BEOSIN
Blockchain Security

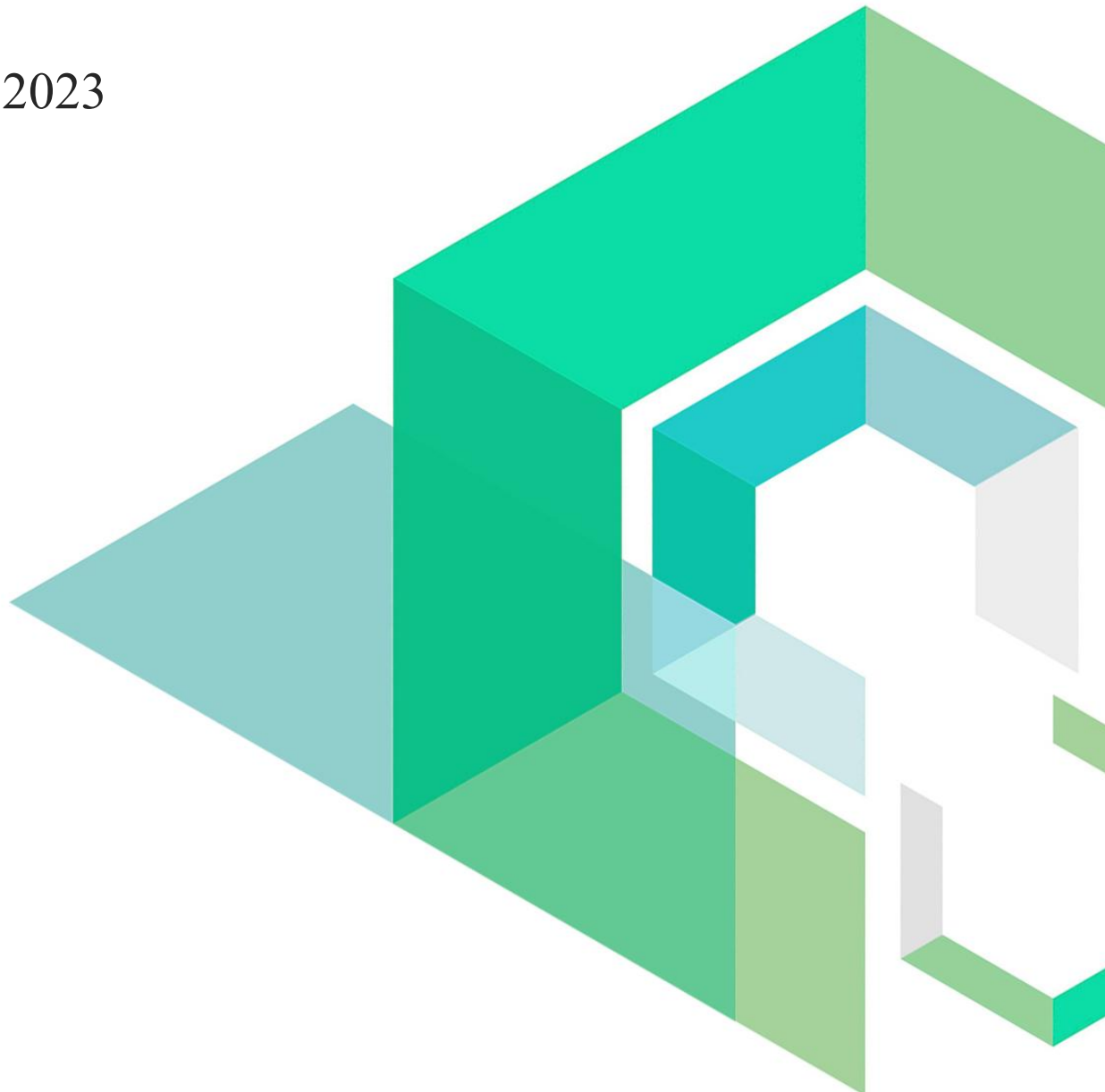
ESPL ARENA

Smart Contract Security Audit

V1.0

No. 202301111051

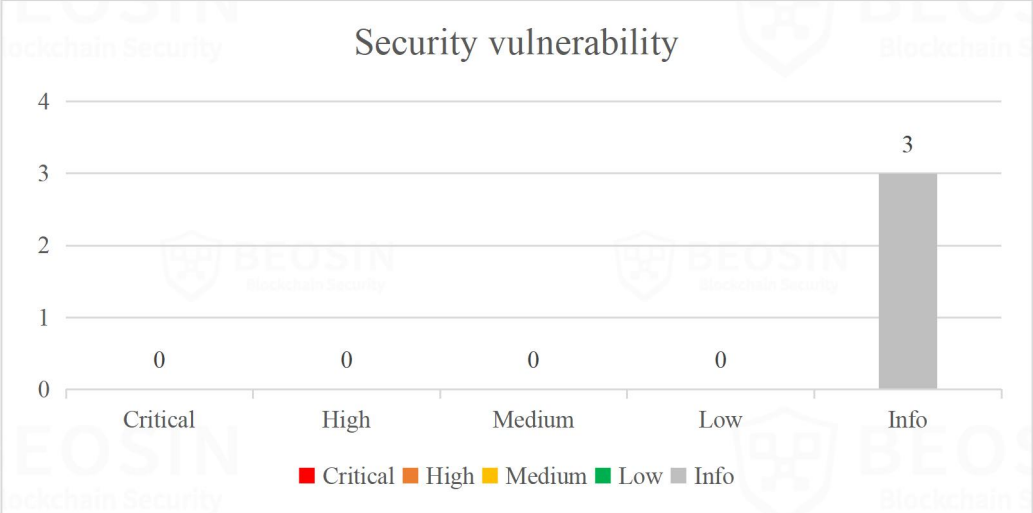
Jan 11th, 2023



Contents

Summary of Audit Results

After auditing, 3 Info-risk items were identified in the ESPL ARENA project. Specific audit details will be presented in the **Findings** section. Users should pay attention to the following aspects when interacting with this project:



Project Description:

1. Basic Token Information

Token name	ESPL ARENA
Token symbol	ARENA
Decimals	18
Pre-mint	0
Total supply	1 billion
Token type	BEP-20

Table 1 Basic information of ESPL ARENA

2. Business overview

The ESPL ARENA project is based on BEP-20 token that will be deployed on BNB chain. The initial mint is zero and the max supply of ESPL ARENA is 1 billion. ESPL ARENA token can be minted but can't be burned. The deployer will be granted DEFAULT_ADMIN_ROLE and MINTER_ROLE when the contract is deployed. The admin can add multiple Minter addresses, addresses with MINTER_ROLE can mint tokens up to the amount of MAXIMUMSUPPLY minus _totalSupply.

1 Overview

1.1 Project Overview

Project Name	ESPL ARENA
Platform	BSC
Audit scope	https://github.com/espl-co/ESPL-ARENA/blob/main/Contract/ESPL.sol
Commit Hash	f80a6e90370273b86b8612ed0f50b0ba9c3dc3c4 b9452305d79fb862a42dc7031548a3aeee4fae6c 862812b562c8f687162a4f11ab6792f6fdb9db0a
Contract Address	0xCfFD4D3B517b77BE32C76DA768634dE6C738889B

1.2 Audit Overview

Audit work duration: Jan 7, 2023 – Jan 11, 2023

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
ESPL ARENA-1	Centralization risk	Info	Fixed
ESPL ARENA-2	Improper role control design	Info	Fixed
ESPL ARENA-3	Redundant code	Info	Fixed

Finding Details:

[ESPL ARENA-1] Centralization risk

Severity Level	Info
Type	Business Security
Lines	ESPL.sol #L18-28
Description	The contract deployer grant MINTER_ROLE and DEFAULT_ADMIN_ROLE in the constructor. The admin can add multiple minter addresses, and address with MINTER_ROLE can mint tokens. There is a certain centralization risk.

```
18    constructor() ERC20("ESPL ARENA", "ARENA") {
19        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
20        _grantRole(MINTER_ROLE, msg.sender);
21    }
22
23    function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
24        require((_totalSupply+amount)<=MAXIMUMSUPPLY,"Maximum supply has been reached");
25        _totalSupply = _totalSupply.add(amount);
26        _balances[to] = _balances[to].add(amount);
27        _mint(to, amount);
28    }
```

Figure 1 Source code of *mint* function

Recommendations	It is recommended to use multi-signature wallet or DAO governance to manage admin and minter.
Status	Fixed.sent all ARENA tokens to 0x5d7EBbc57EdC38568A2A7F067CeAC9dc993978a3(multi-signature wallet)

[ESPL ARENA-2] Improper role control design

Severity Level	Info
Type	Business Security
Lines	ESPL.sol #L161-163
Description	An address with DEFAULT_ADMIN_ROLE permission can grant DEFAULT_ADMIN_ROLE and MINTER_ROLE to arbitrary addresses. But ROLE at the same level can revoke each other, for example, admin role can revoke role for other role address, include other admin role, which may cause permission management confusion.

```
150   /**
151    * @dev Revokes `role` from `account`.
152    *
153    * If `account` had been granted `role`, emits a {RoleRevoked} event.
154    *
155    * Requirements:
156    *
157    * - the caller must have ``role``'s admin role.
158    *
159    * May emit a {RoleRevoked} event.
160    */
161   function revokeRole(bytes32 role, address account) public virtual override onlyRole(getRoleAdmin(role)) {
162       _revokeRole(role, account);
163   }
164
165   /**
```

Figure 2 Source code of *_revokeRole* function

Recommendations	It is recommended to limit to only one DEFAULT_ADMIN_ROLE account, the admin can add multiple minter accounts, the minter account can reounce its own role but it cannot delete other people's MINTER_ROLE. The admin can remove the MINTER_ROLE.
Status	Fixed. The fixed code is shown in Figure 3. In the *_grantRole* function, admin can be added only once, and Minter can be added multiple times. In Figure 4, due to the condition of the code shown in the red box, Minter cannot renounce role by himself, and only be removed by admin role.

```
903   function _grantRole(bytes32 role, address account) internal virtual {
904       if (!hasRole(role, account)) {
905           if (role == DEFAULT_ADMIN_ROLE) {
906               if (!_defaultAdminRoleGranted) {
907                   _roles[role].members[account] = true;
908                   _defaultAdminRoleGranted = true;
909                   emit RoleGranted(role, account, _msgSender());
910               } else {
911                   revert(
912                       string(
913                           abi.encodePacked(
914                               "AccessControl: cannot have more than 1 DEFAULT_ADMIN_ROLE"
915                           )
916                       )
917                   );
918               }
919           }
920           else{
921               _roles[role].members[account] = true;
922               emit RoleGranted(role, account, _msgSender());
923           }
924       }
925   }
```

Figure 3 Source code of *_grantRole* function

```
934     function _revokeRole(bytes32 role, address account) internal virtual {
935         if (hasRole(role, account)) {
936             //check if caller has the same role as targeted revoke role
937             if (
938                 !hasRole(role, msg.sender) ||
939                 hasRole(DEFAULT_ADMIN_ROLE, msg.sender)
940             ) {
941                 //caller must not have the same role as targeted revoke role or must be DEFAULT_ADMIN_ROLE
942                 _roles[role].members[account] = false;
943                 emit RoleRevoked(role, account, _msgSender());
944             } else {
945                 revert(
946                     string(
947                         abi.encodePacked(
948                             "AccessControl: cannot revoke roles that are the same as caller unless caller is DEFAULT_ADMIN_ROLE"
949                         )
950                     )
951                 );
952             }
953         }
954     }
955 }
```

Figure 4 Source code of *_revokeRole* function

[ESPL ARENA-3] Redundant code

Severity Level	Info
Type	Coding Conventions
Lines	ESPL.sol #L14-163
Description	The contract already inherit from ERC20 which has variables _balances and _totalSupply. Variables _balances and _totalSupply in ESPLARENA contract is redundant.

```
9   contract ESPLARENA is ERC20, AccessControl {
10      using SafeMath for uint256;
11
12      bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
13
14      mapping(address => uint256) private _balances;
15      uint256 private _totalSupply;
16      uint256 constant MAXIMUMSUPPLY=1000000000*10**18;
17
18      constructor() ERC20("ESPL ARENA", "ARENA") {
19          _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
20          _grantRole(MINTER_ROLE, msg.sender);
21      }
22
23      function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
24          require((_totalSupply+amount)<=MAXIMUMSUPPLY,"Maximum supply has been reached");
25          _totalSupply = _totalSupply.add(amount);
26          _balances[to] = _balances[to].add(amount);
27          _mint(to, amount);
28      }
29
30      function totalSupply() public override view returns (uint256) {
31      return _totalSupply;
32      }
33
34      function maxSupply() public  pure returns (uint256) {
35      return MAXIMUMSUPPLY;
36      }
37  }
```

Figure 5 Source code of related code(unfixed)

Recommendations	It is recommended to delete redundant code.
Status	Fixed.

```
contract ESPLARENA is ERC20, AccessControl {
    using SafeMath for uint256;

    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");

    uint256 constant MAXIMUMSUPPLY=1000000000*10**18;

    constructor() ERC20("ESPL ARENA", "ARENA") {
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _grantRole(MINTER_ROLE, msg.sender);
    }

    function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
        require((totalSupply()+amount)<=MAXIMUMSUPPLY,"Maximum supply has been reached");
        _mint(to, amount);
    }
```

Figure 6 Source code of related code(fixed)

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.

Likelihood \ Impact	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.origin 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.

BEOSIN
Blockchain Security

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com