FON Smart Chain

Smart Contract Security Audit

V1.0

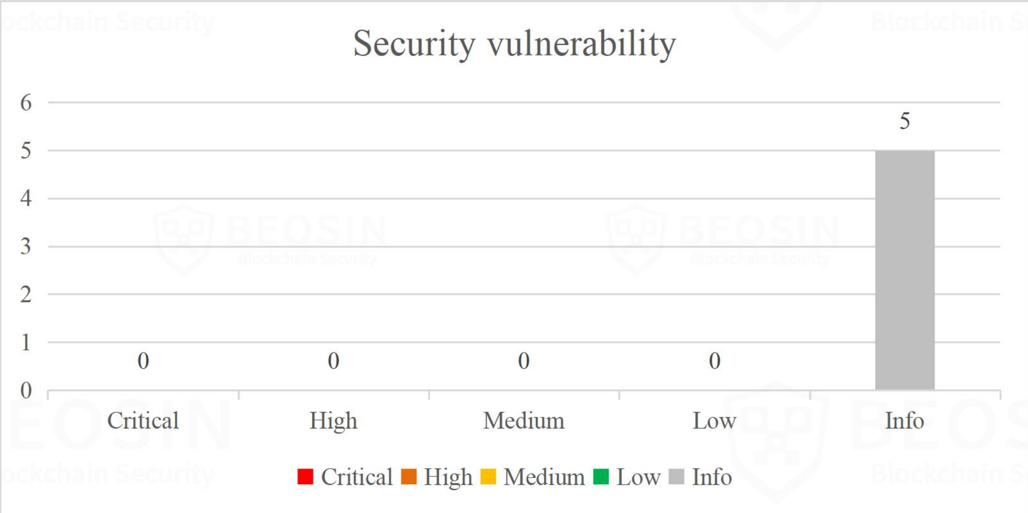
No. 202209291625

Sep 29th, 2022

Contents

Summary of audit results

After auditing, 5 Info items were identified in the FON Smart Chain project. Specific audit details will be presented in the **Findings section**. Users should pay attention to the following aspects when interacting with this project:



*Notes:

- **Project Description:**

1. Business overview

The FON Smart Chain project has reviewed a total of 6 contracts, which is an incremental review of the BNB smart chain(Github Link:https://github.com/bnb-chain/bsc-genesis-contract, Commit Hash: 9d45b31c12b2c04757284717f4351cb44e81a3a7). Among them, Vote contract and Multicall contract are newly added contracts, RelayerHub contract, TokenHub contract, BSCValidatorSet contract and CrossChain contract are slightly modified, and Staking contract deleted. The Vote contract mainly implements the sorting function of mortgage tokens and mining pools, and the Multicall contract mainly provides the function of calling other contracts from the current contract. The RelayerHub contract closes the registration function, the TokenHub contract deletes the withdrawal function provided for the Staking contract, and the CrossChain contract changes the handlePackage function that can only be accessed by registered users to owner user access. In the BSCValidatorSet contract, in addition to the fixed validator, the owner of the mining pool with the most stake will also become the validator and produce blocks.

1 Overview

1.1 Project Overview

Project Name	FSC	
Platform	FON Smart Chain	
File Hash(SHA-256)	Vote.sol	67492a48a6266e36ca68bfeb45c2b8faa9ef3237eb638a8dea2611e0df66aec3
	Multicall.sol	d31f24383a409170ba2d47eef028a4680923b08e5bd824f89caa06040f4d0d39
	RelayerHub.sol	ff7428116734dfae6a78dcf9c1378d9f672340b39063de63e5fecedcf82b44c5
	TokenHub.sol	f8722350a49c31944686a65a230a8ee1282b866b8b58d2efe931236d985821db
	CrossChain.sol	ff3e2e2ea69204a68ff103037b7f140a98b10b953ee5b5cd86408c8e102979e6(Initial) c13beaf76243e637322229fbb302108b073e2884a93fe8c532e901d0c290c310(Latest)
	BSCValidatorSet.sol	9496eb4e6ff3da7ad162a348b8b039747fc0ada319486f9cd09903c7625687be(Initial) 3e183b6a80ab622f10f9d6c065896c7cb3fcbed3bbe10f265af08af64416e2e8(Latest)

1.2 Audit Overview

Audit work duration: September 19, 2022 – September 29, 2022.

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Security Team.

2 Findings

Index	Risk description	Severity level	Status
FSC-1	Redundant Code	Info	Acknowledged
FSC-2	Missing event trigger	Info	Acknowledged
FSC-3	Inappropriate event naming	Info	Acknowledged
FSC-4	Validator refresh failure risk	Info	Fixed
FSC-5	Cannot register after deletion	Info	Acknowledged

Status Notes:

- FSC-1 is unfixed and will not cause any issue.
- FSC-2 is unfixed and will not cause any issue.
- FSC-3 is unfixed and will not cause any issue.
- FSC-5 is unfixed but does not cause security issues.

[FSC-1] Redundant Code

Severity Level	Info
Type	Coding Conventions
Lines	BSCValidatorSet.sol#L28-60&BSCValidatorSet.sol#L530-542
Description	The following code contains redundant parts.

```
28  uint256 constant public DUSTY_INCOMING = 1e17;
```

Figure 1 Source code of related variable

```
34  uint256 public constant PRECISION = 1e10;
```

Figure 2 Source code of related variable

```
59  address public constant BURN_ADDRESS = 0x000000000000000000000000000000000000dEaD;
60  uint256 public constant INIT_BURN_RATIO = 1000;
```

Figure 3 Source code of related variable

```
530  function checkValidatorSet(Validator[] memory validatorSet) private pure returns(bool, string memory) {
531    if (validatorSet.length > MAX_NUM_OF_VALIDATORS){
532      return (false, "the number of validators exceed the limit");
533    }
534    for (uint i; i < validatorSet.length; ++i) {
535      for (uint j = 0; j<i; j++) {
536        if (validatorSet[i].consensusAddress == validatorSet[j].consensusAddress) {
537          return (false, "duplicate consensus address of validatorSet");
538        }
539      }
540    }
541    return (true,"");
542  }
```

Figure 4 Source code of *checkValidatorSet* function

Recommendations	Remove redundant code.
Status	Acknowledged.

[FSC-2] Missing event trigger

Severity Level	Info
Type	Coding Conventions
Lines	BSCValidatorSet.sol#L291-300&CrossChain.sol#L100-102
Description	The following functions of the BSCValidatorSet contract and the CrossChain contract are missing events.

```
291 function transDev(address newDev) external {
292     require(msg.sender == dev, "only dev");
293     dev = newDev;
294 }
295
296 function transDevRate(uint256 newDevRate) external {
297     require(msg.sender == dev, "only dev");
298     require(newDevRate <= 3000);
299     devRate = newDevRate;
300 }
```

Figure 5 source code of *transDev* and *transDevRate* functions

```
100 function transferOwner(address newOwner) onlyOwner external{
101     owner = newOwner;
102 }
```

Figure 6 source code of *transferOwner* function

Recommendations	It is recommended to related functions to trigger events.
Status	Acknowledged.

[FSC-3] Inappropriate event naming

Severity Level	Info
Type	Coding Conventions
Lines	BSCValidatorSet.sol#L129-147
Description	The first letter of multiple events of the BSCValidatorSet contract is lowercase.Does not conform to the canonical naming of the event.

```
129    event validatorSetUpdated();
130    event validatorJailed(address indexed validator);
131    event validatorEmptyJailed(address indexed validator);
132    event batchTransfer(uint256 amount);
133    event batchTransferFailed(uint256 indexed amount, string reason);
134    event batchTransferLowerFailed(uint256 indexed amount, bytes reason);
135    event systemTransfer(uint256 amount);
136    event directTransfer(address payable indexed validator, uint256 amount);
137    event directTransferFail(address payable indexed validator, uint256 amount);
138    event deprecatedDeposit(address indexed validator, uint256 amount);
139    event validatorDeposit(address indexed validator, uint256 amount);
140    event validatorMisdemeanor(address indexed validator, uint256 amount);
141    event validatorFelony(address indexed validator, uint256 amount);
142    event failReasonWithStr(string message);
143    event unexpectedPackage(uint8 channelId, bytes msgBytes);
144    event paramChange(string key, bytes value);
145    event feeBurned(uint256 amount);
146    event validatorEnterMaintenance(address indexed validator);
147    event validatorExitMaintenance(address indexed validator);
```

Figure 7 The source code of related events

Recommendations	Change the first letter of the event to uppercase.
Status	Acknowledged.

[FSC-4] Validator refresh failure risk

Severity Level	Info
Type	Business Security
Lines	BSCValidatorSet.sol#L251-289
Description	If the dev address is the contract address and does not have the function of receiving payment or refuses to receive payment, will cause the *refreshValidators* function to fail.

```
251 function updateValidatorSet(Validator[] memory validatorSet) internal returns (uint32) {
252
253   // step 0: force all maintaining validators to exit `Temporary Maintenance`
254   // - 1. validators exit maintenance
255   // - 2. clear all maintainInfo
256   // - 3. get unjailed validators from validatorSet
257   Validator[] memory validatorSetTemp = _forceMaintainingValidatorsExit(validatorSet);
258
259   //step 1: do calculate distribution, do not make it as an internal function for saving gas.
260   uint validatorsNum = currentValidatorSet.length;
261
262   for (uint i; i < validatorsNum; ++i) {
263     if (currentValidatorSet[i].incoming > 0) {
264       bool success = currentValidatorSet[i].feeAddress.send(currentValidatorSet[i].incoming);
265       if (success) {
266         emit directTransfer(currentValidatorSet[i].feeAddress, currentValidatorSet[i].incoming);
267       } else {
268         emit directTransferFail(currentValidatorSet[i].feeAddress, currentValidatorSet[i].incoming);
269       }
270     }
271   }
272
273   // step 4: do dusk transfer
274   if (address(this).balance>0) {
275     emit systemTransfer(address(this).balance);
276     payable(dev).transfer(address(this).balance);
277   }
278   // step 5: do update validator set state
279   totalInComing = 0;
280   numOfJailed = 0;
281   if (validatorSetTemp.length>0) {
282     doUpdateState(validatorSetTemp);
283   }
284
285   // step 6: clean slash contract
286   // ISlashIndicator(SLASH_CONTRACT_ADDR).clean();
287   emit validatorSetUpdated();
288   return CODE_OK;
289 }
```

Figure 8 Source code of *updateValidatorSet* function (Unfixed)

Recommendations	transfer is changed to send to send tokens.
Status	Fixed.

```
function updateValidatorSet(Validator[] memory validatorSet) internal returns (uint32) {

  // step 0: force all maintaining validators to exit `Temporary Maintenance`
  // - 1. validators exit maintenance
  // - 2. clear all maintainInfo
  // - 3. get unjailed validators from validatorSet
  Validator[] memory validatorSetTemp = _forceMaintainingValidatorsExit(validatorSet);

  //step 1: do calculate distribution, do not make it as an internal function for saving gas.
  uint validatorsNum = currentValidatorSet.length;

  for (uint i; i < validatorsNum; ++i) {
    if (currentValidatorSet[i].incoming > 0) {
      bool success = currentValidatorSet[i].feeAddress.send(currentValidatorSet[i].incoming);
      if (success) {
        emit directTransfer(currentValidatorSet[i].feeAddress, currentValidatorSet[i].incoming);
      } else {
        emit directTransferFail(currentValidatorSet[i].feeAddress, currentValidatorSet[i].incoming);
      }
    }
  }

  // step 4: do dusk transfer
  if (address(this).balance>0) {
    emit systemTransfer(address(this).balance);
    payable(dev).send(address(this).balance);
  }
  // step 5: do update validator set state
  totalInComing = 0;
  numOfJailed = 0;
  if (validatorSetTemp.length>0) {
    doUpdateState(validatorSetTemp);
  }

  // step 6: clean slash contract
  // ISlashIndicator(SLASH_CONTRACT_ADDR).clean();
  emit validatorSetUpdated();
  return CODE_OK;
}
```

Figure 9 Source code of *updateValidatorSet* function (Fixed)

[FSC-5] Cannot register after deletion

Severity Level	Info
Type	Business Security
Lines	Vote.sol#L61-93

Description

In the Nodes contract, after calling the *unreg* function, the node address cannot call the *reg* function again because nodeAdded has been set to true.

```
60    // reg a new node need 9999 FON
61    function reg(string calldata name, string calldata url) external payable {
62      address owner = msg.sender;
63      require(!nodeAdded[owner], 'duplicate node');
64      require(msg.sender != address(0x29F31A69a450d9f0022dc410A2d48291D6a38244), 'genesis node');
65      require(msg.value == 9999 * 1e18);
66      nodeAdded[owner] = true;
67      uint256 index = nodeInfo.length;
68      nodeInfo.push(
69        NodeInfo({
70          owner: owner,
71          name: name,
72          url: url,
73          isActive: true,
74          totalStaked: 0,
75          rank: MAX,
76          index: index
77        })
78      );
79      nodeIndexFromOwner[owner] = index;
80      emit Reg(owner, name, url);
81    }
82
83    // unreg node
84    function unreg() public {
85      NodeInfo storage node = nodeInfo[getNodeIndexFromAddress()];
86      require(node.isActive, 'not active');
87      require(node.owner == msg.sender, 'only owner');
88      payable(msg.sender).transfer(999  9 * 1e18);
89      node.isActive = false;
90      rankDel(node.rank);
91      emit Close(msg.sender);
92    }
```

Figure 10 Source code of *reg* and *unreg* functions

Recommendations	It is recommended to allow users to register again.
Status	Acknowledged.

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact / Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.Unfixed Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in Blockchain.

3.4 About BEOSIN

BEOSIN is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions.BEOSIN has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, BEOSIN has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com