

# Assure DeFi<sup>®</sup>

THE VERIFICATION **GOLD STANDARD**



## Security Assessment

### BlindFaith

Date: 20/03/2025

Audit Status: PASS

Audit Edition: Solana



ASSURE DEFI<sup>®</sup>  
THE VERIFICATION **GOLD STANDARD**

# Risk Analysis

## Vulnerability summary

Classification	Description
 High	High-level vulnerabilities can result in the loss of assets or manipulation of data.
 Medium	Medium-level vulnerabilities can be challenging to exploit, but they still have a considerable impact on smart contract execution, such as allowing public access to critical functions.
 Low	Low-level vulnerabilities are primarily associated with outdated or unused code snippets that generally do not significantly impact execution, sometimes they can be ignored.
 Informational	Informational vulnerabilities, code style violations, and informational statements do not affect smart contract execution and can typically be disregarded.

## Executive Summary

According to the Assure assessment, the Customer's smart contract is **Secured**.



# Scope

## Target Code And Revision

For this audit, we performed research, investigation, and review of the BlindFaith verifying the functional and superficial part of the contract since it is generated in SPL token creator and we do not have access to the base code.

## Target Code And Revision

Project	Assure
Language	Rust
Codebase	<a href="https://github.com/saosci/blindfaith_program">https://github.com/saosci/blindfaith_program</a> Commit: d8e645c176cf44dea7a323b70903b8e4ed57dc cd Fixed version: <a href="https://github.com/saosci/blindfaith_program/commit/bb53a5b624063b63c42b3e1e8951be6bc0dcd59e">https://github.com/saosci/blindfaith_program/commit/bb53a5b624063b63c42b3e1e8951be6bc0dcd59e</a>
Audit Methodology	Static, Manual



# AUDIT OVERVIEW



## 1. Unrestricted PDA Cleanup [Solved ✓]

**Location:** cleanup\_competition\_tickets instruction (end of the file)

**Issue:** This function loops over all remaining\_accounts and unconditionally transfers their entire lamport balance to destination before zeroing them out:

```
for ticket_account in ctx.remaining_accounts.iter() {
    let ticket_info = ticket_account.to_account_info();
    let mut ticket_lamports = ticket_info.try_borrow_mut_lamports()?;
    let lamports_to_transfer = **ticket_lamports;
    if lamports_to_transfer > 0 {
        **ctx.accounts.destination.to_account_info().lamports.borrow_mut() +=
lamports_to_transfer;
        **ticket_lamports = 0;
    }
}
```

There is no check that these ticket\_accounts:

- Are PDAs derived with the correct seeds for your Ticket type.
- Actually belong to this competition.

A malicious caller can pass any accounts (e.g., the protocol\_fee\_account, global\_jackpot, or random user accounts) as “tickets” and forcibly drain them into their destination.

**Recommendation:** Enforce that each ticket\_account is a valid Ticket for this specific competition or parse the account data to confirm it matches the Ticket type's discriminator. Also verify the account is owned by the program.

**Fix:** The cleanup function now verifies that each account:

1. Is owned by the program
2. Can be deserialized as a valid Ticket
3. Has a competition\_id matching the current competition
4. Has an expected PDA derived from [competition.key, participant.key]

## **2. No Ownership Verification of Randomness Accounts [Solved ✓]**

**Location:**

- request\_randomness (RequestRandomness context)
- select\_winner (SelectWinner context)
- request\_global\_jackpot\_randomness
- global\_jackpot\_select\_winner

**Issue:** The program uses RandomnessAccountData::parse(...) to parse the “Switchboard On-Demand” account. However, no code checks that:

randomness\_account\_data.owner == <the real Switchboard program ID>,

or that the pubkey for the aggregator is correct.

This means an attacker can create a fake account with arbitrary data that “simulates” Switchboard’s layout and returns their chosen random bytes. The program will parse them, seeing a plausible seed\_slot, revealed\_random\_value, etc., enabling the attacker to control the winning ticket.

**Recommendation:** Add an ownership check where switchboard\_program\_id is a known constant and optionally check aggregator details, last update slot, or signatures from the Switchboard aggregator.

**Fix:** Added explicit checks (using require!) to ensure the randomness\_account\_data.owner equals the ON\_DEMAND\_PROGRAM\_ID for both competition and global jackpot randomness requests and winner selections



## **1. Weak Winner Selection Randomness (Only 8 Bits) [Solved ✓]**

**Location:** select\_winner and global\_jackpot\_select\_winner

**Issue:** Both the competition and global jackpot only use the first byte of the random seed (revealed\_random\_value[0]). This yields at most 256 distinct outcomes for the winning ticket.

If total\_tickets is much larger than 256, the random distribution is highly biased and predictable.

Even if total\_tickets < 256, attackers can exploit partial control, especially combined with the lack of aggregator ownership checks.

**Recommendation:** Use more entropy.

**Fix:** Instead of using only one byte, the updated code extracts the first 8 bytes to form a 64-bit unsigned integer, which is then used (module total tickets) to select the winning ticket, yielding a much broader entropy space

## **2. Potential Lack of Refund on Failed Competition [Acknowledge]**

**Location:** end\_competition

**Issue:** If competition.jackpot\_balance < competition.min\_jackpot\_balance, the contract sets status = Failed and unconditionally transfers the entire lamport balance to the protocol\_fee\_account. Participants who bought tickets do not get refunded.

**Recommendation:** Consider distributing lamports back to ticket holders if the competition fails, or clearly document that “failed competitions accumulate in the protocol fee.”



---

### **1. Insufficient Validation in distribute influencer fees [Solved ✓]**

**Location:** distribute\_influencer\_fees

**Issue:** While it does check that `ctx.remaining_accounts.len() == influencer_list.influencers.len()`, it does not confirm each influencer account is a system-owned account. If an influencer address is actually a program account with special logic, lamport transfers can have unexpected results.

**Recommendation:** Add a check ensuring each influencer address is a standard system account (if that is the intended design) or at least clarify in documentation.

**Fix:** The code now verifies that each influencer account provided in the remaining accounts:

1. Matches the influencer list
2. Is owned by the system program (using `system_program::ID`)



---

No informational severity issues were found.

# Assessment Results

## Score Results

Review	Score
Global Score	90/100
Assure KYC	<a href="https://projects.assuredefi.com/project/blind-faith">https://projects.assuredefi.com/project/blind-faith</a>
Audit Score	85/100

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project must pass and provide all the data needed for the assessment. Our Passing Score has been changed to 84 Points for a higher standard, if a project does not attain 85% is an automatic failure. Read our notes and final assessment below. The Global Score is a combination of the evaluations obtained between having or not having KYC and the type of contract audited together with its manual audit.

## Audit PASS

This audit report presents an in-depth analysis of the sol contract functions, emphasizing its immutability, renounced ownership, and financial controls.

The project did not meet the audit criteria due to the presence of high and medium severity issues.

All high issues have been resolved, and the project has successfully met the audit criteria.

# Disclaimer

Assure Defi has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocating for the Project, and users relying on this report should not consider this as having any merit for financial advice in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided 'as is, and Assure Defi is under no covenant to audit completeness, accuracy, or solidity of the contracts. In no event will Assure Defi or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report.

The assessment services provided by Assure Defi are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The services may access, and depend upon, multiple layers of third parties.