57:     onlyValidAddress(_reviewAuthority) virtual initializer {
58:         firstStartTime = block.timestamp;
59:         zkfTokenAddress = _zkfTokenAddress;
60:         proposalAuthority = _proposalAuthority;
61:         reviewAuthority = _reviewAuthority;
62:         totalOutput = _totalOutput;
63:         claimEndInterval = _claimEndInterval;
64:         // Initialize OZ contracts
65:         __Ownable_init_unchained(_initialOwner);
66:     }
67:
68:     function setProposalAuthority(address _account) public onlyValidAddress(_account) {
69:         require(msg.sender == proposalAuthority);
70:         proposalAuthority = _account;
71:     }
72:
73:     function setReviewAuthority(address _account) public onlyValidAddress(_account) {
```

Description

defsec : The RewardDistribution contract currently allows proposalAuthority and reviewAuthority to manage the merkle root proposal and review process. This design presents a security risk if either of these authorities is compromised. As it stands, these authorities have the power to propose and approve new merkle roots, which are critical to the integrity of the reward distribution process. If a malicious actor gains control of these addresses, they could manipulate the reward distribution by approving fraudulent merkle roots. Typically, such critical functionalities are managed by the contract owner or through a more robust access control mechanism.

If either proposalAuthority or reviewAuthority is compromised, it could lead to unauthorized or fraudulent changes in the merkle root, affecting the integrity of the reward distribution.

Recommendation

defsec : Utilize OpenZeppelin's AccessControl for managing these critical roles. Require owner approval for changing proposalAuthority and reviewAuthority. This adds an additional layer of security.

Client Response

Acknowledged,Both addresses are managed through multi-signature, and the permissions are sufficiently dispersed.

LMZ-7:Inconsistency in Merkle Tree State due to Post-Increment in Deposit Count

Category	Severity	Client Response	Contributor
Logical	Medium	Acknowledged	BradMoonUESTC

Code Reference

- code/zkfair-cdk-validium-contracts/contracts/lib/DepositContract.sol#L65-L89

```
65:function _deposit(bytes32 leafHash) internal {
66:        bytes32 node = leafHash;
67:
68:        // Avoid overflowing the Merkle tree (and prevent edge case in computing `_branch`)
69:        if (depositCount >= _MAX_DEPOSIT_COUNT) {
70:            revert MerkleTreeFull();
71:        }
72:
73:        // Add deposit data root to Merkle tree (update a single `_branch` node)
74:        uint256 size = ++depositCount;
75:        for (
76:            uint256 height = 0;
77:            height < _DEPOSIT_CONTRACT_TREE_DEPTH;
78:            height++
79:        ) {
80:            if (((size >> height) & 1) == 1) {
81:                _branch[height] = node;
82:                return;
83:            }
84:            node = keccak256(abi.encodePacked(_branch[height], node));
85:        }
86:        // As the loop should always end prematurely with the `return` statement,
87:        // this code should be unreachable. We assert `false` just to be safe.
88:        assert(false);
89:    }
```

Description

BradMoonUESTC : The identified vulnerability exists in the `_deposit` function of the `DepositContract`. The function is intended to update a Merkle tree branch each time a new leaf node is added. The logical flaw arises when the `depositCount` reaches its maximum limit (`_MAX_DEPOSIT_COUNT`). The contract is designed to revert with a `MerkleTreeFull` error when this limit is exceeded. However, due to the post-increment of `depositCount` (`++depositCount`), the Merkle tree branch update occurs even after reaching this limit. This leads to a state where the Merkle tree structure is updated beyond its intended capacity, causing inconsistency in the tree's state. Such a state misalignment can potentially be exploited to disrupt the contract's operations, as it relies on the integrity and consistency of the Merkle tree for its functions.

Recommendation

BradMoonUESTC : To rectify this vulnerability, it is recommended to adjust the incrementation of `depositCount` so that it occurs only after the Merkle tree branch update is successfully executed and validated. This ensures that the Merkle tree is not updated if the maximum deposit count is reached, maintaining the tree's integrity and preventing any inconsistency.

The recommended change in the `_deposit` function is as follows:

```
function _deposit(bytes32 leafHash) internal {
    bytes32 node = leafHash;

    // Prevent Merkle tree overflow
    if (depositCount >= _MAX_DEPOSIT_COUNT) {
        revert MerkleTreeFull();
    }

    // Update Merkle tree branch
    uint256 size = depositCount;
    for (uint256 height = 0; height < _DEPOSIT_CONTRACT_TREE_DEPTH; height++) {
        if (((size >> height) & 1) == 1) {
            _branch[height] = node;
            break;
        }
        node = keccak256(abi.encodePacked(_branch[height], node));
    }

    // Increment deposit count only after successful tree update
    depositCount++;
}
```

By implementing this change, the contract ensures that the Merkle tree is only updated when it is within its operational limits, thus preserving the consistency and reliability of the contract's core functionality.

Client Response

Acknowledged,polygon official code, need to confirm with the official

LMZ-8:Calling deleted Values

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	0xffchain

Code Reference

- code/zkfair-staking-contracts/contracts/ZKFStaking.sol#L116

```
116:function withdraw(uint256 _duration) external {
```

Description

0xffchain :

```
    function withdraw(uint256 _duration) external { // @note no amount specified
        require(durations[_duration].index != 0, "Invalid duration");
        DepositInfo storage depositInfo = deposits[msg.sender][durations[_duration].index];
        require(depositInfo.depositor == msg.sender, "Unauthorized withdrawal");
        require(depositInfo.amount > 0, "empty amount");

        require(block.timestamp >= depositInfo.timestamp + (depositInfo.duration * period), "Deposit
is not matured yet");
        deposits[msg.sender][0].amount -= depositInfo.amount;
        deposits[msg.sender][0].timestamp = block.timestamp;
        weights[msg.sender][0].accountWeight -= _calculateWeight(depositInfo.amount, depositInfo.dur
ation);

        delete deposits[msg.sender][durations[_duration].index];
        delete weights[msg.sender][durations[_duration].index];
        uint256 unaffectedWeight = calculateDepositorWeight(msg.sender);
        bool result = token.transfer(msg.sender, depositInfo.amount);

        require(result, 'ZKFStaking: ZKF transfer failed.');
        emit Withdraw(depositInfo.depositor, _duration, depositInfo.amount, depositInfo.nonce);
        // @audit-issue calling deleted values from storage
        emit UpdateWeight(msg.sender, unaffectedWeight, weights[msg.sender][0].accountWeight, block.
timestamp);
    }
```

In the withdraw function above the array structs depositInfo and Weights are deleted like so:

```
delete deposits[msg.sender][durations[_duration].index];
delete weights[msg.sender][durations[_duration].index];
```

But the deleted values/structs are still called from storage in here

```
    emit Withdraw(depositInfo.depositor, _duration, depositInfo.amount, depositInfo.nonce);
```

When attempting to emit an event, it means that this event values will be the defualt values, which will be:

`depositInfo.depositor = 0x` `depositInfo.amount = 0` `depositInfo.nonce = 0`

Therefore storing false event values on the blockchain log and thus providing invalid data to offchain systems.

The same is also the challenge in

`uint256 unaffectedWeight = calculateDepositorWeight(msg.sender);`

The called function calls the storage values of the deleted variable, although this bears no consequences as opposed to the first and the values returned will be zero and thus no wieght added.

```
    function calculateDepositorWeight(address depositor) public returns (uint256 unaffectedWeight) {
        uint256 today = block.timestamp - block.timestamp % period;
        for (uint8 i = 1; i < 9; i++) {
            Weight memory weight = weights[depositor][i];
            if (weight.update_at < today) {
                unaffectedWeight += weight.accountWeight;
                // @audit-issue will pass when weight deleted, but account weight will also be zero.
            }
        }
        return unaffectedWeight;
    }
```

Recommendation

0xffchain : depositInfo should be copied to memory, then values called from it.

Client Response

Fixed and commit

https://github.com/ZKFair/zkfair-staking-contracts/commit/c017866f1c713968d12e8040431e740881b79ee8

LMZ-9:Vulnerability in Merkle Root Update Mechanism

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	BradMoonUESTC

Code Reference

- code/zkfair-transaction-mining-contract/contracts/RewardDistribution.sol#L113-L137

```
113:function claim(uint256 index, uint256 amount, bytes32[] calldata merkleProof) public {
114:        require(!isClaimed(index), 'MerkleDistributor: Drop already claimed.');
115:        require(amount > 0 && amount <= totalOutput, 'Invalid parameter');
116:
117:        // Verify the merkle proof.
118:        bytes32 node = keccak256(abi.encodePacked(index, msg.sender, amount));
119:        require(verify(merkleProof, merkleRoot, node), 'MerkleDistributor: Invalid proof.');
120:
121:        require(claimStartTime + claimEndInterval >= block.timestamp, 'claim end');
122:        // Mark it claimed and send the token.
123:        _setClaimed(index);
124:
125:        require(totalDistributedReward + amount <= totalOutput, 'Distribution has ended.');
126:
127:        bool bResult = IERC20(zkfTokenAddress).transfer(msg.sender, amount);
128:        require(bResult, 'ZKF erc20 transfer failed.');
129:
130:        if(rewardHistory[msg.sender] == 0) {
131:            allRewardsAddress.push(msg.sender);
132:            emit NewAddFeeRecordEvent(msg.sender);
133:        }
134:        rewardHistory[msg.sender] += amount;
135:        totalDistributedReward += amount;
136:        emit Claimed(index, msg.sender, amount);
137:    }
```

Description

BradMoonUESTC : The identified vulnerability lies within the `proposewMerkleRoot` and `reviewPendingMerkleRoot` functions of the smart contract. The flaw arises due to the lack of restriction on the frequency of calls to `proposewMerkleRoot` by the `proposalAuthority`. The `proposalAuthority` can set a new `pendingMerkleRoot` multiple times before the `reviewAuthority` has a chance to review and approve it. As a result, the `proposalAuthority` has the potential to propose a new `pendingMerkleRoot` after the initial proposal and before the review process, thus changing the `merkleRoot` without appropriate oversight. This oversight can lead to incorrect or malicious distribution of funds, as the `merkleRoot` is crucial for determining the validity of claims.

Recommendation

BradMoonUESTC : To mitigate this vulnerability, it is recommended to introduce a check in the `proposewMerkleRoot` function that ensures a new `merkleRoot` cannot be proposed until the previous `pendingMerkleRoot` has been reviewed and approved or rejected by the `reviewAuthority`. This check will enforce that at any given time, there is only one `pendingMerkleRoot` awaiting review, thus preventing the potential for malicious or unintended updates by the `proposalAuthority`. Additionally, implementing a time lock or delay between the proposal and review processes could provide additional security, ensuring adequate time for any necessary audits or validations.

Client Response

Acknowledged,

```
// Each week, the proposal authority calls to submit the merkle root for a new airdrop.
    function proposewMerkleRoot(bytes32 _merkleRoot) public {
        require(msg.sender == proposalAuthority);
        require(pendingMerkleRoot == 0x00); // There is already a check here. If pendingMerkleRoot h
as data, it will not be submitted.
        require(merkleRoot == 0x00); // The check here ensures that merkleRoot can only be updated o
nce
        pendingMerkleRoot = _merkleRoot;
    }
```

Therefore, I think the two issues reported above should not exist. I will wait for your new reply and then conduct corresponding analysis and follow-up.

LMZ-10:Incompatible with defliationary token

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	defsec, danielt

Code Reference

- code/zkfair-staking-contracts/contracts/ZKFStaking.sol#L76

```
76:bool result = token.transferFrom(msg.sender, address(this), _amount);
```

Description

defsec : The contract does not handle deflationary tokens correctly. Deflationary tokens are tokens that decrease their supply over time, usually by taking a small percentage of each transfer and burning it or redistributing it. This means that the actual amount received from a transfer or transferFrom call can be less than the amount specified in the call.
In the deposit function, the contract assumes that the full amount will be transferred from the user to the contract. However, if token is a deflationary token, the actual amount received could be less than amount.
ZKFStaking.sol#L79

```
function deposit(uint256 _duration, uint256 _amount) external {
        require(durations[_duration].index != 0, "Invalid duration");
        require(_amount > 0, "Amount must be greater than 0");

        bool result = token.transferFrom(msg.sender, address(this), _amount);
        require(result, 'ZKFStaking: ZKF transfer failed.');
}
```

danielt : In the `ZKFStaking` contract, the `deposit` function does not check the actually received token from the user. For example, if a user deposits 100 deflationary token A into the `ZKFStaking` contract, and the contract actually receives 90 deflationary token A, it is bad for the protocol because the contract records that the user deposited 100 deflationary tokens.

Recommendation

defsec : The contract does not handle deflationary tokens correctly. Deflationary tokens are tokens that decrease their supply over time, usually by taking a small percentage of each transfer and burning it or redistributing it. This means that the actual amount received from a transfer or transferFrom call can be less than the amount specified in the call.
In the deposit function, the contract assumes that the full amount will be transferred from the user to the contract. However, if token is a deflationary token, the actual amount received could be less than amount.

ZKFStaking.sol#L79

```
function deposit(uint256 _duration, uint256 _amount) external {
        require(durations[_duration].index != 0, "Invalid duration");
        require(_amount > 0, "Amount must be greater than 0");

        bool result = token.transferFrom(msg.sender, address(this), _amount);
        require(result, 'ZKFStaking: ZKF transfer failed.');
}
```

danielt : Recommend checking the actually received tokens from the user and recording the real amount of token received for the user.

Client Response

Acknowledged,This ERC20 is deployed by ourselves based on the standard ERC20. During the event, the total amount of tokens issued will not change.

LMZ-11:Vulnerability in ERC20 Token Handling in `bridgeAsset` Function

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	BradMoonUESTC

Code Reference

- code/zkfair-cdk-validium-contracts/contracts/PolygonZkEVMBridge.sol#L175-L301

```
175:function bridgeAsset(
176:        uint32 destinationNetwork,
177:        address destinationAddress,
178:        uint256 amount,
179:        address token,
180:        bool forceUpdateGlobalExitRoot,
181:        bytes calldata permitData
182:    ) public payable virtual ifNotEmergencyState nonReentrant {
183:        if (
184:            destinationNetwork == networkID ||
185:            destinationNetwork >= _CURRENT_SUPPORTED_NETWORKS
186:        ) {
187:            revert DestinationNetworkInvalid();
188:        }
189:
190:        address originTokenAddress;
191:        uint32 originNetwork;
192:        bytes memory metadata;
193:        uint256 leafAmount = amount;
194:
195:        if (token == address(0)) {
196:            // Ether transfer
197:            if ((msg.value - bridgeFee) != amount) {
198:                revert AmountDoesNotMatchMsgValue();
199:            }
200:
201:            // Ether is treated as ether from mainnet
202:            originNetwork = _MAINNET_NETWORK_ID;
203:        } else {
204:            // Check whether msg.value is equal to the cross-chain handling fee
205:            if (msg.value != bridgeFee) {
206:                revert AmountDoesNotMatchMsgValue();
207:            }
208:
209:            TokenInformation memory tokenInfo = wrappedTokenToTokenInfo[token];
210:
211:            if (tokenInfo.originTokenAddress != address(0)) {
212:                // The token is a wrapped token from another network
213:
214:                // Burn tokens
215:                TokenWrapped(token).burn(msg.sender, amount);
216:
```

```
217:                 originTokenAddress = tokenInfo.originTokenAddress;
218:                 originNetwork = tokenInfo.originNetwork;
219:             } else {
220:                 // In order to support fee tokens check the amount received, not the transferred
221:                 uint256 balanceBefore = IERC20Upgradeable(token).balanceOf(
222:                     address(this)
223:                 );
224:                 IERC20Upgradeable(token).safeTransferFrom(
225:                     msg.sender,
226:                     address(this),
227:                     amount
228:                 );
229:                 uint256 balanceAfter = IERC20Upgradeable(token).balanceOf(
230:                     address(this)
231:                 );
232:
233:                 // Override leafAmount with the received amount
234:                 leafAmount = balanceAfter - balanceBefore;
235:
236:                 originTokenAddress = token;
237:                 originNetwork = networkID;
238:
239:                 // Encode metadata
240:                 metadata = abi.encode(
241:                     _safeName(token),
242:                     _safeSymbol(token),
243:                     _safeDecimals(token)
244:                 );
245:             }
246:         }
247:
248:         if (gasTokenAddress != address (0)) { // is gas token
249:             if (token == address(0)) {
250:                 originTokenAddress = gasTokenAddress;
251:                 metadata = gasTokenMetadata;
252:                 if (networkID != _MAINNET_NETWORK_ID) { // is l2 -> l1,
253:                     leafAmount /= gasTokenDecimalDiffFactor;
254:                     if (leafAmount == 0) {
255:                         revert AmountTooSmall();
256:                     }
257:                 }
258:
```

```
259:             } else if (originTokenAddress == gasTokenAddress) {
260:                 originTokenAddress = address(0);
261:                 if (networkID == _MAINNET_NETWORK_ID) { // is l1 -> l2
262:                     leafAmount *= gasTokenDecimalDiffFactor;
263:                 }
264:             }
265:         }
266: 
267:         emit BridgeEvent(
268:             _LEAF_TYPE_ASSET,
269:             originNetwork,
270:             originTokenAddress,
271:             destinationNetwork,
272:             destinationAddress,
273:             leafAmount,
274:             metadata,
275:             uint32(depositCount)
276:         );
277: 
278:         _deposit(
279:             getLeafValue(
280:                 _LEAF_TYPE_ASSET,
281:                 originNetwork,
282:                 originTokenAddress,
283:                 destinationNetwork,
284:                 destinationAddress,
285:                 leafAmount,
286:                 keccak256(metadata)
287:             )
288:         );
289: 
290:         if (feeAddress != address(0) && bridgeFee > 0) {
291:             (bool success, ) = feeAddress.call{value: bridgeFee}(new bytes(0));
292:             if (!success) {
293:                 revert EtherTransferFailed();
294:             }
295:         }
296: 
297:         // Update the new root to the global exit root manager if set by the user
298:         if (forceUpdateGlobalExitRoot) {
299:             _updateGlobalExitRoot();
300:         }
301:     }
```

Description

BradMoonUESTC : The identified vulnerability in the `bridgeAsset` function of the provided smart contract presents a significant security risk involving ERC20 token transfers in a cross-chain bridge mechanism. This vulnerability specifically arises due to inadequate handling of ERC20 tokens that implement transfer fees or deflationary mechanisms.

Key Points:

- **Function Affected:** `bridgeAsset`.
- **Issue:** The contract calculates the amount of ERC20 tokens received (`leafAmount`) by measuring the balance difference before and after the `safeTransferFrom` call. This approach does not account for ERC20 tokens that deduct a fee or burn a percentage of the tokens during transfer.
- **Exploitable Scenario:** An attacker can use a deflationary or fee-charging ERC20 token to exploit this vulnerability. By initiating a bridge transfer with such a token, the contract erroneously assumes the full transfer amount is received, leading to a discrepancy between the actual and recorded token amounts.
- **Impact:** This allows an attacker to effectively bridge more tokens than transferred, potentially inflating the token supply on the destination network and leading to direct theft of assets.

Recommendation

BradMoonUESTC : 1. **Accurate Token Transfer Verification:**

- Implement a mechanism to accurately verify the actual amount of ERC20 tokens received post-transfer. This could involve querying the token balance of the contract both before and after the transfer, then confirming the expected decrease in balance.

2. **Handling Fee-charging and Deflationary Tokens:**
    - Add checks to ensure that the contract can handle tokens with transfer fees or deflationary features correctly. This may include integrating a method to query the token's transfer fee percentage or burn rate and adjusting the calculations accordingly.

Client Response

Acknowledged,polygon official code, need to confirm with the official

LMZ-12:Users loose claim due to poor timing of sent rewards

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	0xffchain

Code Reference

- code/zkfair-staking-contracts/contracts/ZKFRewardContract.sol#L80-L94

```
80:function claimReward(uint256 amount, bytes32[] memory proof) external {
81:        uint256 today = block.timestamp - block.timestamp % period;
82:        require(rootUpdatedAt > today, 'Rewards are being calculated, please try again late');
83:        require(claims[msg.sender].timestamp < today, "You already claimed your reward, please tr
y again tomorrow");
84:        _verify(msg.sender, amount, proof);
85:        uint256 contractBalance = address(this).balance;
86:        if (amount >= contractBalance) {
87:            amount = contractBalance;
88:        }
89:        claims[msg.sender].amount += amount;
90:        claims[msg.sender].timestamp = block.timestamp;
91:        payable(msg.sender).transfer(amount); // send reward
92:        emit ClaimReward(msg.sender, amount, block.timestamp, claims[msg.sender].amount);
93:
94:    }
```

Description

0xffchain : a user might call the contract when the root has been updated but the eth value not sent yet, through the recieve function, this will mark his transacton as recieved where if the balance of the pool is less than the amount or zero, it still marks the transaction as fulfiled and the has lost out on potential rewards due to timing. While the next users are made whole when the recieve function executes. there are many ways such scenario can play out to disadvantage the user.

```
  if (amount >= contractBalance) {
        amount = contractBalance;
    }
```

Since updating both the proof and adding rewards to the contract is a two step process, and both allows only update once in a day, it means that any claim transaction that is executed inbetween the updating of a root and sending rewards to the contract might loose its claims.

POC

1. Its a new day, a new root is proposed by the proposer, the reviewers sees this and accepts after making its own due deligence.
2. Alice sees that its new claim now is 50eth,
3. She makes a request to claim its 50eth, but the pool balance is <50Eth, as the rewards has not been supplied to the contract, taking note that the reward sent to the contract is also capped at once daily like the root.
4. Since the rewards has not been sent or is in the mempool and Alice Transaction gets executed first, it means that Alice inevitably gets less than the reward it meant to get or zero, depending on the contract balance at execution.

Recommendation

0xffchain : Balance should also be updated before a user can make claim, or a mechanism to make sure the claims in the system is equal to the value available to be claimed.

Client Response

Acknowledged,There is a problem with the code. The corresponding developer should check whether the balance of the contract is consistent with the requirements in the setProposalAuthority method. Currently, the recharge of this address will be carried out before the event starts and is mainly controlled through manual review.

LMZ-13:Use of transfer instead of call can run out of gas in some multi-sig wallets

Category	Severity	Client Response	Contributor
Language Specific	Low	Acknowledged	defsec

Code Reference

- code/zkfair-staking-contracts/contracts/ZKFRewardContract.sol#L91

```
91:payable(msg.sender).transfer(amount); // send reward
```

Description

defsec : Using `transfer` instead of `call` for sending ether may lead to the transaction running out of gas in some multi-signature wallets, such as Gnosis.

This is because `transfer` is limited to 2300 gas to prevent reentrancy, which is just enough to cover the transaction, but it is not enough to perform additional operations most multi-signature wallet contracts do.

```
    function claimReward(uint256 amount, bytes32[] memory proof) external {
        uint256 today = block.timestamp - block.timestamp % period;
        require(rootUpdatedAt > today, 'Rewards are being calculated, please try again late');
        require(claims[msg.sender].timestamp < today, "You already claimed your reward, please try a
gain tomorrow");
        _verify(msg.sender, amount, proof);
        uint256 contractBalance = address(this).balance;
        if (amount >= contractBalance) {
            amount = contractBalance;
        }
        claims[msg.sender].amount += amount;
        claims[msg.sender].timestamp = block.timestamp;
        payable(msg.sender).transfer(amount); // send reward
        emit ClaimReward(msg.sender, amount, block.timestamp, claims[msg.sender].amount);

    }
```

Recommendation

defsec : Use `call` instead of transfer.

Client Response

Acknowledged,From the usage scenario, currently only direct regular collection by users is considered, and the possible problems of multi-signature are temporarily ignored.

LMZ-14:Use `disableInitializers`

Category	Severity	Client Response	Contributor
Language Specific	Low	Acknowledged	defsec

Code Reference

- code/zkfair-staking-contracts/contracts/ZKFStaking.sol#L38
- code/zkfair-staking-contracts/contracts/ZKFRewardContract.sol#L39
- code/zkfair-transaction-mining-contract/contracts/RewardDistribution.sol#L48

```
38:constructor(address _tokenAddress) {

39:constructor(address _proposalAuthority, address _reviewAuthority, address _rewardSponsor) onlyVal
idAddress(_proposalAuthority) onlyValidAddress(_reviewAuthority) onlyValidAddress(_rewardSponsor) {

48:function initialize(
```

Description

defsec : The current implementations are missing the `_disableInitializers()` function call in the constructors. Thus, an attacker can initialize the implementation. Usually, the initialized implementation has no direct impact on the proxy itself; however, it can be exploited in a phishing attack. In rare cases, the implementation might be mutable and may have an impact on the proxy.

Recommendation

defsec : It is recommended to call `_disableInitializers` within the contract's constructor to prevent the implementation from being initialized.

Client Response

Acknowledged,Problem confirmed

LMZ-15:Use `call` instead of `transfer` to send ether

Category	Severity	Client Response	Contributor
Code Style	Low	Acknowledged	0xffchain, 8olidity

Code Reference

- code/zkfair-staking-contracts/contracts/ZKFRewardContract.sol#L80-L94
- code/zkfair-staking-contracts/contracts/ZKFRewardContract.sol#L91

```
80:function claimReward(uint256 amount, bytes32[] memory proof) external {
81:        uint256 today = block.timestamp - block.timestamp % period;
82:        require(rootUpdatedAt > today, 'Rewards are being calculated, please try again late');
83:        require(claims[msg.sender].timestamp < today, "You already claimed your reward, please tr
y again tomorrow");
84:        _verify(msg.sender, amount, proof);
85:        uint256 contractBalance = address(this).balance;
86:        if (amount >= contractBalance) {
87:            amount = contractBalance;
88:        }
89:        claims[msg.sender].amount += amount;
90:        claims[msg.sender].timestamp = block.timestamp;
91:        payable(msg.sender).transfer(amount); // send reward
92:        emit ClaimReward(msg.sender, amount, block.timestamp, claims[msg.sender].amount);
93:
94:    }

91:payable(msg.sender).transfer(amount); // send reward
```

Description

0xffchain : When sending ETH, use call() instead of transfer(). The transfer() function only allows the recipient to use 2300 gas and sload opcode already cost 800 gas. If the recipient needs more than that, transfers will fail. In the future gas costs might change increasing the likelihood of that happening. If this happens it means the user can not withdraw its claim causing a possible DOS for the user for that day and thus loosing out on its claim. And if the recieving account is a proxy contract, it might not recieve it correctly.

8olidity : In both of the withdraw functions, transfer() is used for native ETH withdrawal. The transfer() and send() functions forward a fixed amount of 2300 gas. Historically, it has often been recommended to use these functions for value transfers to guard against reentrancy attacks. However, the gas cost of EVM instructions may change significantly

during hard forks which may break already deployed contract systems that make fixed assumptions about gas costs. For example. EIP 1884 broke several existing smart contracts due to a cost increase of the SLOAD instruction.

The use of the deprecated transfer() function for an address will inevitably make the transaction fail when:

The claimer smart contract does not implement a payable function. The claimer smart contract does implement a payable fallback which uses more than 2300 gas unit. The claimer smart contract implements a payable fallback function that needs less than 2300 gas units but is called through proxy, raising the call's gas usage above 2300. Additionally, using higher than 2300 gas might be mandatory for some multisig wallets.

Recommendation

0xffchain : Use call and not transfer. There is no chance of reentrancy on the code here since the code implements the Checks-Effects-Interactions Pattern, so any attempt to reenter the code will fail, cause the timestamp has been updated and greater than today.
8olidity : Use call() instead of transfer().

Client Response

Acknowledged,First, our contract is for very short-term use, and the contract supports upgrades. If such an extreme problem occurs, many projects on the current chain may also be affected.

LMZ-16:Vulnerability in Claim Function due to Unchecked Merkle Root

Category	Severity	Client Response	Contributor
Privilege Related	Low	Fixed	BradMoonUESTC

Code Reference

- code/zkfair-transaction-mining-contract/contracts/RewardDistribution.sol#L89-L97
- code/zkfair-transaction-mining-contract/contracts/RewardDistribution.sol#L113-L137

```
89:function reviewPendingMerkleRoot(bool _approved) public {
90:         require(msg.sender == reviewAuthority);
91:         require(pendingMerkleRoot != 0x00);
92:         if (_approved) {
93:             merkleRoot = pendingMerkleRoot;
94:             claimStartTime = block.timestamp;
95:         }
96:         delete pendingMerkleRoot;
97:     }

113:function claim(uint256 index, uint256 amount, bytes32[] calldata merkleProof) public {
114:         require(!isClaimed(index), 'MerkleDistributor: Drop already claimed.');
115:         require(amount > 0 && amount <= totalOutput, 'Invalid parameter');
116:
117:         // Verify the merkle proof.
118:         bytes32 node = keccak256(abi.encodePacked(index, msg.sender, amount));
119:         require(verify(merkleProof, merkleRoot, node), 'MerkleDistributor: Invalid proof.');
120:
121:         require(claimStartTime + claimEndInterval >= block.timestamp, 'claim end');
122:         // Mark it claimed and send the token.
123:         _setClaimed(index);
124:
125:         require(totalDistributedReward + amount <= totalOutput, 'Distribution has ended.');
126:
127:         bool bResult = IERC20(zkfTokenAddress).transfer(msg.sender, amount);
128:         require(bResult, 'ZKF erc20 transfer failed.');
129:
130:         if(rewardHistory[msg.sender] == 0) {
131:             allRewardsAddress.push(msg.sender);
132:             emit NewAddFeeRecordEvent(msg.sender);
133:         }
134:         rewardHistory[msg.sender] += amount;
135:         totalDistributedReward += amount;
136:         emit Claimed(index, msg.sender, amount);
137:     }
```

Description

BradMoonUESTC : The identified vulnerability exists within the `claim` function of the smart contract. This function fails to check whether the `merkleRoot` is set to a non-zero value before allowing claims to be processed. In its current state, `merkleRoot` can be reset to 0x00 by the `reviewPendingMerkleRoot` function if the `reviewAuthority` sets `_approved` to false. This oversight allows the potential for claims to be made and processed when no valid `merkleRoot` is present, leading to unauthorized or erroneous claims. This flaw could result in the misallocation or freezing of funds intended for legitimate claimants.

Recommendation

BradMoonUESTC : To mitigate this vulnerability, it is recommended to include a validation check in the `claim` function to ensure that `merkleRoot` is set to a valid (non-zero) value before proceeding with any claims processing. This check would prevent the function from processing claims unless a legitimate and valid `merkleRoot` is established, thereby safeguarding against the possibility of unauthorized or incorrect claims being approved. Implementing this safeguard ensures that the contract operates as intended and protects the integrity of the reward distribution mechanism.

Client Response

Fixed https://github.com/ZKFair/zkfair-transaction-mining-contract/commit/95fdc3c8476ce2efb033fb7c1a4214447bb1ac9e

LMZ-17:Use `safeTransfer` instead of `transfer`

Category	Severity	Client Response	Contributor
Code Style	Low	Acknowledged	8olidity

Code Reference

- code/zkfair-staking-contracts/contracts/ZKFStaking.sol#L75-L77
- code/zkfair-transaction-mining-contract/contracts/RewardDistribution.sol#L126-L128
- code/zkfair-staking-contracts/contracts/ZKFStaking.sol#L129-L131

```
75:bool result = token.transferFrom(msg.sender, address(this), _amount);
76:         require(result, 'ZKFStaking: ZKF transfer failed.');

126:bool bResult = IERC20(zkfTokenAddress).transfer(msg.sender, amount);
127:         require(bResult, 'ZKF erc20 transfer failed.');

129:uint256 unaffectedWeight = calculateDepositorWeight(msg.sender);
130:         bool result = token.transfer(msg.sender, depositInfo.amount);
131:         require(result, 'ZKFStaking: ZKF transfer failed.');
```

Description

8olidity : Tokens not compliant with the ERC20 specification could return false from the transfer function call to indicate the transfer fails, while the calling contract would not notice the failure if the return value is not checked. Checking the return value is a requirement, as written in the EIP-20 specification:

```
Callers MUST handle false from returns (bool success). Callers MUST NOT assume that false is never r
eturned!
```

```
zkfair-transaction-mining-contract\contracts\RewardDistribution.sol:
  126
  127:          bool bResult = IERC20(zkfTokenAddress).transfer(msg.sender, amount);
  128           require(bResult, 'ZKF erc20 transfer failed.');

zkfair-staking-contracts\contracts\ZKFStaking.sol:
  75
  76:          bool result = token.transferFrom(msg.sender, address(this), _amount);
  77           require(result, 'ZKFStaking: ZKF transfer failed.');

zkfair-staking-contracts\contracts\ZKFStaking.sol:
  129           uint256 unaffectedWeight = calculateDepositorWeight(msg.sender);
  130:          bool result = token.transfer(msg.sender, depositInfo.amount);
  131           require(result, 'ZKFStaking: ZKF transfer failed.');
```

Recommendation

8olidity : Use `safeTransfer` instead of `transfer`

Client Response

Acknowledged,The tokens distributed here are deployed by us using standard ERC20

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.