Competitive Security Assessment

Hinkal

Jan 29th, 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.

The high level conslusion is that

- Hinkal smart contracts have no way of taking users' funds
- Hinkal smart contracts are effective at preserving privacy
- All the issues have been successfully fixed and risks have been acknowledged by the team

Overview

Project Detail

Project Name	Hinkal
Platform & Language	Solidity
Codebase	• https://github.com/Novelty-Today/Hinkal-Neo-Protocol/ • audit commit - c3c78543992a6405d3342ca4fedfc9e8643c06b1 • final commit - f89536c0b6720661e246c65e27241aa42093d69b
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

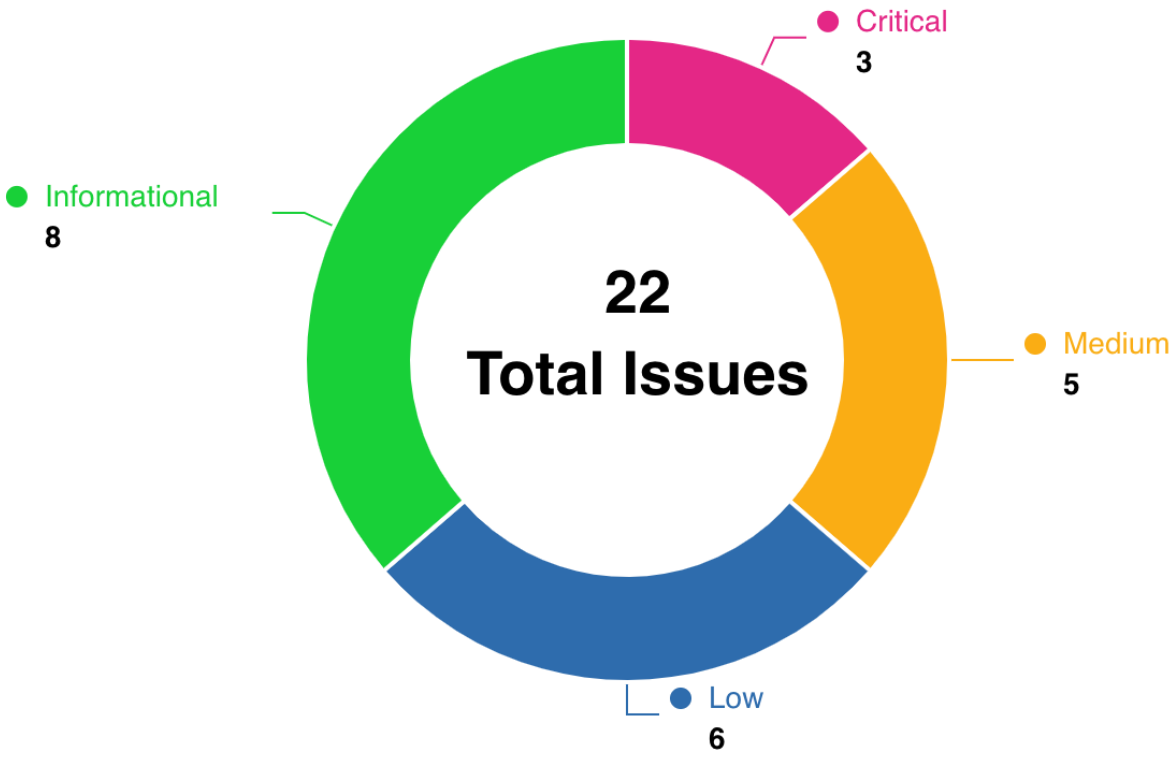
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	3	0	0	3	0	0
Medium	5	0	1	3	0	1
Low	6	0	1	5	0	0
Informational	8	0	1	7	0	0

Audit Scope

File	Commit Hash
solidity/contracts/Hinkal.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1
solidity/contracts/ERC20TokenRegistry.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1
solidity/contracts/RelayStore.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1
solidity/contracts/Transferer.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1
solidity/contracts/AccessToken.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1
solidity/contracts/Merkle.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1
solidity/contracts/interfaces/IHinkal.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1
solidity/contracts/interfaces/IAccessToken.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1
solidity/contracts/interfaces/IPoseidon.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1
solidity/contracts/interfaces/ISwapper2.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1
solidity/contracts/interfaces/IMerkle.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1
solidity/contracts/interfaces/IWrapper.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1
solidity/contracts/interfaces/IVerifier10.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1
solidity/contracts/interfaces/IVerifier2.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1
solidity/contracts/interfaces/ISwapper10.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1
solidity/contracts/interfaces/IRelayStore.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1
solidity/contracts/interfaces/IERC20TokenRegistry.sol	c3c78543992a6405d3342ca4fedfc9e8643c06b1

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
HKL-1	DoS attack can be achived by frontrun	Logical	Critical	Fixed	jayphbee
HKL-2	Duplicate blacklist member check	Gas Optimization	Informational	Fixed	jayphbee
HKL-3	Fail to remove publicKey from merkleTree in `AccessToken` contract	Logical	Critical	Fixed	0xxm
HKL-4	Inconsistent contract implementation with interface	Logical	Low	Fixed	0xxm
HKL-5	Logic Error in `Merkle` contract `insert` & `findAndRemove` function	Logical	Low	Acknowledged	w2ning

HKL-6	No event emitted when critical system parameter changed	Code Style	Informational	Fixed	0xxm, jayphbee
HKL-7	No need to +1/-1 for `restorePoints` value in `Merkle` contract `findAndRemove` and `restoreDeleted` function	Gas Optimization	Informational	Fixed	alansh
HKL-8	No one can call `Merkle.findAndRemove` and `Merkle.restoreDeleted` after `Merkle` transfered ownership to `Hinkal`	Privilege Related	Medium	Acknowledged	jayphbee
HKL-9	Pass an additional parameter `index` in `Merkle` contract `findAndRemove` function to avoid the loop	DOS	Medium	Fixed	alansh, jayphbee
HKL-10	Potential Reentrancy risk & Signature Replay in `Hinkal` contract `transact` function	Reentrancy	Low	Fixed	w2ning
HKL-11	Return value should be used in `Hinkal` contract `swap` function	Logical	Medium	Declined	w2ning
HKL-12	Should check against leaf overflow in `Merkle` contract `insert` function	Logical	Low	Fixed	alansh
HKL-13	Unimplemented or Unused Function `register` in Contract Hinkal	Logical	Informational	Acknowledged	0xxm
HKL-14	Unsafe Signature in Contract AccessToken	Signature Forgery or Replay	Medium	Fixed	0xxm
HKL-15	Unused constant `p`	Gas Optimization	Informational	Fixed	alansh, yekong, jayphbee
HKL-16	Use `call` instead of `transfer` to transfer ETH	Language Specific	Informational	Fixed	0xxm
HKL-17	Use capital letters for constant names in `Hinkal`,`Merkle` contract	Code Style	Informational	Fixed	yekong
HKL-18	Variables can be immutable	Gas Optimization	Informational	Fixed	0xxm

HKL-19	Wrong `require` in `ERC20TokenRegistry` contract	Logical	Critical	Fixed	yekong, jayphbee
HKL-20	`AccessToken.addToken` Redundant authority validation	Logical	Low	Fixed	0xxm, jayphbee
HKL-21	`_relayPercentage` and `_relayPercentageSwap` should be capped	Privilege Related	Low	Fixed	0xxm, jayphbee
HKL-22	missing transferERC20Token to relay in `Hinkal` contract `transact` function	Logical	Medium	Fixed	alansh

HKL-1:DoS attack can be achived by frontrun

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/solidity/contracts/Hinkal.sol#L275-L278 • code/solidity/contracts/Hinkal.sol#L390-L393	Fixed	jayphbee

Code

```
275:         require(
276:             circomData.rootHashHinkal == merkleTree.getRootHash(),
277:             "Invalid merkle tree root"
278:         );

390:         require(
391:             swapData.rootHashHinkal == merkleTree.getRootHash(),
392:             "Invalid merkle tree root"
393:         );
```

Description

jayphbee : When user depositing, withdrawing or swapping, the input `rootHashHinkal` is checked against `merkleTree.getRootHash()`.

```
        // Root Hash Validation
        require(
            circomData.rootHashHinkal == merkleTree.getRootHash(),
            "Invalid merkle tree root"
        );
```

The merkle tree root hash is updated accordingly when `Merkle.insert` is called in `Hinkal.transact` and `Hinkal.swap`.

```
        // Updating Merkle Tree with new commitments
        uint256 index0 = merkleTree.insert(circomData.outCommitments[0]);
        uint256 index1 = merkleTree.insert(circomData.outCommitments[1]);
```

Say there are **pending** transactions simultaneously select the same `rootHashHinkal` as the input parameter for `Hinkal.transact2`, once one of the transaction is successed, the remaining transactions are doomed to fail due to the `rootHashHinkal` check logic mentioned above.

The design is inherently deriving this kind of DoS attack because the merkle tree used to store the leaf is shared by all users and the merkle tree root hash is checked against the lastest `merkleTree.getRootHash()` for each transaction. The impact is that the cost to invalidate other tranactions in the mempool is nearly zero(only transaction fee) and every user of the protocol can think of as "attacker". The availability of the protocol is hugely downgraded or even worse when a malicious user observe that there are transactions in the mempool, he can invalidate all of them by submmiting higher gas price transaction to update the merkle root hash. The whole protocol is hajacked.

Recommendation

jayphbee : I think this need a new design to overcome this weakness.

Client Response

We introduced roots mapping where we store 25 recent roots, so if there is up to 25 transaction in the same block, then malicious user will not be able invalidate them. In Merkle Contract => insert function, we are adding new roots to this mapping.

HKL-2:Duplicate blacklist member check

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/solidity/contracts/AccessToken.sol#L34-L35 • code/solidity/contracts/AccessToken.sol#L85-L87	Fixed	jayphbee

Code

```
34:     require(hasToken(publicKey) == false, "User already has an access token");
35:     require(!blacklist[publicKey], "PublicKey has been blacklisted");

85:   function hasToken(uint256 publicKey) public view returns (bool) {
86:     return !blacklist[publicKey] && accessTokens[publicKey];
87:   }
```

Description

jayphbee : In the `AccessToken.sol::addToken` function, `publicKey` can be added if it haven't been and not in the `blacklist`.

```
    require(hasToken(publicKey) == false, "User already has an access token");
    require(!blacklist[publicKey], "PublicKey has been blacklisted");
```

Here `hasToken` function makes sure `publicKey` is not in the `blacklist`.

```
  function hasToken(uint256 publicKey) public view returns (bool) {
    return !blacklist[publicKey] && accessTokens[publicKey];
  }
```

So the `require(!blacklist[publicKey], "PublicKey has been blacklisted");` statement can be omitted to save gas.

Recommendation

Client Response

`require(!blacklist[publicKey], "PublicKey has been blacklisted")` was omitted

HKL-3:Fail to remove publicKey from merkleTree in `AccessToken` contract

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	code/solidity/contracts/AccessToken.sol#L63	Fixed	0xxm

Code

```
63:     merkleTree.findAndRemove(uint160(publicKey));
```

Description

0xxm : AccessToken is stored in merkleTree as uint256 type publicKey, however it is casted to uint160 when try to remove it in function `blacklistPublicKey`. This will invalid the merkle tree state, which in turn broken the Hinkal contract as `accessToken.getRootHash()` is not as expected.

Note: Actually, I don't see a necessity to use merkleTree in `AccessToken` contract. The proof of passing KYC is already recorded as access token.

```
  function addToken(uint256 publicKey, SignatureData memory signatureData)
    public
    payable
    onlyOwner
  {
    ...
    uint256 index = merkleTree.insert(publicKey);
    accessTokens[publicKey] = true;
    emit NewPubkeyAdded(index, publicKey);
  }

  function blacklistPublicKey(uint256 publicKey) public onlyOwner {
    blacklist[publicKey] = true;
    merkleTree.findAndRemove(uint160(publicKey));
  }
```

Recommendation

0xxm : I suggest to use one of the following solutions:

1. Remove the integer cast in function `blacklistPublicKey` to: `merkleTree.findAndRemove(publicKey)`

2. Consider not remove publicKey from merkleTree when blacklist PublicKey. Then the user does not need to re-mint accessToekn after been removed from blacklist.
3. Remove merkleTree from `AccessToken` contract if applicable.

Client Response

Removed the integer cast in function `blacklistPublicKey()`

HKL-4:Inconsistent contract implementation with interface

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/solidity/contracts/ERC20TokenRegistry.sol#L6 • code/solidity/contracts/Merkle.sol#L7 • code/solidity/contracts/RelayStore.sol#L7 • code/solidity/contracts/AccessToken.sol#L14	Fixed	0xxm

Code

```
6:contract ERC20TokenRegistry is Ownable {

7:contract Merkle is Ownable {

7:contract RelayStore is Ownable {

14:contract AccessToken is Ownable {
```

Description

0xxm : Contract `AccessToken`, `ERC20TokenRegistry`, `Merkle` and `RelayStore` all have interface but not inherit corresponding interface in their implentations, which cause inconsistent function between interface and implementation.
For example, interface `IRelayStore` declared function `addOrSetRelay` as `function addOrSetRelay(string memory url, uint256 priority) external`, while in contract `RelayStore` it is defined as `function addOrSetRelay( address relayAddress, string memory url, uint256 priority) public`.
This inconsistency may cause transaction failure when using interface to call mismatched function.

Recommendation

0xxm : It recommented to inheritate interface in contract implementation and making sure all necessary functions are consistent.

Client Response

We rewrote interfaces, Inheritance added where applicable.

HKL-5:Logic Error in `Merkle` contract `insert` & `findAndRemove` function

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/solidity/contracts/Merkle.sol #L33 • code/solidity/contracts/Merkle.sol #L68	Acknowledged	w2ning

Code

```
33:  function insert(uint256 leaf) public onlyOwner returns (uint256) {

68:  function findAndRemove(uint256 dataToRemove) public onlyOwner {
```

Description

w2ning : Contract `Merkle` inherits from `Ownable`, and there is only one Owner at the same time.
Function `findAndRemove () public onlyOwner` is called by contract `AccessToken`.
Function `insert () public onlyOwner` is called by contract `Hinkal` and `AccessToken`.
Two contracts cannot be the owner of contract `Merkle` at the same time.

Recommendation

w2ning : Add independent modifier
Consider below fix in the `Merkle.findAndRemove()` function

```
    modifier onlyAccessToken() {

        _checkIsAccessToken();
        _;
    }

    function findAndRemove(uint256 dataToRemove) public onlyAccessToken {

        ...

    }
```

Consider below fix in the `Merkle.insert()` function

```
    modifier onlyHinkalOrAccessToken() {

        _checkIsHinkalOrAccessToken();
        _;
    }

    function insert(uint256 leaf) public onlyHinkalOrAccessToken returns (uint256) {

        ...

    }
```

Client Response

In contracts, Hinkal and AccessToken we create two different instances of Merkle Tree Contract, so there is no simultaneous ownership of Merkle Tree by aforementioned contracts.

HKL-6:No event emitted when critical system parameter changed

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/solidity/contracts/RelayStore.sol#L7 • code/solidity/contracts/AccessToken.sol#L14	Fixed	0xxm, jayphbee

Code

```
7:contract RelayStore is Ownable {

14:contract AccessToken is Ownable {
```

Description

0xxm : There should always be events emitted in sensitive functions, especially for functions controlled by centralization roles.

jayphbee : When there are critical system parameters changed, corresponding event should be emitted to notify the user for them to react to this action. This is the case for `setRelayPercentage` and `setRelayPercentageSwap` function:

```
    function setRelayPercentage(uint8 _relayPercentage) public onlyOwner {
        relayPercentage = _relayPercentage;
    }

    function setRelayPercentageSwap(uint8 _relayPercentageSwap)
        public
        onlyOwner
    {
        relayPercentageSwap = _relayPercentageSwap;
    }
```

Recommendation

0xxm : It is recommended emitting events for the following functions:

- RelayStore::setRelayPercentage()
- RelayStore::setRelayPercentageSwap()
- RelayStore::addOrSetRelay()

- AccessToken::blacklistPublicKey()
- AccessToken::blacklistAddress()
- AccessToken::removePublicKeyFromBlacklist()
- AccessToken::removeAddressFromBlacklist()

  jayphbee : emit corresponding event when `setRelayPercentage` and `setRelayPercentageSwap`

Client Response

We added all events mentioned in the issue.

HKL-7:No need to +1/-1 for `restorePoints` value in `Merkle` contract `findAndRemove` and `restoreDeleted` function

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/solidity/contracts/Merkle.sol #L83 • code/solidity/contracts/Merkle.sol #L90	Fixed	alansh

Code

```
83:         restorePoints[dataToRemove] = i + 1;

90:     uint256 restoreIndex = restorePoints[dataToRestore] - 1;
```

Description

alansh : Because `i` is already guaranteed to be `>0`, there's no need to `restorePoints[dataToRemove] = i + 1;` in `findAndRemove` and `uint256 restoreIndex = restorePoints[dataToRestore] - 1;` which causes more gas

Recommendation

alansh : In the `Merkle.findAndRemove` function

```
  restorePoints[dataToRemove] = i;
```

In the `Merkle.restoreDeleted` function

```
  uint256 restoreIndex = restorePoints[dataToRestore];
```

Client Response

Restore Functionality was deleted. We find it redundant.

HKL-8:No one can call `Merkle.findAndRemove` and `Merkle.restoreDeleted` after `Merkle` transfered ownership to `Hinkal`

Category	Severity	Code Reference	Status	Contributor
Privilege Related	Medium	• code/solidity/contracts/Merkle.sol #L68 • code/solidity/contracts/Merkle.sol #L88	Acknowledged	jayphbee

Code

```
68:  function findAndRemove(uint256 dataToRemove) public onlyOwner {

88:  function restoreDeleted(uint256 dataToRestore) public onlyOwner {
```

Description

jayphbee : `Merkle` is an external dependency of `Hinkal` contract. `Merkle` has to transfer ownership to `Hinkal` so that `Merkle.insert` can be called in `Hinkal` contract, because `Merkle.insert` is decorated by the `onlyOwner` modifier.

```
function insert(uint256 leaf) public onlyOwner returns (uint256)
```

This is the same with `findAndRemove` and `restoreDeleted`

```
function findAndRemove(uint256 dataToRemove) public onlyOwner
function restoreDeleted(uint256 dataToRestore) public onlyOwner
```

`findAndRemove` is used to remove some unexpectedly inserted leaf and the `restoreDeleted` is used to revert the changes caused by `findAndRemove`. These two functions can only be called by the `Hinkal` contract or `AccessToken` contract after `Merkle` transfers ownership to `Hinkal` contract or `AccessToken` contract.

However, There is no call to `findAndRemove` or `restoreDeleted` in contracts `AccessToken` or `Hinkal`. Hence `findAndRemove` and `restoreDeleted` become dead code.

The impact is that the unexpectedly inserted leaf can't be removed by calling `findAndRemove` due to the privilege issue.

Recommendation

jayphbee : I would suggest create a new role that can call `findAndRemove` and `restoreDeleted` instead of only the `Hinkal` contract.

```
function findAndRemove(uint256 dataToRemove) public onlyAdmin
function restoreDeleted(uint256 dataToRestore) public onlyAdmin
```

Client Response

In AccessToken contract there is a call of `findAndRemove()` function, hence, we will keep this function in Merkle Tree contract. In Hinkal contract, we do not need to call `findAndRemove()`. `restoreDeleted()` function was deleted.

HKL-9:Pass an additional parameter `index` in `Merkle` contract `findAndRemove` function to avoid the loop

Category	Severity	Code Reference	Status	Contributor
DOS	Medium	• code/solidity/contracts/Merkle.sol #L67-L85	Fixed	alansh, jayphbee

Code

```
67:
68:  function findAndRemove(uint256 dataToRemove) public onlyOwner {
69:    for (uint256 i = 2**(m_levels - 1); i < m_index; i++) {
70:      if (tree[i] == dataToRemove) {
71:        tree[i] = 0;
72:        uint256 count = m_index - 2**(m_levels - 1); // number of inserted leaves
73:        uint256 twoPower = logarithm2(count);
74:
75:        uint256 currentNodeIndex = i;
76:        for (uint256 j = 1; j <= twoPower; j++) {
77:          currentNodeIndex /= 2;
78:          tree[currentNodeIndex] = hash(
79:            tree[currentNodeIndex * 2],
80:            tree[currentNodeIndex * 2 + 1]
81:          );
82:        }
83:        restorePoints[dataToRemove] = i + 1;
84:      }
85:    }
```

Description

alansh : The current implementation iterates over all leafs to find a value, which is way too heavy(when `m_index - 2**(m_levels - 1)` is very large, the gas cost will be very big too), should pass an additional parameter `index` to locate the leaf directly, and compare whether the leaf is equal to `dataToRemove`. So the caller `AccessToken.blacklistPublicKey` should also have this parameter.
jayphbee : In the `Merkle` contract `m_index` is monotonically increasing. As with the popularity of the protocol, `m_index` could be very large so that the for loop in the `findAndRemove` function can consume gas greater than the block gas limit.

```
function findAndRemove(uint256 dataToRemove) public onlyOwner {
  for (uint256 i = 2**(m_levels - 1); i < m_index; i++) {
    ...
  }
}
```

The impact is that `findAndRemove` can't be called anymore if the block gas limit doesn't increase.

Recommendation

alansh :

```
function findAndRemove(uint256 dataToRemove, uint256 index) public onlyOwner {
    require(index >= 2**(m_levels - 1) && index < m_index, "index out of range");
    require(tree[index] == dataToRemove, "leaf doesn't match dataToRemove");
    ... // same as original
    restorePoints[dataToRemove] = index; // +1 is not needed since index is guaranteed >0 here
}
```

jayphbee : There's no easy way to fix issue due to the merkle tree root hash calculation is dependent on `m_index`, we can't simplely decrease the value of `m_index` when remove leaf from the merkle tree. This could rely on the team to propose a new design.

Client Response

Followed recommendation of `alansh` - introduced index parameter in AccessToken and Merkle Tree contracts, so we do not need to go over for loop

HKL-10:Potential Reentrancy risk & Signature Replay in `Hinkal` contract `transact` function

Category	Severity	Code Reference	Status	Contributor
Reentrancy	Low	• code/solidity/contracts/Hinkal.sol#L227 • code/solidity/contracts/Hinkal.sol#L245 • code/solidity/contracts/Hinkal.sol#L457 • code/solidity/contracts/Hinkal.sol#L476	Fixed	w2ning

Code

```
227:    function transact2(

245:    function transact10(

457:    function swap2(

476:    function swap10(
```

Description

w2ning : `transact12`, `transact10`, `Swap2` & `Swap10`. These four functions have potential arbitrary external calls, which may lead to the risk of reentry and may lead to signature retransmission attacks.

Recommendation

w2ning : 1. consider using function modifiers such as `nonReentrant` from Reentrancy Guard to prevent re-entrancy at the contract level.

2. Updating Merkle Tree before calling internal function in `transact12`, `transact10`, `Swap2` & `Swap10`

Consider below fix in the `Hinkal.transact2()` and `Hinkal.transact10()` function

```
        CircomData memory circomData = processCircomInputs(input);


        // Updating Merkle Tree first
        uint256 index0 = merkleTree.insert(circomData.outCommitments[0]);
        uint256 index1 = merkleTree.insert(circomData.outCommitments[1]);

        // Then call internal function
        transact(encryptedOutputs, circomData, msg.value, msg.sender);
```

Consider below fix in the `Hinkal.Swap2()` and `Hinkal.Swap10()` function

```
        SwapData memory swapData = processSwapperInputs(input);

        // Updating Merkle Tree first
        uint256 index0 = merkleTree.insert(swapData.outCommitments[0]);
        uint256 index1 = merkleTree.insert(swapData.outCommitments[1]);

        // Then call internal function
        swap(encryptedOutputs, swapData);
```

Client Response

ReentrancyGuard contract from openzeppelin imported, added nonReentrant to applicable functions.

HKL-11:Return value should be used in `Hinkal` contract `swap` function

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/solidity/contracts/Hinkal.sol#L437	Declined	w2ning

Code

```
437:            wrapper.withdraw(swapData.outSwapAmount);
```

Description

w2ning : Return value of swapRouter.exactInputSingle() should be used in wrapper.withdraw() function

```
    uint256 swapOutput = swapRouter.exactInputSingle(params);

    if (swapData.relay != address(0)) {
        require(msg.sender == swapData.relay, "Unauthorized relay 1");
        require(
            relayStore.isRelayInList(swapData.relay),
            "Unauthorized relay 2"
        );
        if (swapOutput >= swapData.outSwapAmount) {
            transferERC20Token(
                swapData.outErc20TokenAddress,
                swapData.relay,
                swapOutput - swapData.outSwapAmount
            );
        }
    }

    if (swapData.outErc20TokenAddress == address(wrapper)) {

        //  The return value should be used instead of the value in the passed parameter
        wrapper.withdraw(swapData.outSwapAmount);
    }
```

Recommendation

w2ning : Consider below fix in the `Hinkal.swap()` function

```
if (swapData.outErc20TokenAddress == address(wrapper)) {

    //  The return value should be used instead of the value in the passed parameter
    wrapper.withdraw(swapOutput);
}
```

Client Response

This is the desired behaviour, since swap is always done through the relayer who submit transaction on behalf of the user and takes fee of it. Amount of wrapped token equal to (`relayer fee = swapOutput - swapData.outSwapAmount`) will be transferred to the relayer, meaning that the contract would not have enough balance to call wrapper.withdraw(`swapOutput`).

HKL-12:Should check against leaf overflow in `Merkle` contract `insert` function

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/solidity/contracts/Merkle.sol #L33-L48	Fixed	alansh

Code

```
33:  function insert(uint256 leaf) public onlyOwner returns (uint256) {
34:    tree[m_index] = leaf;
35:    m_index++;
36:
37:    uint256 count = m_index - 2**(m_levels - 1); // number of inserted leaves
38:    uint256 twoPower = logarithm2(count);
39:
40:    uint256 currentNodeIndex = m_index - 1;
41:    for (uint256 i = 1; i <= twoPower; i++) {
42:      currentNodeIndex /= 2;
43:      tree[currentNodeIndex] = hash(
44:        tree[currentNodeIndex * 2],
45:        tree[currentNodeIndex * 2 + 1]
46:      );
47:    }
48:    return m_index - 1;
```

Description

alansh : The merkle tree has a capacity of `2**(m_levels-1)`, should check against overflow when `insert`, otherwise the leaf node may be overwritten as parent node.

Recommendation

alansh : Consider below fix in the `Merkle.insert()` function

```
    function insert(uint256 leaf) public onlyOwner returns (uint256) {
        require(m_index < 2**m_levels, "merkle tree overflow");

        tree[m_index] = leaf;
        m_index++;
        ...
    }
```

Client Response

Require Statement is introduced

HKL-13:Unimplemented or Unused Function `register` in Contract Hinkal

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/solidity/contracts/Hinkal.sol#L497-L503	Acknowledged	0xxm

Code

```
497:    function register(bytes calldata shieldedAddress) public {
498:        require(
499:            !accessToken.isBlacklistedAddress(msg.sender),
500:            "Blacklisted addresses aren't allowed to register"
501:        );
502:        emit Register(msg.sender, shieldedAddress);
503:    }
```

Description

0xxm : Function `register` in Contract Hinkal is not correctly implemented as it does nothing but emit an event.

```
function register(bytes calldata shieldedAddress) public {
    require(
        !accessToken.isBlacklistedAddress(msg.sender),
        "Blacklisted addresses aren't allowed to register"
    );
    emit Register(msg.sender, shieldedAddress);
}
```

Recommendation

0xxm : Delete this function if it is deprecated, otherwise update it with correct implemention.

Client Response

This function is used in front-end to link shielded address and etherium address. The event emitted allows sender to scan events and see if there exists receiver shielded address which corresponds to etherium public address. The visibility is changed to external instead of public (see linked commit)

HKL-14:Unsafe Signature in Contract AccessToken

Category	Severity	Code Reference	Status	Contributor
Signature Forgery or Replay	Medium	• code/solidity/contracts/AccessToken.sol#L37-L50	Fixed	0xxm

Code

```
37:     string memory pubkeyToString = publicKey.toHexString(32);
38:     string memory prefixPubKey = "pubkey";
39:     bytes memory prefix = "\x19Ethereum Signed Message:\n72";
40:
41:     bytes32 prefixedMessage = keccak256(
42:       abi.encodePacked(prefix, prefixPubKey, pubkeyToString)
43:     );
44:
45:     address signer = ecrecover(
46:       prefixedMessage,
47:       signatureData.v,
48:       signatureData.r,
49:       signatureData.s
50:     );
```

Description

0xxm : The signature used in function `addToken()` of contract AccessToken does not contain `chainId` and `nonce`, which is subject to replay attacks. Especially, the signature is generated from simple elements and Hinkal protocal is likely to be deployed on multiple chains, which exposes more attack surface even further.

```
string memory pubkeyToString = publicKey.toHexString(32);
string memory prefixPubKey = "pubkey";
bytes memory prefix = "\x19Ethereum Signed Message:\n72";

bytes32 prefixedMessage = keccak256(
  abi.encodePacked(prefix, prefixPubKey, pubkeyToString)
);
```

Recommendation

0xxm : We recomment to add `nonce`, 'owner address' and `chainId` to prevent reply attack.

Client Response

We added `chainId` and `nonce` and owner address

HKL-15:Unused constant `p`

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/solidity/contracts/Merkle.sol #L16-L17	Fixed	alansh, yekong, jayphbee

Code

```
16:  uint256 constant p =
17:    21888242871839275222246405745257275088548364400416034343698204186575808495617; //
https://docs.circom.io/circom-language/basic-operators/
```

Description

alansh : `uint256 constant p` is unused , should delete it to save some gas
yekong : The presence of state or local variables that are declared but never used in the codebase. They may increase computation costs and lead to unnecessary gas consumption.
jayphbee : The constant `p` is never used in the `Merkle` contract.

```
  uint256 constant p =
    21888242871839275222246405745257275088548364400416034343698204186575808495617; //
https://docs.circom.io/circom-language/basic-operators/
```

Recommendation

alansh : Delete these two lines

```
    uint256 constant p =
        21888242871839275222246405745257275088548364400416034343698204186575808495617; //
https://docs.circom.io/circom-language/basic-operators/
```

yekong : Remove the unused variables to avoid negative effects and improve code readability if there is no plan for further usage.
jayphbee : remove the unused constant `p`.

Client Response

`p` was deleted from Merkle Tree

HKL-16:Use `call` instead of `transfer` to transfer ETH

Category	Severity	Code Reference	Status	Contributor
Language Specific	Informational	• code/solidity/contracts/AccessToken.sol#L98 • code/solidity/contracts/Hinkal.sol#L340 • code/solidity/contracts/Hinkal.sol#L343 • code/solidity/contracts/Hinkal.sol#L345	Fixed	0xxm

Code

```
98:     payable(msg.sender).transfer(address(this).balance);

340:                     payable(circomData.recipientAddress).transfer(

343:                     payable(circomData.relay).transfer(relayFee);

345:                     payable(circomData.recipientAddress).transfer(
```

Description

0xxm : In EIP-1884(https://eips.ethereum.org/EIPS/eip-1884), the gas of some opcodes is increased, for example, SLOAD is increased from 200 gas to 800 gas, and BALANCE is increased from 400 gas to 700 gas. Therefore, it is recommended to use `call` instead of `transfer` which has a limit of 2300 gas to transfer money, otherwise the transaction may fail.

```
function transact(
    bytes[2] memory encryptedOutputs,
    CircomData memory circomData,
    uint256 value,
    address sender
) private {
  ...
  if (circomData.relay != address(0)) {
      uint256 relayFee = (relayStore.relayPercentage() *
          uint256(-circomData.publicAmount)) / 1000;
      require(
          circomData.relayFee == p - relayFee,
          "relay Fee Mismatch"
      );
      payable(circomData.recipientAddress).transfer(
          uint256(-circomData.publicAmount) - relayFee
      );
      payable(circomData.relay).transfer(relayFee);
  } else {
      payable(circomData.recipientAddress).transfer(
          uint256(-circomData.publicAmount)
      );
  }
  ...
}

function withdraw() public onlyOwner {
  payable(msg.sender).transfer(address(this).balance);
}
```

Recommendation

0xxm : Use `call` to send eth in the abovementioned code.

Client Response

We replaced transfer with call in AccessToken.sol, introduced transferETH function in Transferer.sol, replaced all occurances of transfer with transferETH.

HKL-17:Use capital letters for constant names in `Hinkal`,`Merkle` contract

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/solidity/contracts/Merkle.sol#L10 • code/solidity/contracts/Merkle.sol#L16 • code/solidity/contracts/Hinkal.sol#L34	Fixed	yekong

Code

```
10:  uint256 public constant m_levels = 20; // 20 levels deep tree

16:  uint256 constant p =

34:    uint256 constant p =
```

Description

yekong : Solidity defines a naming convention that should be followed.

Recommendation

yekong : Constants should be named with all capital letters with underscores separating words. Examples: MAX_BLOCKS, TOKEN_NAME, TOKEN_TICKER, CONTRACT_VERSION.

Client Response

We renamed `p`, `m_levels` and `maxRootNumber` constants appropriatly.

HKL-18:Variables can be immutable

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/solidity/contracts/AccessToken.sol#L17 • code/solidity/contracts/Hinkal.sol#L22-L31	Fixed	0xxm

Code

```
17:  IMerkle merkleTree; // additional merkle tree to store deposit addresses

22:    IMerkle public merkleTree;
23:    IVerifier2 public verifier2;
24:    IVerifier10 public verifier10;
25:    ISwapper2 public swapper2;
26:    ISwapper10 public swapper10;
27:    IAccessToken public accessToken;
28:    IERC20TokenRegistry public ERC20TokenRegistry;
29:    IRelayStore public relayStore;
30:    ISwapRouter public swapRouter;
31:    IWrapper public wrapper;
```

Description

0xxm : External contract addresses in contract Hinkal and AccessToken is declared as public storage variables, but they are only intialized in constructor and never modifed. It is recommended to change them as immutable variables to save gas.

```
contract Hinkal is Ownable, Transferer {
    IMerkle public merkleTree;
    IVerifier2 public verifier2;
    IVerifier10 public verifier10;
    ISwapper2 public swapper2;
    ISwapper10 public swapper10;
    IAccessToken public accessToken;
    IERC20TokenRegistry public ERC20TokenRegistry;
    IRelayStore public relayStore;
    ISwapRouter public swapRouter;
    IWrapper public wrapper;
   ...
}

contract AccessToken is Ownable {
  using Strings for uint256;
  IMerkle merkleTree; // additional merkle tree to store deposit addresses
 ...
}
```

Recommendation

0xxm : Change the above-mentioned variables as immutable to save gas.

Client Response

We made all variables that are only used in the constructor immutable.

HKL-19:Wrong `require` in `ERC20TokenRegistry` contract

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/solidity/contracts/ERC20TokenRegistry.sol#L16 • code/solidity/contracts/ERC20TokenRegistry.sol#L21	Fixed	yekong, jayphbee

Code

```
16:         require(tokenRegistry[erc20Token] = false, "Token is already added");

21:         require(tokenRegistry[erc20Token] = true, "Token is not in the list");
```

Description

yekong : Note the difference between = and ==
jayphbee : `ERC20TokenRegistry.addToken` always revert due to require condition always evaluate to false.

```
  require(tokenRegistry[erc20Token] = false, "Token is already added");
```

The impact is that `ERC20TokenRegistry` can't not add any token after initlalization.
jayphbee : Tokens not in the `tokenRegistry` can be removed due the require condition awlays evaluate to true in the `ERC20TokenRegistry.removeToken` funciton.

```
  require(tokenRegistry[erc20Token] = true, "Token is not in the list");
```

Recommendation

yekong : Make the following modifications: require(tokenRegistry[erc20Token] == false, "Token is already added");
require(tokenRegistry[erc20Token] == true, "Token is not in the list");
jayphbee : change `tokenRegistry[erc20Token] = false` to `tokenRegistry[erc20Token] == false`
jayphbee : change `tokenRegistry[erc20Token] = true` to `tokenRegistry[erc20Token] == true`

Client Response

Corrected - was a typo

HKL-20: `AccessToken.addToken` Redundant authority validation

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/solidity/contracts/AccessToken.sol#L28-L31	Fixed	0xxm, jayphbee

Code

```
28:  function addToken(uint256 publicKey, SignatureData memory signatureData)
29:    public
30:    payable
31:    onlyOwner
```

Description

0xxm : According to the project white paper, the user is required to mint access token after verification of PII is done. However, the `addToken` function in AccessToken contract is restricted to owner with `onlyOwner` modifer, which is inconsistent with white paper.

The `addToken` function already uses owner's signature as KYC proof, and requires a 0.001 ETH as mint fee. It should be open to user to mint access token by himself.

```
function addToken(uint256 publicKey, SignatureData memory signatureData)
    public
    payable
    onlyOwner
  {
    require(msg.value == 10**15, "minting fee is 0.001");
    require(hasToken(publicKey) == false, "User already has an access token");
    require(!blacklist[publicKey], "PublicKey has been blacklisted");

    string memory pubkeyToString = publicKey.toHexString(32);
    string memory prefixPubKey = "pubkey";
    bytes memory prefix = "\x19Ethereum Signed Message:\n72";

    bytes32 prefixedMessage = keccak256(
      abi.encodePacked(prefix, prefixPubKey, pubkeyToString)
    );

    address signer = ecrecover(
      prefixedMessage,
      signatureData.v,
      signatureData.r,
      signatureData.s
    );
    require(
      signer == owner(),
      "Signature must be signed by the owner of the contract"
    );
```

jayphbee : `AccessToken.sol::addToken` has redundant authority validation.

```
function addToken(uint256 publicKey, SignatureData memory signatureData)
    public
    payable
    onlyOwner
  {
    ...
    address signer = ecrecover(
      prefixedMessage,
      signatureData.v,
      signatureData.r,
      signatureData.s
    );
    require(
      signer == owner(),
      "Signature must be signed by the owner of the contract"
    );
   ...
  }
```

In order for `addToken` to be called successfully, it has to pass the `onlyOwner` modifier **and** the signature based authority validation. Here `signer == owner()` is equivalent to `onlyOwner` modifier, so this is a duplicate authority check.

Recommendation

0xxm : Remove `onlyOwner` modifer in the `addToken` function of AccessToken contract.
jayphbee : Either use the `onlyOwner` modifier or the signature based authority validation.

Client Response

Removed onlyOwner modifier

HKL-21: `_relayPercentage` and `_relayPercentageSwap` should be capped

Category	Severity	Code Reference	Status	Contributor
Privilege Related	Low	• code/solidity/contracts/RelayStore.sol#L12-L21	Fixed	0xxm, jayphbee

Code

```
12:     function setRelayPercentage(uint8 _relayPercentage) public onlyOwner {
13:         relayPercentage = _relayPercentage;
14:     }
15:
16:     function setRelayPercentageSwap(uint8 _relayPercentageSwap)
17:         public
18:         onlyOwner
19:     {
20:         relayPercentageSwap = _relayPercentageSwap;
21:     }
```

Description

0xxm : In the RelayStore contract, the owner can set the fee for relay. Either by intention or mistake, the contract owner may set a very high fee, causing a loss to users.

```
function setRelayPercentage(uint8 _relayPercentage) public onlyOwner {
    relayPercentage = _relayPercentage;
}

function setRelayPercentageSwap(uint8 _relayPercentageSwap)
    public
    onlyOwner
{
    relayPercentageSwap = _relayPercentageSwap;
}
```

jayphbee : There should be an upper bound for `_relayPercentage` and `_relayPercentageSwap` in `RelayStore.setRelayPercentage` and `RelayStore.setRelayPercentageSwap` function. If there's no such

upper bound the owner of `RelayStore` could unexpectedly set them to an unreasonable large value, which could lead to user lose money.

Recommendation

0xxm : Add upper limit check to `relayPercentage` and `relayPercentageSwap` to prevent unexpected fee value. I also recommend to carefully manage the owner's private keys to avoid single point breach.
jayphbee : Add upper bound for `_relayPercentage` and `_relayPercentageSwap`.

Client Response

We set maximum values of `_relayPercentage` and `_relayPercentageSwap` equal to 1%

HKL-22:missing transferERC20Token to relay in `Hinkal` contract `transact` function

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/solidity/contracts/Hinkal.sol# L320-L324	Fixed	alansh

Code

```
320:                 transferERC20Token(
321:                     circomData.erc20TokenAddress,
322:                     circomData.recipientAddress,
323:                     uint256(-circomData.publicAmount) - relayFee
324:                 );
```

Description

alansh : when both `circomData.erc20TokenAddress` and `circomData.relay` are set, should transfer `relayFee` to `circomData.relay`

Recommendation

alansh : Add the `transferERC20Token` call for `circomData.relay` in `Hinkal.transact` function

```
                transferERC20Token(
                    circomData.erc20TokenAddress,
                    circomData.recipientAddress,
                    uint256(-circomData.publicAmount) - relayFee
                );
                transferERC20Token(
                    circomData.erc20TokenAddress,
                    circomData.relay,
                    relayFee
                );
```

Client Response

We added the recommended transferERC20Token function call.

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.