



# SECURITY ASSESSMENT LOOTED Lottery

February 25, 2024

Audit Status: Pass



BLADE POOL



## LOOTED

### Executive Summary

TYPES

DeFi

ECOSYSTEM

BNBCHAIN

LANGUAGE

undefined

### Timeline



**Audit Request**  
2024-02-25



**Onboarding Process**  
2024-02-25

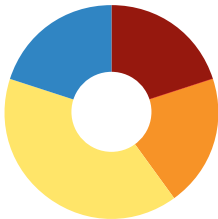


**Audit Preview**  
2024-02-25



**Audit Release**  
2024-02-25

### Vulnerability Summary



5

Total Findings

4

Resolved

1

Pending

1

Unresolved

#### 1 Critical

1 Resolved, 0 Pending



Critical risks are the most severe and can have a significant impact on the smart contracts functionality, security, or the entire system. These vulnerabilities can lead to the loss of user funds, unauthorized access, or complete system compromise.

#### 0 High

High-risk vulnerabilities have the potential to cause significant harm to the smart contract or the system. While not as severe as critical risks, they can still result in financial losses, data breaches, or denial of service attacks.

#### 1 Medium

1 Resolved, 0 Pending



Medium-risk vulnerabilities pose a moderate level of risk to the smart contracts security and functionality. They may not have an immediate and severe impact but can still lead to potential issues if exploited. These risks should be addressed to ensure the contracts overall security.

#### 2 Low

1 Resolved, 1 Pending



Low-risk vulnerabilities have a minimal impact on the smart contracts security and functionality. They may not pose a significant threat, but it is still advisable to address them to maintain a robust security posture.

## 1 Informational

1 Resolved, 0 Pending

Informational risks are not actual vulnerabilities but provide useful information about potential improvements or best practices. These findings may include suggestions for code optimizations, documentation enhancements, or other non-critical areas for improvement.

# PROJECT OVERVIEW | LOOTED.

## Token Summary

Parameter	Result
Address	
Name	LOOTED
Token Tracker	LOOTED (LOOTED)
Decimals	
Supply	
Platform	BNBCHAIN
Compiler	v0.8.20+commit.a1b79de6
Contract Name	Lottery
Optimization	Yes with 200 runs
LicenseType	Unlicensed
Language	undefined
Codebase	

## ■ Main Contract Assessed

Name	Contract	Live
LOOTED		Solidity

## ■ TestNet Contract Assessed

Name	Contract	Live
LOOTED	0x0Ecc81ce83C8ca1e0BA651b553203B622672dFeD	Solidity

## ■ Solidity Code Provided

SoIID	File Sha-1	FileName
LOOTED	8a5addcfc02a70a3576ac26e8ae9e7ef02ab046f	looted_final.sol

## Smart Contract Vulnerability Details | SWC-103 – Floating Pragma.

### CWE-664: Improper Control of a Resource Through its Lifetime.

#### References:

#### Description:

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

#### Remediation:

Lock the pragma version and also consider known bugs (<https://github.com/ethereum/solidity/releases>) for the compiler version that is chosen.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package.

Otherwise, the developer would need to manually update the pragma in order to compile locally.

#### References:

Ethereum Smart Contract Best Practices – Lock pragmas to specific compiler version.

## SMART CONTRACT VULNERABILITY DETAILS | LOOTED.

### | SWC-108 – State Variable Default Visibility.

#### CWE-710: Improper Adherence to Coding Standards

##### Description:

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

##### Remediation:

Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.

##### References:

Ethereum Smart Contract Best Practices – Explicitly mark visibility in functions and state variables

## TECHNICAL FINDINGS | LOOTED.

Smart contract security audits classify risks into several categories: Critical, High, Medium, Low, and Informational. These classifications help assess the severity and potential impact of vulnerabilities found in smart contracts.



### Classification of Risk

Severity	Description
 Critical	Critical risks are the most severe and can have a significant impact on the smart contracts functionality, security, or the entire system. These vulnerabilities can lead to the loss of user funds, unauthorized access, or complete system compromise.
 High	High-risk vulnerabilities have the potential to cause significant harm to the smart contract or the system. While not as severe as critical risks, they can still result in financial losses, data breaches, or denial of service attacks.
 Medium	Medium-risk vulnerabilities pose a moderate level of risk to the smart contracts security and functionality. They may not have an immediate and severe impact but can still lead to potential issues if exploited. These risks should be addressed to ensure the contracts overall security.
 Low	Low-risk vulnerabilities have a minimal impact on the smart contracts security and functionality. They may not pose a significant threat, but it is still advisable to address them to maintain a robust security posture.
 Informational	Informational risks are not actual vulnerabilities but provide useful information about potential improvements or best practices. These findings may include suggestions for code optimizations, documentation enhancements, or other non-critical areas for improvement.

By categorizing risks into these classifications, smart contract security audits can prioritize the resolution of critical and high-risk vulnerabilities to ensure the contract's overall security and protect user funds and data.



## LOOTED-03 | Lack of Input Validation.

Category	Severity	Location	Status
Volatile Code	 Low	looted_final.sol: L: 374 C:14	 Acknowledge

### Description

The given input is missing the check for the non-zero address.

The given input is missing the check for the onlyOwners need to be revisited for require..

### Recommendation

We advise the client to add the check for the passed-in values to prevent unexpected errors as below:

```
...
require(receiver != address(0), "Receiver is the zero address");
...
...
require(value X limitation, "Your not able to do this function");
...
```

We also recommend customer to review the following function that is missing a required validation. onlyOwners need to be revisited for require..

### Mitigation

#### References:

Zero Address check. The danger!!!

## FINDINGS

In this document, we present the findings and results of the smart contract security audit. The identified vulnerabilities, weaknesses, and potential risks are outlined, along with recommendations for mitigating these issues. It is crucial for the team to address these findings promptly to enhance the security and trustworthiness of the smart contract code.

Severity	Found	Pending	Resolved
<span>●</span> Critical	0	0	1
<span>●</span> High	0	0	0
<span>●</span> Medium	0	0	1
<span>●</span> Low	1	1	1
<span>i</span> Informational	0	0	1
Total	1	1	4

In a smart contract, a technical finding summary refers to a compilation of identified issues or vulnerabilities discovered during a security audit. These findings can range from coding errors and logical flaws to potential security risks. It is crucial for the project owner to thoroughly review each identified item and take necessary actions to resolve them. By carefully examining the technical finding summary, the project owner can gain insights into the weaknesses or potential threats present in the smart contract. They should prioritize addressing these issues promptly to mitigate any risks associated with the contract's security. Neglecting to address any identified item in the security audit can expose the smart contract to significant risks. Unresolved vulnerabilities can be exploited by malicious actors, potentially leading to financial losses, data breaches, or other detrimental consequences. To ensure the integrity and security of the smart contract, the project owner should engage in a comprehensive review process. This involves understanding the nature and severity of each identified item, consulting with experts if needed, and implementing appropriate fixes or enhancements. Regularly updating and maintaining the smart contract's codebase is also essential to address any emerging security concerns. By diligently reviewing and resolving all identified items in the technical finding summary, the project owner can significantly reduce the risks associated with the smart contract and enhance its overall security posture.

## SOCIAL MEDIA CHECKS | LOOTED.

Social Media		URL	Result
Website		<a href="https://Looted.net">https://Looted.net</a>	Pass
Telegram		<a href="https://t.me/LootedPortal">https://t.me/LootedPortal</a>	Pass
Twitter		<a href="https://X.com/LootedNetwork">https://X.com/LootedNetwork</a>	Pass
Facebook			N/A
Reddit	N/A		N/A
Instagram	N/A		N/A
CoinGecko	N/A		N/A
Github			N/A
CMC	N/A		N/A
Email	N/A		Contact
Other			Fail

From a security assessment standpoint, inspecting a project's social media presence is essential. It enables the evaluation of the project's reputation, credibility, and trustworthiness within the community. By analyzing the content shared, engagement levels, and the response to any security-related incidents, one can assess the project's commitment to security practices and its ability to handle potential threats.

### Social Media Information Notes:

**Auditor Notes:** Website needs a bit of improvement.

**Project Owner Notes:**

# ASSESSMENT RESULTS | LOOTED.

## Score Results

Review	Score
Overall Score	86/100
Auditor Score	82/100

Review by Section	Score
Manual Scan Score	23
SWC Scan Score	33
Advance Check Score	30

Our security assessment or audit score system for the smart contract and project follows a comprehensive evaluation process to ensure the highest level of security. The system assigns a score based on various security parameters and benchmarks, with a passing score set at 80 out of a total attainable score of 100. The assessment process includes a thorough review of the smart contracts codebase, architecture, and design principles. It examines potential vulnerabilities, such as code bugs, logical flaws, and potential attack vectors. The evaluation also considers the adherence to best practices and industry standards for secure coding. Additionally, the system assesses the projects overall security measures, including infrastructure security, data protection, and access controls. It evaluates the implementation of encryption, authentication mechanisms, and secure communication protocols. To achieve a passing score, the smart contract and project must attain a minimum of 80 points out of the total attainable score of 100. This ensures that the system has undergone a rigorous security assessment and meets the required standards for secure operation.



## Important Notes for LOOTED

- Please DYOR on the project itself.■
- VRF Chainlist requires a license to ensure the project is prepared.

**Auditor Score =82**  
**Audit Passed**



## Appendix

### Finding Categories

#### Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

#### Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

#### Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

#### Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

#### Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

#### Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

#### Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different requirements on the input variables than a setter function.

#### Coding Best Practices

ERC 20 Coding Standards are a set of rules that each developer should follow to ensure the code meets a set of criteria and is readable by all the developers.

## Disclaimer

The purpose of this disclaimer is to outline the responsibilities and limitations of the security assessment and smart contract audit conducted by Bladepool/CFG NINJA. By engaging our services, the project owner acknowledges and agrees to the following terms:

1. Limitation of Liability: Bladepool/CFG NINJA shall not be held liable for any damages, losses, or expenses incurred as a result of any contract malfunctions, vulnerabilities, or exploits discovered during the security assessment and smart contract audit. The project owner assumes full responsibility for any consequences arising from the use or implementation of the audited smart contract. 2. No Guarantee of Absolute Security: While Bladepool/CFG NINJA employs industry-standard practices and methodologies to identify potential security risks, it is important to note that no security assessment or smart contract audit can provide an absolute guarantee of security. The project owner acknowledges that there may still be unknown vulnerabilities or risks that are beyond the scope of our assessment. 3. Transfer of Responsibility: By engaging our services, the project owner agrees to assume full responsibility for addressing and mitigating any identified vulnerabilities or risks discovered during the security assessment and smart contract audit. It is the project owner's sole responsibility to ensure the proper implementation of necessary security measures and to address any identified issues promptly. 4. Compliance with Applicable Laws and Regulations: The project owner acknowledges and agrees to comply with all applicable laws, regulations, and industry standards related to the use and implementation of smart contracts. Bladepool/CFG NINJA shall not be held responsible for any non-compliance by the project owner. 5. Third-Party Services: The security assessment and smart contract audit conducted by Bladepool/CFG NINJA may involve the use of third-party tools, services, or technologies. While we exercise due diligence in selecting and utilizing these resources, we cannot be held liable for any issues or damages arising from the use of such third-party services. 6. Confidentiality: Bladepool/CFG NINJA maintains strict confidentiality regarding all information and data obtained during the security assessment and smart contract audit. However, we cannot guarantee the security of data transmitted over the internet or through any other means. 7. Not a Financial Advice: Bladepool/CFG NINJA please note that the information provided in the security assessment or audit should not be considered as financial advice. It is always recommended to consult with a financial professional or do thorough research before making any investment decisions.

By engaging our services, the project owner acknowledges and accepts these terms and releases Bladepool/CFG NINJA from any liability, claims, or damages arising from the security assessment and smart contract audit. It is recommended that the project owner consult legal counsel before entering into any agreement or contract.

