QuillAudits

AUDIT REPORT

February, 2025

For

Table of Content

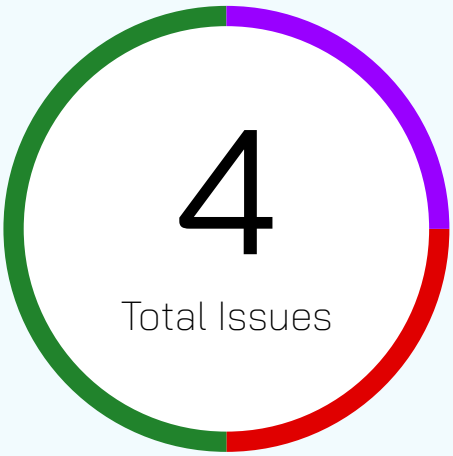
Table of Content

Executive Summary

Project name	Enjoyoors
Overview	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.
Timeline	21 Jan 2025 - 25 Jan 2025
Update code Received	2025-02-03
Second Review	04 Feb 2025
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Blockchain	Solana
Audit Scope	The scope of this audit was to analyse the Solana Programs for quality, security, and correctness.
Source code	.rs file was provided
Contracts In Scope	Vault & Withdrawal Approver

Branch	NA
Fixed In	1 Week
Mainnet address	WithdrawalApprover: DviTbaDaneHwazX2TGSMY3wyvxjVmYVZyRCXEiYYAuYb Vault: D1RK3JevvGmF3tYNRbuSphAG6AZtxaDkUYxd-Fp9F9Uhy

Number of Issues per Severity



High	1 (25.00%)
Medium	0 (0.00%)
Low	2 (50.00%)
Informational	1 (25.00%)

Severity

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

Checked Vulnerabilities

Techniques and Methods

Throughout the audit of smart contracts, 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 analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

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 analyzed, 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 behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools And Platforms Used For Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistical analysis.

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 Issues

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

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

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

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.

Recommendation

To mitigate this issue, replace the init constraint in the vault_token_account initialization logic with the init_if_needed constraint. This ensures the account is initialized only if it does not already exist. If the account is already initialized, it can be used directly without attempting reinitialization, thereby resolving the reinitialization conflict and preventing errors caused by pre-initialized accounts.

POC

The following test case demonstrates the vulnerability:

```
29
30      // Step 5: Grant TOKEN_LISTER_ROLE to the admin
31      const role = { tokenListerRole: {} };
32      await program.methods.grantRole(role, signer.publicKey).accounts({
33          signer: signer.publicKey,
34      }).signers([signer]).rpc();
35
36      // Step 6: Attempt to add the token to the whitelist
37      // This will fail because the ATA was already pre-initialized
38      await program.methods.addToken().accounts({
39          signer: signer.publicKey,
40          tokenMint: token_mint_account,
41      }).signers([signer]).rpc();
42  });
43
```

```
1   it("creates mint and pre-initializes associated token account for token config PDA", as
2
3       // Step 1: Create a new token mint account
4       const token_mint_account = await splToken.createMint(
5           programProvider.connection,
6           signer,
7           signer.publicKey,
8           signer.publicKey,
9           9 // Decimal places for the token
10      );
11
12      // Step 2: Derive the token_config PDA address based on the token mint
13      const token_config = anchor.web3.PublicKey.findProgramAddressSync(
14          [Buffer.from("token_config"), token_mint_account.toBuffer()],
15          program.programId
16      );
17
18      // Step 3: Log the derived token_config PDA for debugging
19      console.log(token_config[0]);
20
21      // Step 4: Pre-initialize the ATA for the token_config PDA
22      const token_config_ata = await      splToken.getOrCreateAssociatedTokenAccount(
23          programProvider.connection,
24          signer,
25          token_mint_account,
26          token_config[0],
27          true
28      );
```

```
29
30      // Step 5: Grant TOKEN_LISTER_ROLE to the admin
31      const role = { tokenListerRole: {} };
32      await program.methods.grantRole(role, signer.publicKey).accounts({
33          signer: signer.publicKey,
34      }).signers([signer]).rpc();
35
36      // Step 6: Attempt to add the token to the whitelist
37      // This will fail because the ATA was already pre-initialized
38      await program.methods.addToken().accounts({
39          signer: signer.publicKey,
40          tokenMint: token_mint_account,
41      }).signers([signer]).rpc();
42  });
43
```



```
1   it("creates mint and pre-initializes associated token account for token config PDA", as
2
3       // Step 1: Create a new token mint account
4       const token_mint_account = await splToken.createMint(
```

Low Severity Issues

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

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 Severity Issues

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 secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

February, 2025

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com