Assure DeFi®

THE VERIFICATION **GOLD STANDARD**

Security Assessment

BlindFaith

Date: 20/03/2025

Audit Status: FAIL

Audit Edition: Solana

ASSURE DEFI®

THE VERIFICATION GOLD STANDARD

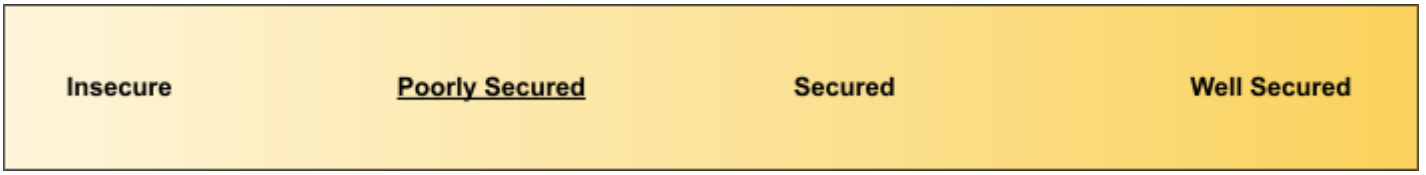
Risk Analysis

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Low	Low-level vulnerabilities are primarily associated with outdated or unused code snippets that generally do not significantly impact execution, sometimes they can be ignored.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 **Poorly 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	https://github.com/saosci/blindfaith_program Commit: d8e645c176cf44dea7a323b70903b8e4ed57dccd Fixed version: https://github.com/saosci/blindfaith_program/commit/bb53a5b624063b63c42b3e1e8951be6bc0dcd59e V2 Version: 5cb78293b8098df002623f1ca2f81a49b7e16ec8
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

3. Protocol fee sink not constrained

Location: PurchaseTickets, PurchaseGlobalJackpotTickets, TopOffGlobalJackpot

Issue: In purchase_tickets, purchase_global_jackpot_tickets, and top_off_global_jackpot, the protocol_fee_account is only #[account(mut)] (no seeds/owner constraint).

A buyer can pass any account as protocol_fee_account and the program will transfer the protocol fee to the attacker's account. This lets users siphon protocol revenue on every purchase/top-off.

Recommendation: Add: #[account(mut, seeds = [b"protocol_fee_account"], bump)] pub protocol_fee_account: Account<'info, ProtocolFeeAccount>. (Or at minimum: seeds+owner checks on AccountInfo.)

4. Global Jackpot prize uses field never updated

Location: purchase_tickets, purchase_global_jackpot_tickets, top_off_global_jackpot (no updates to total_balance), claim_global_jackpot_prize (uses total_balance)

Issue: global_jackpot.total_balance is never increased when users buy/top-off tickets; prize payment uses this field.

Winner of global jackpot will receive 0 lamports even though lamports were transferred to the jackpot account. The lamports remain stuck in the PDA.

Recommendation: On each deposit to the jackpot PDA, also do global_jackpot.total_balance = global_jackpot.total_balance.checked_add(amount)? and on claim, transfer actual balance (or keep the tracked field in sync).

MEDIUM

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.”

3. Ticket purchases after end

Location: purchase_tickets, end_competition

Issue: purchase_tickets only checks clock.unix_timestamp < competition.end_time, it doesn’t check competition.status == Active.

Recommendation: Add require!(competition.status == CompetitionStatus::Active, …). In end_competition, also require Clock::get()?.unix_timestamp >= competition.end_time.

4. Slice without length check

Location: select_winner, global_jackpot_select_winner

Issue: revealed_random_value[..8] assumes ≥ 8 bytes.

Recommendation: Check require!(revealed_random_value.len() >= 8,..); before slicing.

5. Randomness account binding

Location: select_winner, global_jackpot_select_winner

Issue: You store randomness_account (Pubkey) but don’t later assert that both randomness_account and randomness_account_data.key() equal the stored value during selection.

Recommendation: Add require_keys_eq!(*ctx.accounts.randomness_account.key, competition.randomness_account, …); and same check for randomness_account_data.key().

LOW

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)

INFORMATIONAL

1. Devnet program ID

Issue: ON_DEMAND_PROGRAM_ID defaults to Devnet PID.

Recommendation: Switch to mainnet PID for production, ideally via config gate.

Assessment Results

Score Results

Review	Score
Global Score	**60/100**
Assure KYC	https://projects.assuredefi.com/project/blind-faith
Audit Score	60/100

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 FAIL

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.

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.