Competitive Security Assessment

Mufex

Jul 11th, 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

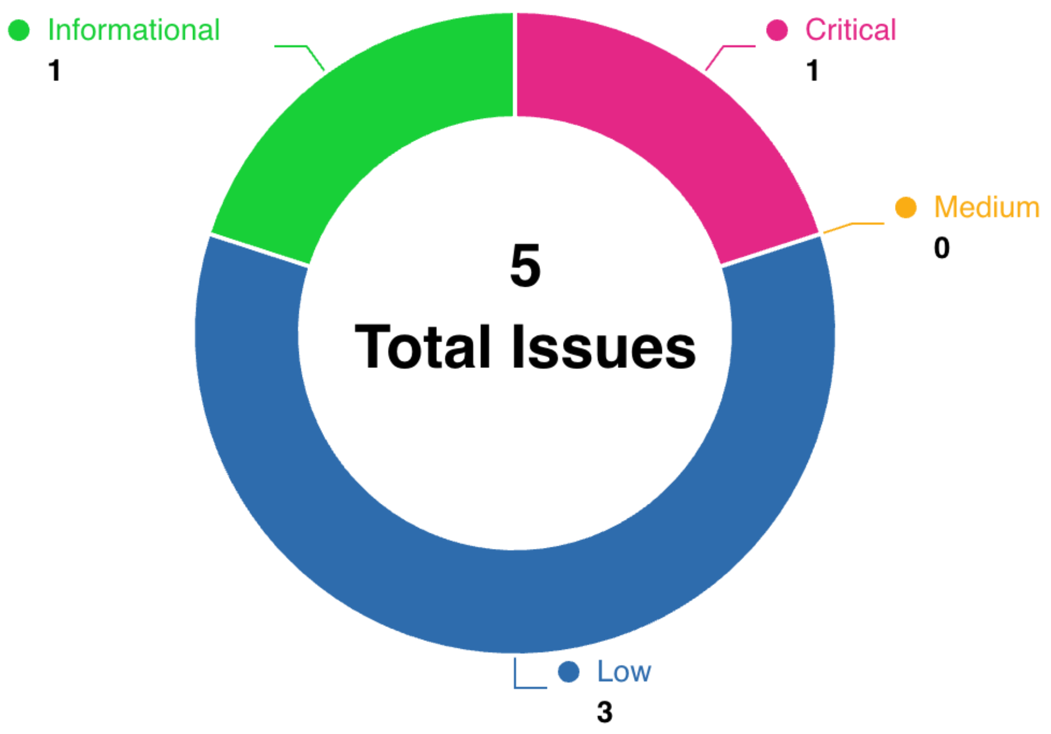
Project Detail

Project Name	Mufex
Platform & Language	Solidity
Codebase	• https://github.com/MUFEX-Exchange/smart-contract • audit commit - 2b7b95417098376b3e69c9a17cd6406213db8ee0 • final commit - f5f1288af4656e598a93bebcdce23933c6f8da13
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	Commit Hash
contracts/MainTreasury.sol	2b7b95417098376b3e69c9a17cd6406213db8ee0
contracts/Verifier.sol	2b7b95417098376b3e69c9a17cd6406213db8ee0
contracts/BaseTreasury.sol	2b7b95417098376b3e69c9a17cd6406213db8ee0
contracts/libraries/Pairing.sol	2b7b95417098376b3e69c9a17cd6406213db8ee0
contracts/DepositWalletFactory.sol	2b7b95417098376b3e69c9a17cd6406213db8ee0
contracts/interfaces/IMainTreasury.sol	2b7b95417098376b3e69c9a17cd6406213db8ee0
contracts/DepositWallet.sol	2b7b95417098376b3e69c9a17cd6406213db8ee0
contracts/libraries/MiMC.sol	2b7b95417098376b3e69c9a17cd6406213db8ee0
contracts/libraries/TransferHelper.sol	2b7b95417098376b3e69c9a17cd6406213db8ee0
contracts/Ownable.sol	2b7b95417098376b3e69c9a17cd6406213db8ee0
contracts/libraries/MerkleProof.sol	2b7b95417098376b3e69c9a17cd6406213db8ee0
contracts/interfaces/IERC20.sol	2b7b95417098376b3e69c9a17cd6406213db8ee0
contracts/interfaces/ITreasury.sol	2b7b95417098376b3e69c9a17cd6406213db8ee0
contracts/interfaces/IDepositWallet.sol	2b7b95417098376b3e69c9a17cd6406213db8ee0
contracts/interfaces/IDepositWalletFactory.sol	2b7b95417098376b3e69c9a17cd6406213db8ee0
contracts/HotTreasury.sol	2b7b95417098376b3e69c9a17cd6406213db8ee0

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
MUF-1	Inconsistent check condition in function `updateZKP()` and `generalWithdraw()`	Logical	Critical	Fixed	Yaodao
MUF-2	verifier does not constrain the relationship between IC and input length.	DOS	Medium	Declined	LiRiu
MUF-3	`batchWithdrawETH()` may fail	DOS	Low	Fixed	8olidity
MUF-4	`batchCollectTokens()` not checking array length	Code Style	Low	Fixed	8olidity

MUF-5	No native tokens transferred in function `collectETH()`	Logical	Low	Declined	Yaodao
MUF-6	Incompatibility With Deflationary Tokens	Logical	Low	Declined	Yaodao
MUF-7	The `forceWithdrawOpened` cannot be set to false	Logical	Low	Declined	8olidity
MUF-8	Inconsistent code implementation and comments in library `MiMC`	Logical	Low	Fixed	Yaodao
MUF-9	Unlocked Pragma Version	Code Style	Informational	Declined	Yaodao
MUF-10	Gas Optimization: Variables that could be declared as immutable	Gas Optimization	Informational	Declined	Yaodao
MUF-11	Gas Optimization: Lack of check to save gas	Gas Optimization	Informational	Declined	Yaodao
MUF-12	Gas Optimization: Use calldata instead of memory	Gas Optimization	Informational	Declined	Yaodao
MUF-13	code redundancy at `TransferHelper.sol`	Code Style	Informational	Declined	8olidity
MUF-14	Gas Optimization: Improve loop code to reduce gas consumption.	Gas Optimization	Informational	Declined	LiRiu
MUF-15	Gas Optimization: Cache array length out of the loop to save gas	Gas Optimization	Informational	Fixed	Yaodao

MUF-1:Inconsistent check condition in function `updateZKP()` and `generalWithdraw()`

Category	Severity	Status	Contributor
Logical	Critical	Fixed	Yaodao

Code Reference

- code/contracts/MainTreasury.sol#L77-L90
- code/contracts/MainTreasury.sol#L141-L143

```
77:             require(getWithdrawFinished[token], "last withdraw not finish yet");
78:             getWithdrawFinished[token] = false;
79:
80:             if (token == ETH) {
81:                 balanceOfThis = address(this).balance;
82:             } else {
83:                 balanceOfThis = IERC20(token).balanceOf(address(this));
84:             }
85:             require(balanceOfThis >= newTotalBalances[i] + newTotalWithdraws[i], "not enough balance");
86:
87:             getBalanceRoot[token] = newBalanceRoots[i];
88:             getWithdrawRoot[token] = newWithdrawRoots[i];
89:             getTotalBalance[token] = newTotalBalances[i];
90:             getTotalWithdraw[token] = newTotalWithdraws[i];

141:         getWithdrawn[token] += amount;
142:         require(getWithdrawn[token] <= getTotalWithdraw[token], "over totalWithdraw");
143:         if (getWithdrawn[token] == getTotalWithdraw[token]) getWithdrawFinished[token] = true;
```

Description

Yaodao : The variable `getWithdrawn[token]` is only updated in the function `generalWithdraw()`, the value of the variable `getWithdrawn[token]` will add the withdraw amount and should be smaller than the `getTotalWithdraw[token]`. So the variable `getWithdrawn[token]` will get bigger and bigger.

```
        getWithdrawn[token] += amount;
```

The variable `getTotalWithdraw[token]` is updated in the function `updateZKP()` and the function `updateZKP()` can only be called when the `getWithdrawn[token]` is equal to the `getTotalWithdraw[token]`. The following check in the function `updateZKP()` will check the `newTotalBalances[i]` and `newTotalWithdraws[i]` is smaller than the `balanceOfThis` which is the balance of this token in the contract. The new value of `getTotalWithdraw[token]` will be the `newTotalWithdraws[i]` which is smaller than the balance.

```
        require(getWithdrawFinished[token], "last withdraw not finish yet");
        getWithdrawFinished[token] = false;

        if (token == ETH) {
            balanceOfThis = address(this).balance;
        } else {
            balanceOfThis = IERC20(token).balanceOf(address(this));
        }
        require(balanceOfThis >= newTotalBalances[i] + newTotalWithdraws[i], "not enough balance");

        getBalanceRoot[token] = newBalanceRoots[i];
        getWithdrawRoot[token] = newWithdrawRoots[i];
        getTotalBalance[token] = newTotalBalances[i];
        getTotalWithdraw[token] = newTotalWithdraws[i];

        WithdrawnInfo storage withdrawnInfo = getWithdrawnInfo[token];
```

However, in the function `generalWithdraw()`, the value of `getWithdrawn[token]` should be smaller than the `getTotalWithdraw[token]`. As a result, as `getWithdrawn[token]` grows larger, the new value of `getTotalWithdraw[token]` which can call `updateZKP()` success will not meet the check in the function `generalWithdraw()`, or the value of `getTotalWithdraw[token]` which can meet the check in the function `generalWithdraw()` can't call `updateZKP()` success.

```
            getWithdrawn[token] += amount;
            require(getWithdrawn[token] <= getTotalWithdraw[token], "over totalWithdraw");
            if (getWithdrawn[token] == getTotalWithdraw[token]) getWithdrawFinished[token] = true;
```

As a result, the two check is inconsistent as the `getWithdrawn[token]` is not reset.

For example:

User a deposit 1000 ETH into the contract and User b deposit 1000 ETH too. The `getTotalWithdraw[ETH]` should be 2000. Then user a and user b withdraw the 2000 ETH. The value of `getWithdrawn[ETH]` will be 2000 and the `getWithdrawFinished[ETH]` will be true. And then the user a deposit 1000 ETH and the user b deposit 1000 ETH. The balance of ETH is 2000. So the max value of `newTotalWithdraws[i]` is 2000. However, the new `getTotalWithdraw[ETH]` should be 4000 to meet the user to withdraw their deposit.

Recommendation

Yaodao : Recommend updating the value of `getWithdrawn[token]` in the function `updateZKP()` or updating the check in the `updateZKP()` to make sure that the new `getTotalWithdraw[token]` can meet the check in the function `generalWithdraw()`.

Client Response

Fixed

MUF-2:verifier does not constrain the relationship between IC and input length.

Category	Severity	Status	Contributor
DOS	**Medium**	**Declined**	**LiRiu**

Code Reference

- code/contracts/Verifier.sol#L69-L120

```
    function updateZKP(
        uint64 newZkpId,
        address[] calldata tokens,
        uint256[] calldata newBalanceRoots,
        uint256[] calldata newWithdrawRoots,
        uint256[] calldata newTotalBalances,
        uint256[] calldata newTotalWithdraws
    ) external override onlyVerifierSet {
        require(msg.sender == verifier, "forbidden");
        require(!forceWithdrawOpened, "force withdraw opened");
        require(
            tokens.length == newBalanceRoots.length &&
            newBalanceRoots.length == newWithdrawRoots.length &&
            newWithdrawRoots.length == newTotalBalances.length &&
            newTotalBalances.length == newTotalWithdraws.length,
            "length not the same"
        );

        uint256 balanceOfThis;
        address token;
        for (uint256 i = 0; i < tokens.length; i++) {
            token = tokens[i];
            require(getWithdrawFinished[token], "last withdraw not finish yet");
            getWithdrawFinished[token] = false;

            if (token == ETH) {
                balanceOfThis = address(this).balance;
            } else {
                balanceOfThis = IERC20(token).balanceOf(address(this));
            }
            require(balanceOfThis >= newTotalBalances[i] + newTotalWithdraws[i], "not enough balance");

            getBalanceRoot[token] = newBalanceRoots[i];
            getWithdrawRoot[token] = newWithdrawRoots[i];
            getTotalBalance[token] = newTotalBalances[i];
            getTotalWithdraw[token] = newTotalWithdraws[i];

            WithdrawnInfo storage withdrawnInfo = getWithdrawnInfo[token];
            // clear claimed records
            for (uint256 j = 0; j < withdrawnInfo.allGeneralWithdrawnIndex.length; j++) {
                delete withdrawnInfo.generalWithdrawnBitMap[withdrawnInfo.allGeneralWithdrawnIndex[j]];
            }
            delete withdrawnInfo.allGeneralWithdrawnIndex;
        }

        require(newZkpId > zkpId, "old zkp");
        zkpId = newZkpId;
        lastUpdateTime = block.timestamp;

        emit ZKPUpdated(newZkpId, tokens, newBalanceRoots, newWithdrawRoots, newTotalBalances, newTotalWithdraws);
    }
```

Description

LiRiu : In groth16, the length of VK.IC should be equal to the input length plus one. the length of the input should be constrained by verifyProof. If the input is constructed incorrectly, it will not pass the validation.

Recommendation

LiRiu : Add the following constraints in the verifyProof function.

```
require(vk.IC.length == input.length + 1, "verifier-IC-length-mismatch");
```

Client Response

declined, Vulnerabilities need to be confirmed again, whether they really exist

MUF-3: `batchWithdrawETH()` may fail

Category	Severity	Status	Contributor
DOS	Low	Fixed	8olidity

Code Reference

- code/contracts/BaseTreasury.sol#L51-L62

```
        uint256[2] memory a,
        uint256[2][2] memory b,
        uint256[2] memory c,
        uint256[4] memory input
    ) public view returns (bool r) {

        Proof memory proof;
        proof.A = Pairing.G1Point(a[0], a[1]);
        proof.B = Pairing.G2Point([b[0][0], b[0][1]], [b[1][0], b[1][1]]);
        proof.C = Pairing.G1Point(c[0], c[1]);

        VerifyingKey memory vk = verifyingKey();

        // Compute the linear combination vk_x
        Pairing.G1Point memory vk_x = Pairing.G1Point(0, 0);

        // Make sure that proof.A, B, and C are each less than the prime q
        require(proof.A.X < PRIME_Q, "verifier-aX-gte-prime-q");
        require(proof.A.Y < PRIME_Q, "verifier-aY-gte-prime-q");

        require(proof.B.X[0] < PRIME_Q, "verifier-bX0-gte-prime-q");
        require(proof.B.Y[0] < PRIME_Q, "verifier-bY0-gte-prime-q");

        require(proof.B.X[1] < PRIME_Q, "verifier-bX1-gte-prime-q");
        require(proof.B.Y[1] < PRIME_Q, "verifier-bY1-gte-prime-q");

        require(proof.C.X < PRIME_Q, "verifier-cX-gte-prime-q");
        require(proof.C.Y < PRIME_Q, "verifier-cY-gte-prime-q");

        // Make sure that every input is less than the snark scalar field
        for (uint256 i = 0; i < input.length; i++) {
             require(input[i] < SNARK_SCALAR_FIELD,"verifier-gte-snark-scalar-field");
             vk_x = Pairing.plus(vk_x, Pairing.scalar_mul(vk.IC[i + 1], input[i]));
         }

         vk_x = Pairing.plus(vk_x, vk.IC[0]);

         return Pairing.pairing(
             Pairing.negate(proof.A),
             proof.B,
             vk.alfa1,
             vk.beta2,
            vk_x,
            vk.gamma2,
            proof.C,
            vk.delta2
        );
    }

    function submit(
        uint64 zkpId,
        uint256[] memory BeforeAccountTreeRoot,
```

Description

8olidity : `batchWithdrawETH()` will traverse the `recipients` array and send a certain amount of ETH to each recipient, but if the `balance` of the contract is less than the sum of the amounts, the function will fail to execute. Because in `_withdrawETH()`, it is only judged whether the amount is greater than 0, as follows

```
require(amount > 0, "zero amount");
```

Instead of judging whether the balance of the contract itself is greater than the amount like in `_withdrawToken()`

```
require(amount > 0, "zero amount");
require(IERC20(token).balanceOf(address(this)) >= amount, "balance not enough");
```

poc

```
uint sum  = 0;
for (uint256 i = 0; i < amounts.length; i++) {
        sum += amounts[i]
}
```

when `address(this).balance < sum`, `batchWithdrawETH()` may fail

Recommendation

8olidity : It is recommended to judge that `address(this).balance` is greater than the sum of `amounts[]` in `batchWithdrawETH()`

Client Response

Fixed

MUF-4: `batchCollectTokens()` not checking array length

Category	Severity	Status	Contributor
Code Style	Low	Fixed	8olidity

Code Reference

- code/contracts/DepositWalletFactory.sol#L72

```
    function batchWithdrawETH(
        address[] calldata recipients,
        uint256[] calldata amounts,
        string[] calldata requestIds
    ) external override onlyOperator {
        require(
            recipients.length == amounts.length &&
            recipients.length == requestIds.length, "length not the same");
        for (uint256 i = 0; i < recipients.length; i++) {
            _withdrawETH(recipients[i], amounts[i], requestIds[i]);
        }
    }
```

Description

8olidity : The `batchCollectTokens()` function is missing to check whether the lengths of the `wallets`, `tokens`, and `requestIds` arrays are consistent. In traversing the `wallets` array, the `tokens` and `requestIds` arrays are also traversed, so it is also necessary to check whether the array lengths are the same, like `batchCollectETH` and `batchCreateWallets`.

poc:

```
function batchCollectTokens(address[] calldata wallets, address[] calldata tokens, string[] calldata
requestIds) external override onlyOperator {
        address[] memory tokens_ = new address[](1);
        string[] memory requestIds_ = new string[](1);
        for (uint256 i = 0; i < wallets.length; i++) {
            DepositWallet wallet = DepositWallet(payable(wallets[i]));
            tokens_[0] = tokens[i];
            requestIds_[0] = requestIds[i];
            wallet.collectTokens(tokens_, requestIds_);
        }
        emit BatchCollectTokens(wallets, tokens, requestIds);
    }
```

Recommendation

8olidity :

```
function batchCollectTokens(address[] calldata wallets, address[] calldata tokens, string[] calldata
requestIds) external override onlyOperator {
+         require(wallets.length == tokens.length && tokens.length == requestIds.length);
        address[] memory tokens_ = new address[](1);
        string[] memory requestIds_ = new string[](1);
        for (uint256 i = 0; i < wallets.length; i++) {
            DepositWallet wallet = DepositWallet(payable(wallets[i]));
            tokens_[0] = tokens[i];
            requestIds_[0] = requestIds[i];
            wallet.collectTokens(tokens_, requestIds_);
        }
        emit BatchCollectTokens(wallets, tokens, requestIds);
    }
```

Client Response

Fixed

MUF-5:No native tokens transferred in function `collectETH()`

Category	Severity	Status	Contributor
Logical	Low	Declined	Yaodao

Code Reference

- code/contracts/DepositWallet.sol#L14-L17
- code/contracts/DepositWallet.sol#L37-L41

```
        address[] memory tokens_ = new address[](1);
```

Description

Yaodao : According to the codes in the contract `DepositWallet`, the function `collectETH()` is used to transfer all the ETH deposit in the contract to the treasury. And the function `collectETH()` is not declear with the keyword `payable`.

```
    function collectETH(string calldata requestId) external override {
        uint256 balance = address(this).balance;
        TransferHelper.safeTransferETH(treasury, balance);
        emit EtherCollected(treasury, balance, requestId);
    }
```

However, there is a `receive()` in the contract `DepositWallet` and there are no other payable functions in the contract. The function `receive()` will transfer the native tokens to treasury directly. As a result, there will be no native tokens in the contract forever and there will be no native tokens to transfer in function `collectETH()`.

Recommendation

Yaodao : Recommend updating the logic in the function `receive()`.

Client Response

Declined, Duì wǒmen méiyǒu rènhé ānquán fēngxiǎn, gǎidòng zhège dehuà, nà yìwèizhe yào xiūgǎi DepositWallet héyuē, nà yònghù rùjīn dìzhǐ jiù huì biànle, zhège duì wǒmen yǐngxiǎng fǎn'ér dà, suǒyǐ jiù bù zuò gǎidòngle
There is no security risk for us. Changing this means modifying the DepositWallet contract, and the user's deposit address will change. This will have a greater impact on us, so we will not make any changes

MUF-6:Incompatibility With Deflationary Tokens

Category	Severity	Status	Contributor
Logical	Low	Declined	Yaodao

Code Reference

- code/contracts/BaseTreasury.sol#L36-L41

```
        uint64 zkpId_ = zkpId;
        // Verify the merkle proof.
        uint256[] memory msgs = new uint256[](9);
        msgs[0] = zkpId_;
        msgs[1] = index;
        msgs[2] = withdrawId;
        msgs[3] = accountId;
        msgs[4] = uint256(uint160(account));
        msgs[5] = uint256(uint160(to));
        msgs[6] = uint256(uint160(token));
        msgs[7] = withdrawType;
        msgs[8] = amount;
        uint256 node = MiMC.Hash(msgs);
        require(MerkleProof.verify(proof, getWithdrawRoot[token], node), "Invalid proof");

        uint64 zkpId_ = zkpId;
        // Verify the merkle proof.
        uint256[] memory msgs = new uint256[](5);
        msgs[0] = index;
        msgs[1] = accountId;
        msgs[2] = uint256(uint160(msg.sender));
        msgs[3] = uint256(uint160(token));
        msgs[4] = equity;
        uint256 node = MiMC.Hash(msgs);
        require(MerkleProof.verify(proof, getBalanceRoot[token], node), "Invalid proof");
```

Description

Yaodao : When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. As a result, an inconsistency in the amount will occur and the transaction may fail due to the validation checks. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee) to the target contract, only 90 tokens actually arrive to the contract.

The function `depositToken()` is used to deposit the users tokens and report the data via the emit events. However, the amount used in the event is the transferred amount instead of the real amount transferred into the contract.

```
    function depositToken(address token, uint256 amount) external override {
        require(token != address(0), "zero address");
        require(amount > 0, "deposit amount is zero");
        TransferHelper.safeTransferFrom(token, msg.sender, address(this), amount);
        emit TokenDeposited(token, msg.sender, amount);
    }
```

As a result, the fees will not be recorded and the protocol will loss these fees.

Recommendation

Yaodao : Recommend regulating the set of tokens supported and adding necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Client Response

Declined, Bugs are real, but we don't support deflationary tokens, no need to fix them.

MUF-7:The `forceWithdrawOpened` cannot be set to false

Category	Severity	Status	Contributor
Logical	Low	Declined	8olidity

Code Reference

- code/contracts/MainTreasury.sol#L175

```
    receive() external payable {
        TransferHelper.safeTransferETH(treasury, msg.value);
        emit EtherCollected(treasury, msg.value, "");
    }

    function collectETH(string calldata requestId) external override {
        uint256 balance = address(this).balance;
        TransferHelper.safeTransferETH(treasury, balance);
        emit EtherCollected(treasury, balance, requestId);
    }
```

Description

8olidity : When calling `forceWithdraw()`, you can set `forceWithdraw` to true, but once `forceWithdrawOpened` is set to `true`, you cannot call `updateZKP()`

```
require(!forceWithdrawOpened, "force withdraw opened");
```

But with no other code in the contract to set `forceWithdrawOpened` to `false`, the contract will never be able to call `updateZKP()`

poc: when forceWithdrawOpened == true

Recommendation

8olidity : Add a function that can set forceWithdrawOpened to false

Client Response

Declined, Vulnerability does not exist, our business scenario does not support

MUF-8:Inconsistent code implementation and comments in library `MiMC`

Category	Severity	Status	Contributor
Logical	Low	Fixed	Yaodao

Code Reference

- code/contracts/libraries/MiMC.sol#L20-L52

```
36:    function depositToken(address token, uint256 amount) external override {
37:        require(token != address(0), "zero address");
38:        require(amount > 0, "deposit amount is zero");
39:        TransferHelper.safeTransferFrom(token, msg.sender, address(this), amount);
40:        emit TokenDeposited(token, msg.sender, amount);
41:    }
```

Description

Yaodao : In the protocol, the keccak256 that MerkleProof is used frequently. And the protocol uses the MiMC algorithm to replace the keccak256, the specific implementation function `MiMCpe7()` is in the library `MiMC`

```
    /**
    * MiMC-p/p with exponent of 7
    *
    * Recommended at least 46 rounds, for a polynomial degree of 2^126
    */
    function MiMCpe7( uint256 in_x, uint256 in_k, uint256 in_seed, uint256 round_count ) internal pu
re returns(uint256 out_x) {
        assembly {
            if lt(round_count, 1) { revert(0, 0) }

            // Initialise round constants, k will be hashed
            let c := mload(0x40)
            mstore(0x40, add(c, 32))
            mstore(c, in_seed)

            let localQ := 0x30644e72e131a029b85045b68181585d2833e84879b9709143e1f593f0000001
            let t
            let a

            // Further n-2 subsequent rounds include a round constant
            for { let i := round_count } gt(i, 0) { i := sub(i, 1) } {
                // c = H(c)
                mstore(c, keccak256(c, 32))

                // x = pow(x + c_i, 7, p) + k
                t := addmod(addmod(in_x, mload(c), localQ), in_k, localQ)              // t = x + c_
i + k
                a := mulmod(t, t, localQ)                                               // t^2
                in_x := mulmod(mulmod(a, a, localQ), t, localQ)     // t^5
            }

            // Result adds key again as blinding factor
            out_x := addmod(in_x, in_k, localQ)
        }
    }
```

According to the comments, the implementation should use exponent of 7. However, in the function `MiMCpe7()`, the current implementation `in_x := mulmod(mulmod(a, a, localQ), t, localQ)` is exponent of 5. The exponent of 7 should be `in_x := mulmod(mulmod(a, mulmod(a, a, localQ), localQ), t, localQ)`.

Recommendation

Yaodao : Recommend updating the comments or the implementation in the function `MiMCpe7()`.

Client Response

Fixed

MUF-9:Unlocked Pragma Version

Category	Severity	Status	Contributor
Code Style	Informational	Declined	Yaodao

Code Reference

- code/contracts/interfaces/IDepositWallet.sol#L2
- code/contracts/interfaces/IDepositWalletFactory.sol#L2
- code/contracts/interfaces/IERC20.sol#L2
- code/contracts/interfaces/IMainTreasury.sol#L2
- code/contracts/interfaces/ITreasury.sol#L2
- code/contracts/libraries/MerkleProof.sol#L2
- code/contracts/libraries/MiMC.sol#L2
- code/contracts/libraries/Pairing.sol#L22
- code/contracts/libraries/TransferHelper.sol#L2
- code/contracts/BaseTreasury.sol#L2
- code/contracts/DepositWallet.sol#L2
- code/contracts/DepositWalletFactory.sol#L2
- code/contracts/HotTreasury.sol#L2
- code/contracts/MainTreasury.sol#L2
- code/contracts/Ownable.sol#L2

```
175:        if (!forceWithdrawOpened) forceWithdrawOpened = true;
```

Description

Yaodao : Solidity files in packages have a pragma version `^0.8.10`. The caret `(^)` points to unlocked pragma, meaning the compiler will use the specified version or above.

Recommendation

Yaodao : Recommend the compiler version is instead locked at the lowest version possible that the contract can be compiled at.

Client Response

Declined, same as MUF-7

MUF-10:Gas Optimization: Variables that could be declared as immutable

Category	Severity	Status	Contributor
Gas Optimization	Informational	Declined	Yaodao

Code Reference

- code/contracts/DepositWallet.sol#L10
- code/contracts/DepositWallet.sol#L12

```
    /**
    * MiMC-p/p with exponent of 7
    *
    * Recommended at least 46 rounds, for a polynomial degree of 2^126
    */
    function MiMCpe7( uint256 in_x, uint256 in_k, uint256 in_seed, uint256 round_count ) internal pure returns(uint256 out_x) {
        assembly {
            if lt(round_count, 1) { revert(0, 0) }

            // Initialise round constants, k will be hashed
            let c := mload(0x40)
            mstore(0x40, add(c, 32))
            mstore(c, in_seed)

            let localQ := 0x30644e72e131a029b85045b68181585d2833e84879b9709143e1f593f0000001
            let t
            let a

            // Further n-2 subsequent rounds include a round constant
            for { let i := round_count } gt(i, 0) { i := sub(i, 1) } {
                // c = H(c)
                mstore(c, keccak256(c, 32))

                // x = pow(x + c_i, 7, p) + k
                t := addmod(addmod(in_x, mload(c), localQ), in_k, localQ)              // t = x + c_i + k
                a := mulmod(t, t, localQ)                                              // t^2
                in_x := mulmod(mulmod(a, a, localQ), t, localQ)     // t^5
            }

            // Result adds key again as blinding factor
            out_x := addmod(in_x, in_k, localQ)
        }
    }
```

Description

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

```
    address public override factory;
    address public override treasury;
```

Recommendation

Yaodao : Recommend declaring these variables as immutable. Please note that the immutable keyword only works in Solidity version v0.6.5 and up.

```
    address public immutable override factory;
    address public immutable override treasury;
```

Client Response

Declined, same as MUF-7

MUF-11:Gas Optimization: Lack of check to save gas

Category	Severity	Status	Contributor
Gas Optimization	Informational	Declined	Yaodao

Code Reference

- code/contracts/DepositWallet.sol#L37-L41

```
2:pragma solidity ^0.8.10;

2:pragma solidity ^0.8.10;

2:pragma solidity ^0.8.10;

2:pragma solidity ^0.8.10;

2:pragma solidity ^0.8.10;

2:pragma solidity ^0.8.10;

2:pragma solidity ^0.8.10;

2:pragma solidity ^0.8.10;

2:pragma solidity ^0.8.10;

2:pragma solidity ^0.8.10;

2:pragma solidity ^0.8.10;

2:pragma solidity ^0.8.10;

2:pragma solidity ^0.8.10;

2:pragma solidity ^0.8.10;

22:pragma solidity ^0.8.10;
```

Description

Yaodao : According to the codes in the contract `DepositWallet`, the function `collectETH()` is used to transfer all the ETH deposit in the contract to the treasury. And this function will be batch called by the contract `DepositWalletFactory`. When the balance is 0, the call of transfer is redundant. In the batch calls, these call will cost more gas.

```
    function collectETH(string calldata requestId) external override {
        uint256 balance = address(this).balance;
        TransferHelper.safeTransferETH(treasury, balance);
        emit EtherCollected(treasury, balance, requestId);
    }
```

Recommendation

Yaodao : Recommend adding a check to check whether the balance is over 0 and then transfer can save the gas in the batch calls.

Client Response

Declined, same as MUF-7

MUF-12:Gas Optimization: Use calldata instead of memory

Category	Severity	Status	Contributor
Gas Optimization	Informational	Declined	Yaodao

Code Reference

- code/contracts/Verifier.sol#L119

```
    address public override factory;

    address public override treasury;
```

Description

Yaodao : It's better to use `calldata` instead of memory for function parameters that represent variables that will not be modified.

```
    function submit(
        uint64 zkpId,
        uint256[] memory BeforeAccountTreeRoot,
        uint256[] memory AfterAccountTreeRoot,
        uint256[] memory BeforeCEXAssetsCommitment,
        uint256[] memory AfterCEXAssetsCommitment,
        uint256[2][] memory a, // zk proof参数
        uint256[2][2][] memory b, // zk proof参数
        uint256[2][] memory c, // zk proof参数
        uint256 withdrawMerkelTreeToot,
        uint256 totalBalance,
        uint256 totalWithdraw
    ) public returns (bool r) {
```

Recommendation

Yaodao : Recommend using calldata instead of memory to save gas.

Client Response

Declined, After changing to calldata, a stack too deep error will appear instead, and it will be troublesome to solve, so no modification

MUF-13:code redundancy at `TransferHelper.sol`

Category	Severity	Status	Contributor
Code Style	Informational	Declined	8olidity

Code Reference

- code/contracts/libraries/TransferHelper.sol#L6-L17

```
37:    function collectETH(string calldata requestId) external override {
38:        uint256 balance = address(this).balance;
39:        TransferHelper.safeTransferETH(treasury, balance);
40:        emit EtherCollected(treasury, balance, requestId);
41:    }
```

Description

8olidity : The `safeApprove()` in the `TransferHelper` library is redundant, this function is not used in the project, poc

```
    function safeApprove(
        address token,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, valu
e));
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "TransferHelper::safeApprove: approve failed"
        );
    }
```

Recommendation

8olidity : it is recommended to delete

Client Response

Declined, Same as MUF-7

MUF-14:Gas Optimization: Improve loop code to reduce gas consumption.

Category	Severity	Status	Contributor
Gas Optimization	**Informational**	**Declined**	**LiRiu**

Code Reference

- code/contracts/Verifier.sol#L100
- code/contracts/Verifier.sol#L142
- code/contracts/Verifier.sol#L148
- code/contracts/MainTreasury.sol#L75
- code/contracts/MainTreasury.sol#L94
- code/contracts/DepositWalletFactory.sol#L62
- code/contracts/DepositWalletFactory.sol#L74
- code/contracts/DepositWalletFactory.sol#L85
- code/contracts/DepositWallet.sol#L46
- code/contracts/BaseTreasury.sol#L59
- code/contracts/BaseTreasury.sol#L74
- code/contracts/libraries/MerkleProof.sol#L19
- code/contracts/libraries/MiMC.sol#L58
- code/contracts/libraries/Pairing.sol#L119

```
119:        uint64 zkpId,
```

Description

LiRiu : In this project, loops have been used multiple times. I suggest optimizing your loop code to reduce gas consumption.

Recommendation

LiRiu : Change all `for (uint256 i = 0; i < tokens.length; i++)` into

```
for (uint256 i = 0; i < tokens.length;) {
    ...
    unchecked{
        ++i;
    }
```

Client Response

Declined, There are a lot of changes, no security holes are involved, no modification

MUF-15:Gas Optimization: Cache array length out of the loop to save gas

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	Yaodao

Code Reference

- code/contracts/MainTreasury.sol#L94-L96

```
    function safeApprove(
        address token,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value));
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "TransferHelper::safeApprove: approve failed"
        );
    }
```

Description

Yaodao : In function `updateZKP`, there is a loop which will always read the length of the array in each iteration:

```
for (uint256 j = 0; j < withdrawnInfo.allGeneralWithdrawnIndex.length; j++) {
    delete withdrawnInfo.generalWithdrawnBitMap[withdrawnInfo.allGeneralWithdrawnIndex[j]];
}
```

Since `withdrawnInfo` is a storage struct, there is an extra sload operation which will cost 100 addition gas for each operation.

Recommendation

Yaodao : Recommend caching the length out of the loop.

```
uint256 length = withdrawnInfo.allGeneralWithdrawnIndex.length;
for (uint256 j = 0; j <length; j++) {
    delete withdrawnInfo.generalWithdrawnBitMap[withdrawnInfo.allGeneralWithdrawnIndex[j]];
}
```

Client Response

Fixed

Disclaimer