



# AUDIT REPORT




---

February 2025

For

**enjoyors**

# Table of Content

Executive Summary	03
Number of Security Issues per Severity	05
Checked Vulnerabilities	06
Techniques and Methods	07
Types of Severity	09
Types of Issues	10
 <b>High Severity Issues</b>	11
1. Denial of Service via Pre-initialization of Associated Token Account	11
 <b>Low Severity Issues</b>	14
2. Insecure Initialization Leading to Unauthorized Configuration Control	14
3. Lack of Two-Step Ownership Transfer for RoleAdmin Role	15
 <b>Informational Issues</b>	16
4. Approver Data Not Utilized in claim_withdrawal Function	16
Closing Summary & Disclaimer	17



# Executive Summary

<b>Project Name</b>	Enjoyoors
<b>Overview</b>	Enjoyoors is a DeFi platform that streamlines liquidity and yield optimization across multiple asset classes, including cryptocurrencies, governance tokens, and DeFi derivatives. It enables users to stake diverse assets, earn weekly yields, and withdraw funds anytime, all while simplifying transactions by eliminating the need for wallet or network switching. Positioned as an “Omni-asset Liquidity Primitive,” Enjoyoors aims to enhance liquidity access and usability for the evolving DeFi ecosystem.
<b>Timeline</b>	21st January 2025 - 25th January 2025
<b>Updated Code Received</b>	03rd February 2025
<b>Review 2</b>	04th February 2025
<b>Method</b>	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
<b>Audit Scope</b>	The scope of this audit was to analyse the Solana Programs for quality, security, and correctness.
<b>Blockchain</b>	Solana
<b>Source Code</b>	<a href="https://github.com/eq-lab/enjoyoors-solana-programs"><u>https://github.com/eq-lab/enjoyoors-solana-programs</u></a>
<b>Contracts in Scope</b>	Vault & Withdrawal Approver

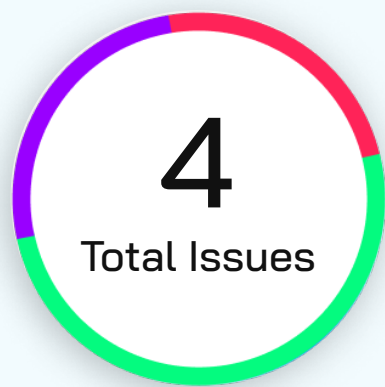


# Executive Summary

Branch	Main
Commit Hash	740bd53e25d0e1a0fdb9e325f01226b45f92cdf1
Fixed In	01e9fa0300390e4fe214328b5d99a4a0dca0a132



# Number of Issues per Severity



High	1 (25%)
Medium	0 (0%)
Low	2 (50%)
Informational	1 (25%)

		Severity			
		High	Medium	Low	Informational
Issues	Open	0	0	0	0
	Resolved	1	0	2	0
	Acknowledged	0	0	0	1
	Partially Resolved	0	0	0	0



# Checked Vulnerabilities

We have scanned the solana program for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

✓ **Signer authorization**

✓ **Account data matching**

✓ **Sysvar address checking**

✓ **Owner checks**

✓ **Type cosplay**

✓ **Initialization**

✓ **Arbitrary cpi**

✓ **Duplicate mutable accounts**

✓ **Bump seed canonicalization**

✓ **PDA Sharing**

✓ **Incorrect closing accounts**

✓ **Missing rent exemption checks**

✓ **Arithmetic overflows/underflows**

✓ **Numerical precision errors**

✓ **Solana account confusions**

✓ **Casting truncation**

✓ **Insufficient SPL token account verification**

✓ **Signed invocation of unverified programs**



# Techniques and Methods

Throughout the audit of Solana Programs, care was taken to ensure:

- The overall quality of code.
- Use of best practices
- Code documentation and comments, match logic and expected behavior
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Implementation of ERC standards
- Efficient use of gas
- Code is safe from re-entrancy and other vulnerabilities

The following techniques, methods, and tools were used to review all the smart contracts:

## Structural Analysis

In this step, we have analysed the design patterns and structure of Solana programs. A thorough check was done to ensure the Solana program is structured in a way that will not result in future problems.

## Static Analysis

Static analysis of Solana programs was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of Solana programs.



# Techniques and Methods

## Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, and their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

## Gas Consumption

In this step, we have checked the behaviour of Solana programs in production. Checks were done to know how much gas gets consumed and the possibilities of optimising code to reduce gas consumption.





# Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

## High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

## Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

## Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

## Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.



# Types of Issues

## Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

## Resolved

These are the issues identified in the initial audit and have been successfully fixed.

## Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

## Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

# High Severity Issues

## 1. Denial of Service via Pre-initialization of Associated Token Account

**Resolved**

### Path

programs/vault/src/instructions/admin/add\_token.rs

### Description

The "add\_token" function in the Enjoyoors protocol is designed to whitelist new tokens for deposits by initializing several critical accounts. Among these accounts, the vault\_token\_account is an Associated Token Account (ATA) tied to the token\_config Program Derived Address (PDA). This account ensures each token has a properly configured vault within the protocol.

However, implementing the function allows a malicious user to pre-initialize the vault\_token\_account ATA for a specific token\_config PDA before the add\_token function is invoked. Since the ATA is already initialized, the subsequent attempt by the legitimate process to initialize the same ATA fails

This vulnerability enables a Denial of Service (DoS) attack, effectively blocking the whitelisting of targeted tokens.

### Exploitation Flow

1. A malicious actor identifies a token mint they wish to target.
2. They compute the token\_config PDA based on the token mint.
3. Using their wallet, they pre-initialize the ATA (vault\_token\_account) associated with the token\_config PDA and the token mint.
4. When the legitimate add\_token function attempts to initialize the already-existing ATA, it encounters an error and fails, thereby preventing the token from being added to the whitelist.

### Impact

1. **Denial of Service:** The protocol's administrators cannot whitelist certain tokens, rendering these tokens unusable within the system.
2. **Disruption of Deposits:** Users cannot deposit the targeted tokens into the vault, potentially affecting liquidity and protocol usability.
3. **Attack Scalability:** This attack can be repeated for multiple tokens, causing widespread disruption to the protocol's functionality.



### Proof of Concept (PoC)

The following test case demonstrates the vulnerability:

```
it("creates mint and pre-initializes associated token account for token config PDA",  
  async () => {
```

**// Step 1: Create a new token mint account**

```
    const token_mint_account = await splToken.createMint(  
      programProvider.connection,  
      signer,  
      signer.publicKey,  
      signer.publicKey,  
      9 // Decimal places for the token  
    );
```

**// Step 2: Derive the token\_config PDA address based on the token mint**

```
    const token_config = anchor.web3.PublicKey.findProgramAddressSync(  
      [Buffer.from("token_config"), token_mint_account.toBuffer()],  
      program.programId  
    );
```

**// Step 3: Log the derived token\_config PDA for debugging**

```
    console.log(token_config[0]);
```

**// Step 4: Pre-initialize the ATA for the token\_config PDA**

```
    const token_config_ata = await splToken.getOrCreateAssociatedTokenAccount(  
      programProvider.connection,  
      signer,  
      token_mint_account,  
      token_config[0],  
      true  
    );
```

**// Step 5: Grant TOKEN\_LISTER\_ROLE to the admin**

```
    const role = { tokenListerRole: {} };  
    await program.methods.grantRole(role, signer.publicKey).accounts({  
      signer: signer.publicKey,  
    }).signers([signer]).rpc();
```

**// Step 6: Attempt to add the token to the whitelist**

**// This will fail because the ATA was already pre-initialized**

```
    await program.methods.addToken().accounts({  
      signer: signer.publicKey,  
      tokenMint: token_mint_account,  
    }).signers([signer]).rpc();  
  });
```





# Low Severity Issues

## 2. Insecure Initialization Leading to Unauthorized Configuration Control

**Resolved**

### Path

programs/vault/src/instructions/admin/init.rs  
programs/withdrawal-approver/src/instructions/admin/init.rs

### Description

In Solana programs, initialization functions are critical for setting up essential state variables and accounts required for the program's operation. Solana programs rely on manually invoked initialization functions, which introduces potential risks.

The Vault and Withdrawal Approver programs feature insecure initialize functions that allow any signer to invoke the initialization process. These functions are responsible for setting the program's administrative authority (admin) and other essential parameters. However, there is no mechanism in place to restrict these functions to authorized users, such as the program's upgrade\_authority.

### Recommendation

To mitigate this issue, enforce constraints in the initialize function to ensure only the authorized account call it such as program's upgrade\_authority.

### Reference

[https://docs.rs/anchor-lang/latest/anchor\\_lang/accounts/account/struct.Account.html#example-1](https://docs.rs/anchor-lang/latest/anchor_lang/accounts/account/struct.Account.html#example-1)



### 3. Lack of Two-Step Ownership Transfer for RoleAdmin Role

**Resolved**

#### Path

programs/vault/src/instructions/admin/grant\_role.rs  
programs/withdrawal-approver/src/instructions/admin/grant\_role.rs

#### Description

The RoleAdmin role in the protocol is critical as it grants administrative privileges, including the ability to manage other roles within the system. During initialization, the RoleAdmin role is assigned to the admin address provided in the initialize function. This role can then be transferred to another address using the grant\_role function.

However, the current implementation lacks safeguards to prevent accidental or malicious reassignment of the RoleAdmin role. If the RoleAdmin role is mistakenly transferred to an unintended or malicious address, it could lead to a complete loss of administrative control over the protocol.

#### Recommendation

A secure approach to mitigating this issue is to create a two-step process for transferring authority. This process would allow the current authority to nominate a new pending\_authority, which must explicitly accept the role. Not only would this provide authority transfer functionality, but it would also protect against accidental transfers or malicious takeovers.



# Informational Issues

## 4. Approver Data Not Utilized in claim\_withdrawal Function

Acknowledged

### Path

programs/vault/src/instructions/user/claim\_withdrawal.rs

### Description

The claim\_withdrawal function includes an approver\_data parameter that is intended to be used during the withdrawal verification process by the withdrawal\_approver\_program. However, the approver\_data is not actively utilized in the function, rendering it redundant.

### Recommendation

**Remove the approver\_data Parameter:** If the approver\_data is not required, it should be removed from the function signature and related structures to simplify the code.





# Closing Summary

In this report, we have considered the security of the Enjoyoors Solana Programs. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

# Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the submitted smart contract source code, including its compilation, deployment, and intended functionality.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats. Stay proactive. Stay secure.



# About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over \$30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

**6+**

Years of Expertise

**1M+**

Lines of Code Audited

**\$30B+**

Secured in Digital Assets

**1K+**

Projects Secured

Follow Our Journey



# AUDIT REPORT

---

February 2025

For

**enjoyors**



Canada, India, Singapore, UAE, UK

[www.quillaudits.com](http://www.quillaudits.com)

[audits@quillaudits.com](mailto:audits@quillaudits.com)