HASHBOX

Smart Contract Review

Deliverable: Smart Contract Audit Report

Security Report Aug 2022

Disclaimer

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the Company. The content, conclusions and recommendations set out in this publication are elaborated in the specific for only project.

HireCA does not guarantee the accuracy of the data included in this study. All representations, warranties, undertakings and guarantees relating to the report are excluded, particularly concerning – but not limited to – the qualities of the assessed projects and products. Neither the Company nor any personating on the Company's behalf may be held responsible for the use that may be made of the information contained herein. HireCA retains the right to display audit reports and other content elements as examples of their work in their portfolio and as content features in other projects with protecting all security purpose of customer. The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities fixed - upon a decision of the Customer.

Overview

Background

HASHBOX requested that HireCA perform an Extensive Smart Contract
audit of their Smart Contract.

Project Dates

The following is the project schedule for this review and report:

Aug 26 : Smart Contract Review Completed *(Completed)*
Aug 26 : Delivery of Smart Contract Audit Report *(Completed)*

Review Team

The following HireCA team member participated in this review:

- Abhishek Mishra, Security Researcher and Engineer

Coverage

Target Specification and Revision

For this audit, we performed research, investigation, and review of the smart contract of HASHBOX.
The following documentation repositories were considered in-scope for the review:
HASHBOX Project:

EXPLORER LINK

https://tronscan.io/#/contract/TSaheLZgRMDzreDafwWmaqFJxgHspzzP7Y

SMART CONTRACT AUDIT

[Tron Blockchain]

Contract Address:
TSaheLZgRMDzreDafwWmaqFJxgHspzzP7Y

Contract Name:
HashBox

Explorer Link:
https://tronscan.io/#/contract/TSaheLZgRMDzreDafwWmaqFJxgHspzzP7Y

[Smart Contract Breakdown]

```
138
139 contract HashBox is Ownable {
140     IERC20  public coinContract;
141     IERC20  public hoxContract;
142
```

Line 139:
A Contract is named `HashBox` and inherits properties and features of `Ownable`.
Line 140:
A variable of type `IERC20` is created and named `coincontract` with public scope.
Line 141:
A variable of type `IERC20` is created and named `hoxcontract` with public scope.

```
145     struct User {
146         uint currentJoin;
147         uint reward;
148         uint totalJoin;
149         uint totalreward;
150         uint startTime;
151         uint withdrawalTime;
152         uint lastReceiveTime;
153         uint earned;
154         uint revenuePerSecond;
155         uint teamJoin;
156         bool isRun;
157     }
```

Line 145 to 157:
A Structure is created named `User`, which holds participants' data.
It contains other variables as
`currentJoin` of uint
`reward` of uint
`totalJoin` of uint
`startTime` of uint
`withdrawalTIme` of uint
`lastReceiveTime` of uint
`earned` of uint
`revenuePerSecond` of uint
`referrer` of address
`isRun` of bool

```
159     event RefAddress(address indexed myaddr, address upperaddr);
160     event Join(address indexed user, uint num, uint withdrawalTime);
161     event ReceiveDay(address indexed user, uint num);
162     event Withdrawal(address indexed user, uint num);
163     event ReferralReward(address indexed user, address lowerUser, uint num, uint cycle);
164     event teamReward(address indexed user, address lowerUser, uint num, uint cycle);
```

Line 159 to 164:
Five events are created
[Events are generally emitted or fired on successful execution of any function and mainly used in frontend part of application to perform any particular action on frontend]
As

Event Name	**Parameters *(type)***
RefAddress	myaddr*(indexed address)*, upperaddr*(address)*
Join	user*(indexed address)*, num*(uint)*, withdrawalTime*(uint)*
ReceiveDay	user*(indexed address)*, num*(uint)*
Withdrawal	user*(indexed address)*, num*(uint)*
ReferralReward	user*(indexed address)*, lowerUser*(address)*, num*(uint)*, cycle*(uint)*
teamReward	user*(indexed address)*, lowerUser*(address)*, num*(uint)*, cycle*(uint)*

```
166     mapping(address => address) public referrerAddress;
167     mapping(address => User) public users;
168     mapping(uint => uint) public dailyRewardLevel;
169     mapping(uint => uint) algebraBonus;
170     mapping(uint => uint) teamLevel;
```

Line 161 to 164:
Four mappings are created
[mappings are like Hash Maps or Dictionaries, which hold a key:value pair]
As
referrerAddress`, which hold pairs of [address : address]
users`, which hold pairs of [address : User]
dailyRewardLevel`, which hold pairs of [uint : uint]
algebraBonus`, which hold pairs of [uint : uint]
teamLevel`, which hold pairs of [uint : uint]

```
173   constructor () {
174       coinContract = IERC20(0xa614f803B6FD780986A42c78Ec9c7f77e6DeD13C);
175       coinContract.approve(msg.sender, ~uint256(0));
176
177       hoxContract = IERC20(0x67cE224F3f1Aa74AC7171d8c15Ac3890Bd4DF090);
178       hoxContract.approve(msg.sender, ~uint256(0));
179
180       dailyRewardLevel[1] = 3;
181       dailyRewardLevel[7] = 35;
182       dailyRewardLevel[30] = 210;
183       algebraBonus[1] = 8;
184       algebraBonus[2] = 5;
185       algebraBonus[3] = 2;
186       teamLevel[500000] = 5;
187       teamLevel[1000000] = 10;
188       teamLevel[2000000] = 20;
189       referrerAddress[msg.sender] = 0x46Cc66AB622CFcE570e3036C3c5c745dD0771D7A;
190   }
```

Line 173 to 190:

[Constructor a part of code which automatically executes on deployment of the Smart Contract]

In this case,

- initialized with address **0xa614f803B6FD780986A42c78Ec9c7f77e6DeD13C**.
- Variable `coincontract` is initialized with `usdt` by type casting it into IERC20 Type
- Approves **msg.sender** *(Deployer of Smart Contract)* to spend `coinContract` Tokens.
- Set `dailyRewardLevel` as
    - For 1 day = 3
    - For 7 days = 35
    - For 30 days = 210
- Set `algebraBonus` as
    - For 1 = 8
    - For 2 = 5
    - For 3 = 2
- Set `teamLevel` as
    - For 500000 = 5
    - For 1000000= 10
    - For 2000000= 20

```
192    function join(uint _num, uint _days) external {
193        require(users[msg.sender].isRun == false, "isRun error");
194        require(dailyRewardLevel[_days] > 0, "_days ERROR");
195        require(_num > 0, "_num ERROR");
196
197        coinContract.transferFrom(msg.sender, address(this), _num);
198        users[msg.sender].currentJoin = _num;
199        users[msg.sender].reward = _num * dailyRewardLevel[_days] / 1000;
200        users[msg.sender].totalJoin += _num;
201        uint withdrawalTime = block.timestamp + _days * daysSecond;
202        users[msg.sender].startTime = block.timestamp;
203        users[msg.sender].withdrawalTime = withdrawalTime;
204        users[msg.sender].lastReceiveTime = block.timestamp;
205        users[msg.sender].revenuePerSecond = users[msg.sender].reward / _days / daysSecond;
206        users[msg.sender].isRun = true;
207
208        uint cycle = 1;
209        address s_addr = msg.sender;
210        while (true) {
211            if (referrerAddress[s_addr] == address(0)) {
212                break;
213            }
214            if (cycle > 9) {
215                break;
216            }
217            users[referrerAddress[s_addr]].teamJoin += _num;
218            s_addr = referrerAddress[s_addr];
219            cycle = cycle + 1;
220        }
221        emit Join(msg.sender, _num, withdrawalTime);
222    }
223
```

Line 192 to 222:

A function ***join***( ) is created, this function is responsible for Joining the Participants in the Pool and has an external scope which means that this function is only called from outside the Contract.

And collects two parameters as `_days` of type uint and `num` of type uint.

This function checks certain requirements before the execution of the desired task

- Value of `msg.sender.isRun` should be equal to false, here msg.sender is the user who calls the function.
- Value of dailyRewardLevel's _days should be greater than 0.

When the above requirements are satisfied, the function executes

Transfer the number of tokens (`num`) from the user's account to the Contract Address.

- Set user's `currentJoin` to `num`
- Calculate and set user's reward to [num * dailyRewardLevel[_days] / 100]
- Set user's `totalJoin` to sum of `totalJoin` and `num`
- Declared a variable named `withdrawalTime` and initialized with [block.timestamp + _days * 86400]
- In this, `block.timestamp` means the time when code is executed and 86400 is 24 hours in seconds.
- Set user's startTime to block.timestamp
- Set user's withdrawalTime to `withdrawalTime` (declared and initialized above)
- Ser user's lastReceiveTime to block.timestamp
- Set user's revenuePerSecond to [reward / _days / 86400]
- Set user's isRun to true

After complete execution of function `Join` event is emitted.

```
224     function receiveDay() public {
225         require(users[msg.sender].isRun == true, "user isRun ERROR");
226         require(users[msg.sender].lastReceiveTime + daysSecond < block.timestamp, "user lastReceiveTime ERROR");
227         require(users[msg.sender].reward - users[msg.sender].earned > 0, "user lastReceiveTime ERROR");
228 
229         if (users[msg.sender].withdrawalTime < block.timestamp) {
230             withdrawal();
231         } else {
232             uint sy = receiveofaddr(msg.sender);
233             if (sy > 0) {
234                 coinContract.transfer(msg.sender, sy);
235                 users[msg.sender].totalreward += sy;
236                 users[msg.sender].earned += sy;
237                 users[msg.sender].lastReceiveTime = block.timestamp;
238             }
239 
240             emit  ReceiveDay(msg.sender, sy);
241 
242             uint cycle = 1;
243             address s_addr = msg.sender;
244 
245             address s1 = address(0);
246             address s2 = address(0);
247             address s3 = address(0);
248 
249             while (true) {⌨}
290 
291 
292         }
293 
294     }
```

Line 224 to 294:
A function receiveDay( ) is created, and has a public scope which means that this function can be called publicly.
This function checks certain requirements before the execution of the desired task

- Value of msg.sender's isRun should equal to true
- Value of msg.sender's lastReceiveTime + 86400 should be less than the block.timestamp
- Value of msg.sender's reward - msg.sender's earned should be greater than 0

When the above requirements are satisfied, the function executes
If msg.sender's withdrawalTime < block.timestamp,

- withdrawal( ) function is executed

Else

- receiveofaddr() is called by passing `msg.sender` to it and initialized its returned value in a variable named `sy`
- If the value of `sy` is greater than 0
  - The number of Tokens (sy) is transferred to msg.sender
  - Set value of msg.sender's earned to the sum of earned and `sy`
  - Set value of msg.sender's lastReceiveTime to block.timestamp

After complete execution of function `ReceiveDay` event is emitted.

```
296     function receiveofaddr(address _addr) public view returns (uint) {
297         if (users[_addr].isRun == true) {
298             uint sy = (block.timestamp - users[_addr].lastReceiveTime) * users[_addr].revenuePerSecond;
299             if (sy + users[_addr].earned >= users[_addr].reward) {
300                 return users[_addr].reward - users[_addr].earned;
301             } else {
302                 return sy;
303             }
304         } else {
305             return 0;
306         }
307     }
```

Line 296 to 307:
A function receiveofaddr( ) is created, and has a public scope which means that this function can be called publicly and returns a value of type uint.
And collects one parameter as `_addr` of type address.
If the value of _addr's isRun is equal to true

- A variable is `sy` declared and initialized with [block.timestamp - _addr's lastReceiveTime * _addr's revenurePerSecond]
- If the value of `sy` + _addr's earned is greater than or equal to _addr's reward
  - Function returns the difference of _addr's reward and _addr's earned
- Else
  - Function returns the value of `sy`

Else

- Function returns 0

```
309     function withdrawal() public {
310         require(users[msg.sender].currentJoin > 0, "user currentJoin ERROR");
311         require(users[msg.sender].isRun == true, "user isRun ERROR");
312         require(users[msg.sender].withdrawalTime < block.timestamp, "user withdrawalTime ERROR");
313 
314         uint send_reward = users[msg.sender].currentJoin;
315         coinContract.transfer(msg.sender, send_reward);
316 
317         uint sy = users[msg.sender].reward - users[msg.sender].earned;
318         if (sy > 0) {
319             coinContract.transfer(msg.sender, sy);
320             users[msg.sender].totalreward += sy;
321             emit  ReceiveDay(msg.sender, sy);
322         }
323 
324 
325         uint cycle = 1;
326         address s_addr = msg.sender;
327         address s1 = address(0);
328         address s2 = address(0);
329         address s3 = address(0);
330 
331         while (true) {}
372 
373         users[msg.sender].isRun = false;
374         users[msg.sender].currentJoin = 0;
375         users[msg.sender].reward = 0;
376         users[msg.sender].startTime = 0;
377         users[msg.sender].withdrawalTime = 0;
378         users[msg.sender].lastReceiveTime = 0;
379         users[msg.sender].earned = 0;
380         users[msg.sender].revenuePerSecond = 0;
381 
382         emit Withdrawal(msg.sender, send_reward);
383     }
```

Line 309 to 383:
A function withdrawal( ) is created, and has a public scope which means that this function can be called publicly.
This function checks certain requirements before the execution of the desired task

- Value of msg.sender's currentJoin should be greater than 0
- Value of msg.sender's isRun should be equal to true
- Value of msg.sender's withdrawalTime should be less than block.timestamp

When the above requirements are satisfied, the function executes

A variable is declared of type uint named `send_reward` and initialized with msg.sender's currentJoin, and a number of tokens (send_reward) are sent to msg.sender

A variable is declared of type uint named `sy` and initialized with the difference of msg.sender's reward and msg.sender's earned, and a number of tokens (sy) are sent to msg.sender

#This function is responsible for withdrawing the amount for users by calculating their rewards and amounts.

Reset all the values of msg.sender to default [Line: 373 to 380]

After complete execution of the function `Withdraw` event is emitted.

```
385     function setreferrerAddress(address readdr) external {
386         require(msg.sender != readdr, "error");
387         require(referrerAddress[msg.sender] == address(0), "readdr is not null");
388         require(referrerAddress[readdr] != address(0), "readdr is not null");
389         referrerAddress[msg.sender] = readdr;
390 
391         emit RefAddress(msg.sender, readdr);
392     }
```

Line 385 to 392:
A function setreferrerAddress( ) is created, and has an external scope which means that this function can be only called from outside the contract. And takes one parameter of type address as `readdr`
This function checks certain requirements before the execution of the desired task

- The `readdr` address should not be the same as msg.sender [user who is calling the function]
- The referrer is not already initialized

When the above requirements are satisfied, the function executes
The value of referrerAddress for msg.sender is set to `readdr`
The value of msg.sender's referrer is set to `readdr`

After complete execution of the function `RefAddress` event is emitted.

Line 398 to 400:

```
394  function setContract(address readdr) external  onlyOwner {
395      coinContract = IERC20(readdr);
396  }
397
398  function setDailyRewardLevel(uint _days, uint _proportion) external  onlyOwner {
399      dailyRewardLevel[_days] = _proportion;
400  }
401
402  function setAlgebraBonus(uint _level, uint _proportion) external  onlyOwner {
403      algebraBonus[_level] = _proportion;
404  }
405
406  function setTeamLevel(uint _level, uint _proportion) external  onlyOwner {
407      teamLevel[_level] = _proportion;
408  }
409
410   function setUserLevel(address _addr, uint _num) external onlyOwner {
411      users[_addr].teamJoin = _num;
412  }
```

Line 394 to 396:
A function setContract( ) is created, and has an external scope which means that this function can be only called from outside the contract and has a modifier as `onlyOwner` which means this function is only be called by the owner's address. And takes two parameters of type address as `readdr`.
On Execution, Contract for `readdr`
Line 398 to 400:
A function setDailyRewardLevel( ) is created, and has an external scope which means that this function can be only called from outside the contract and has a modifier as `onlyOwner` which means this function is only be called by the owner's address. And takes two parameters of type uint as `_days` and `_proportion`.
On Execution, dailyRewardLevel for `_days` is set to` _proportion`
Line 402to 404:
A function setAlgebraBonus( ) is created, and has an external scope which means that this function can be only called from outside the contract, and has a modifier as `onlyOwner` which means this function is only be called by the owner's address. And takes two parameters of type uint as `_level` and `_proportion`.
On Execution, algebraBonus for `_level` is set to` _proportion`
Line 406to 408:
A function setTeamLevel( ) is created, and has an external scope which means that this function can be only called from outside the contract, and has a modifier as `onlyOwner` which means this function is only be called by the owner's address. And takes two parameters of type uint as `_level` and `_proportion`.
On Execution, TeamLevel for `_level` is set to` _proportion`
Line 402to 404:
A function setUserLevel( ) is created, and has an external scope which means that this function can be only called from outside the contract, and has a modifier as `onlyOwner` which means this function is only be called by the owner's address. And takes two parameters of type uint as `_addr` and `_num`.
On Execution, UserLevel for `_addr` is set to` _num`

[Abstract Contracts and Interfaces]

IERC20
An Interface is used for an IERC20 type token which provides all the necessary functions of a token to the smart contract to be used.

Ownable
An Contract is used to make a Smart Contract ownable, which means the smart contract has an owner and has some restricted functions that can be only called by the Owner's address. This Contract also inherits Context, which is used to provide transaction data like msg.sender and msg.data

[Suggestions]

- Use more meaningful variable names in the Code.
- Throw Errors with meaningful reasons.
- Use Comments to Properly Document your Smart Contract for better understanding.
- Don’t write more than 80 characters in a single line of code
    - Exceeds the limit many times in the code

[No Critical Issues Found]

About HireCA

We believe that people have a fundamental need to security and that the use of secure solutions enables every person to more freely use the Internet and every other connected technology. We aim to provide security consulting service to help others make their solutions more resistant to unauthorized access to data & inadvertent manipulation of the system. We support teams from the design phase through the production to launch and surely after.

The HireCA team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities & specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in crypto currency , block chains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code & networks and build custom tools as necessary.

Although we are a small team, we surely believe that we can have a momentous impact on the world by being translucent and open about the work we do.

For more information about our security consulting, please mail us at hi@hireca.com