Lottery Smart Contract

SMART CONTRACT AUDIT REPORT

April 2025

ExVul

www.exvul.com

Table of Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by **Lottery** to review smart contract implementation. The assessment was conducted in accordance with our systematic approach to evaluate potential security issues based upon customer requirement. The report provides detailed recommendations to resolve the issue and provide additional suggestions or recommendations for improvement.
The outcome of the assessment outlined in chapter 3 provides the system's owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendations that will resolve the underlying technical issue.

1.1 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [10] which is the gold standard in risk assessment using the following risk models:

- Likelihood: represents how likely a particular vulnerability is to be uncovered and exploited in the wild.
- Impact: measures the technical loss and business damage of a successful attack.
- Severity: determine the overall criticality of the risk.

Likelihood can be: High, Medium and Low and impact are categorized into for: High, Medium, Low, Informational. Severity is determined by likelihood and impact and can be classified into five categories accordingly, Critical, High, Medium, Low, Informational shown in table 1.1.

Likelihood \ IMPACT	Informational	Low	Medium	High
High	Informational	Medium	High	Critical
Medium	Informational	Low	Medium	High
Low	Informational	Low	Low	Medium

Table 1.1 Overall Risk Severity

To evaluate the risk, we will be going through a list of items, and each would be labelled with a severity category. The audit was performed with a systematic approach guided by a comprehensive assessment list carefully designed to identify known and impactful security issues. If our tool or analysis does not identify any issue, the contract can be considered safe regarding the assessed item. For any discovered issue, we might further deploy contracts on our private test environment and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.2.

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Code and business security testing: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Category	Assessment Item
Basic Coding Assessment	Apply Verification Control
	Authorization Access Control
	Forged Transfer Vulnerability
	Forged Transfer Notification
	Numeric Overflow
	Transaction Rollback Attack
	Transaction Block Stuffing Attack
	Soft Fail Attack
	Hard Fail Attack
	Abnormal Memo
	Abnormal Resource Consumption
	Secure Random Number
Advanced Source Code Scrutiny	Asset Security
	Cryptography Security
	Business Logic Review
	Source Code Functional Verification
	Account Authorization Control
	Sensitive Information Disclosure
	Circuit Breaker
	Blacklist Control
	System API Call Analysis
	Contract Deployment Consistency Check
Additional Recommendations	Semantic Consistency Checks
	Following Other Best Practices

Table 1.2: The Full List of Assessment Items

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [14], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development.

2. FINDINGS OVERVIEW

2.1 Project Info And Contract Address

Project Name: Lottery

Audit Time: April 2, 2025 - April 4, 2025

Language: rust

Soure code	Link
Lottery	https://github.com/zikkkscc/lottery-rust-solana
Commit Hash	cfd21e95660d7c20dd0282e9e7dea96b82d8b493

2.2 Summary

Severity	Found	
Critical	2	
High	0	
Medium	1	
Low	1	
Informational	0	

2.3 Key Findings

ID	Severity	Findings Title	Status	Confirm
NVE- 001	Critical	Missing Recipient Account Validation	Acknowledge	Confirmed
NVE- 002	Critical	No Admin Access Control in init_config	Fixed	Confirmed
NVE- 003	Medium	Inadequate Validation in claim Function	Acknowledge	Confirmed
NVE- 004	Low	Unused Field signer_address in GlobalConfig	Acknowledge	Confirmed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Missing Recipient Account Validation

ID:	NVE-001	**Location:**	lib.rs
Severity:	Critical	**Category:**	Business Issues
Likelihood:	High	**Impact:**	Critical

Description:

The claim function in the program does not validate the ownership relationships between the token accounts and their claimed owners. Specifically, there is no validation to ensure that:
1. output_sec_account is owned by output_sec_origin_account.
2. output_third_account is owned by output_third_origin_account.
Although the code contains comments (///CHECK: Checked in origin address), no actual validation logic is implemented to enforce these ownership relationships. This lack of validation could lead to security risks, such as unauthorized access or manipulation of token accounts.

```
353     #[account(
354         mut,
355         token::mint = token_mint,
356     )]
357     pub output_sec_account: Box<Account<'info, TokenAccount>>,
358
359     #[account(
360         mut,
361         token::mint = token_mint,
362     )]
363     pub output_third_account: Box<Account<'info, TokenAccount>>,
364
365     ///CHECK: Checked in origin address
366     pub output_sec_origin_account: UncheckedAccount<'info>,
367     ///CHECK: Checked in origin address
368     pub output_third_origin_account: UncheckedAccount<'info>,
369
370     pub token_program: Program<'info, Token>,
371     pub system_program: Program<'info, System>,
372
373     /// CHECK: ix sign check
374     #[account(address = IX_ID)]
375     pub ix_sysvar: AccountInfo<'info>,
376 }
```

Recommendations:

Implement explicit ownership validation to ensure that the provided token accounts (output_sec_account and output_third_account) are indeed owned by their respective origin accounts (output_sec_origin_account and output_third_origin_account).

Result: Acknowledge

3.2 No Admin Access Control in init_config

ID:	NVE-002	Location:	lib.rs
Severity:	Critical	Category:	Business Issues
Likelihood:	High	Impact:	Critical

Description:

The init_config function lacks proper access control, allowing anyone to call it and set themselves as the admin. This vulnerability exposes the system to unauthorized administrative control, as any user can overwrite the admin address. Additionally, there is no mechanism to update the admin address if it is compromised, leading to permanent loss of control over the system.

```
18      pub fn init_config(
19          ctx: Context<InitConfig>,
20          signer_address: Pubkey,
21          token_mint_address: Pubkey,
22          lottery_fees: u64,
23      ) -> Result<()> {
24          let global_config = &mut ctx.accounts.global_config.load_init()?;
25          const ADMIN_PUBKEY:&str = "9nBEAzgig4PCbY2jyNfKLQM7uX51EpLsvg6ptGoHRPxW" ; // owner
26          require!(ctx.accounts.signer.key().to_string() == ADMIN_PUBKEY, StakeErrorCode::NotSigner);
27          global_config.signer_address = signer_address;
28          global_config.token_mint_address = token_mint_address;
29          global_config.lottery_fees = lottery_fees;
30          Ok(())
31      }
```

Recommendations:

Implement strict access control to restrict the init_config function to authorized users only. This can be achieved by adding a constraint that ensures only the current admin (or a predefined authority) can call this function.

Result: Fixed

3.3 Inadequate Validation in claim Function

ID:	NVE-003	Location:	lib.rs
Severity:	Medium	Category:	Business Issues
Likelihood:	Medium	Impact:	Medium

Description:

The claim function contains two notable issues:
1. No Minimum Amount Check: The function does not validate whether the amounts (amount_one, amount_two, amount_three) meet a minimum threshold. This could allow claims with trivial or zero amounts, potentially leading to unintended behavior or resource waste.
2. Independent Amount Validation Missing: While the function checks if the sum of amount_one, amount_two, and amount_three exceeds the pool's token balance, it fails to independently verify that each individual amount is greater than zero. This oversight could result in invalid or unintended distributions.

```
69      ) -> Result<()>{
70          let config = &mut ctx.accounts.global_config.load()?;
71          if amount_one + amount_two + amount_three > ctx.accounts.pool_token_account.amount {
72              return Err(ClaimErrorCode::InsufficientBalance.into());
73          }
74
75          let current_timestamp = Clock::get()?.unix_timestamp as u64;
76
77          if (timestamp + 300) < current_timestamp {
78              return Err(ClaimErrorCode::InvalidTimestamp.into());
79          }
80
81          let addr_nonce = ctx.accounts.address_manager.nonce + 1;
```

Recommendations:

Implement comprehensive validation to ensure robustness in the claim process:

1. Add Minimum Amount Check: Enforce a minimum threshold for each amount to prevent trivial or zero-value claims.

2. Validate Individual Amounts: Ensure each of amount_one, amount_two, and amount_three is greater than zero before processing the claim.

Result: Acknowledge

3.4 Unused Field signer_address in GlobalConfig

ID:	NVE-004	Location:	lib.rs
Severity:	Low	Category:	Business Issues
Likelihood:	Medium	Impact:	Low

Description:

The signer_address field in the GlobalConfig struct is set during the initialization of the configuration but is never used in the program. This unused field introduces unnecessary complexity and bloat to the configuration structure, potentially leading to confusion or misinterpretation of its purpose. Unused fields should be removed to maintain code clarity and efficiency.

```
18    pub fn init_config(
19        ctx: Context<InitConfig>,
20        signer_address: Pubkey,
21        token_mint_address: Pubkey,
22        lottery_fees: u64,
23    ) -> Result<()> {
24        let global_config = &mut ctx.accounts.global_config.load_init()?;
25        const ADMIN_PUBKEY:&str = "9nBEAzgig4PCbY2jyNfKLQM7uX51EpLsvg6ptGoHRPxW" ; // owner
26        require!(ctx.accounts.signer.key().to_string() == ADMIN_PUBKEY, StakeErrorCode::NotSigner);
27        global_config.signer_address = signer_address;
28        global_config.token_mint_address = token_mint_address;
29        global_config.lottery_fees = lottery_fees;
30        Ok(())
31    }
```

Recommendations:

Remove the unused signer_address field from the GlobalConfig struct unless it is explicitly required for future functionality. If it is intended for future use, document its purpose clearly to avoid confusion.

Result: Acknowledge

4. CONCLUSION

In this audit, we thoroughly analyzed **Lottery** smart contract implementation. The problems found are described and explained in detail in Section 3. The problems found in the audit have been communicated to the project leader. We therefore consider the audit result to be **PASSED**. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

5. APPENDIX

5.1 Basic Coding Assessment

5.1.1 Apply Verification Control

- Description: The security of apply verification
- Result: Not found
- Severity: Critical

5.1.2 Authorization Access Control

- Description: Permission checks for external integral functions
- Result: Not found
- Severity: Critical

5.1.3 Forged Transfer Vulnerability

- Description: Assess whether there is a forged transfer notification vulnerability in the contract
- Result: Not found
- Severity: Critical

5.1.4 Transaction Rollback Attack

- Description: Assess whether there is transaction rollback attack vulnerability in the contract.
- Result: Not found
- Severity: Critical

5.1.5 Transaction Block Stuffing Attack

- Description: Assess whether there is transaction blocking attack vulnerability.
- Result: Not found
- Severity: Critical

5.1.6 Soft Fail Attack Assessment

- Description: Assess whether there is soft fail attack vulnerability.
- Result: Not found
- Severity: Critical

5.1.7 Hard Fail Attack Assessment

- Description: Examine for hard fail attack vulnerability
- Result: Not found
- Severity: Critical

5.1.8 Abnormal Memo Assessment

- Description: Assess whether there is abnormal memo vulnerability in the contract.
- Result: Not found
- Severity: Critical

5.1.9 Abnormal Resource Consumption

- Description: Examine whether abnormal resource consumption in contract processing.
- Result: Not found
- Severity: Critical

5.1.10 Random Number Security

- Description: Examine whether the code uses insecure random number.
- Result: Not found
- Severity: Critical

5.2 Advanced Code Scrutiny

5.2.1 Cryptography Security

- Description: Examine for weakness in cryptograph implementation.
- Results: Not Found
- Severity: High

5.2.2 Account Permission Control

- Description: Examine permission control issue in the contract
- Results: Not Found
- Severity: Medium

5.2.3 Malicious Code Behavior

- Description: Examine whether sensitive behavior present in the code
- Results: Not found
- Severity: Medium

5.2.4 Sensitive Information Disclosure

- Description: Examine whether sensitive information disclosure issue present in the code.
- Result: Not found
- Severity: Medium

5.2.5 System API

- Description: Examine whether system API application issue present in the code
- Results: Not found
- Severity: Low

6. 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 Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without ExVul's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts ExVul to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. ExVul's position is that each company and individual are responsible for their own due diligence and continuous security. ExVul's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

7. REFERENCES

[1] MITRE. CWE- 191: Integer Underflow (Wrap or Wraparound).

https://cwe.mitre.org/data/ definitions/191.html.

[2] MITRE. CWE- 197: Numeric Truncation Error.

https://cwe.mitre.org/data/definitions/197. html.

[3] MITRE. CWE-400: Uncontrolled Resource Consumption.

https://cwe.mitre.org/data/ definitions/400.html.

[4] MITRE. CWE-440: Expected Behavior Violation.

https://cwe.mitre.org/data/definitions/440. html.

[5] MITRE. CWE-684: Protection Mechanism Failure.

https://cwe.mitre.org/data/definitions/ 693.html.

[6] MITRE. CWE CATEGORY: 7PK - Security Features.

https://cwe.mitre.org/data/definitions/ 254.html.

[7] MITRE. CWE CATEGORY: Behavioral Problems.

https://cwe.mitre.org/data/definitions/438. html.

[8] MITRE. CWE CATEGORY: Numeric Errors.

https://cwe.mitre.org/data/definitions/189.html.

[9] MITRE. CWE CATEGORY: Resource Management Errors.

https://cwe.mitre.org/data/ definitions/399.html.

[10] OWASP. Risk Rating Methodology.

https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

www.exvul.com

contact@exvul.com

@EXVULSEC

github.com/EXVUL-Sec

ExVul