VITAL BLOCK

Security Assessment

THE SOL TRAIN

Vital Block Verified On Feb 10th, 2024

@Vital-Block

@VB_Audit

info@vitalblock.org

www.vitalblock.org

PREPARED FOR:

THE SOL TRAIN

TABLE OF CONTENTS

INTRODUCTION

Auditing Firm	**VITAL BLOCK SECURITY**
Client Firm	**THE SOL TRAIN**
Methodology	**Automated Analysis, Manual Code Review.**
Language	**Anchor**
Cluster	**DEVNET**
Contract	Anchor.toml Cargo.toml
Source Code Light	**Verified**
Centralization	**Active ownership**
Blockchain	**SOLANA CHIAN**
Telegram Group	**https://t.me/thesoltrain**
Twitter	**https://twitter.com/TheSolTrain**
Doc	**https://docs.thesoltrain.xyz/**
Prelim Report Date	**FEBRUARY 5th 2024**
Final Report Date	**FEBRUARY 10th 2024**

ℹ️ **Verify the authenticity of this report on our GitHub Repo: https://www.github.com/vital-block**

Document Properties

Client	The Sol Train
Title	Smart Contract Audit Report
Target	The Sol Train
Audit Version	1.0
Author	Akhmetshin Marat
Auditors	Akhmetshin Marat, James BK, Benny Matin
Reviewed by	Dima Meru
Approved by	Prince Mitchell
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	February 10th, 2024	James BK	Final Released
1.0-AP	February 10th , 2024	Benny Matin	Release Candidate

Contact

For more information about this document and its contents, please contact Vital Block Security Inc.

Name	Akhmetshin Marat
Phone	+44 7944 248057
Email	info@vitalblock.org

In the following, we show the specific pull request and the commit hash value used in this audit.

- Anchor.toml (DD459R0)
- Cargo.toml (PH73278)

About Vital Block Security

Vital Block Security provides professional, thorough, fast, and easy-to-understand smart contract security audit. We do in-depth and penetrative static, manual, automated, and intelligent analysis of the smart contract. Some of our automated scans include tools like ConsenSys MythX, Mythril, Slither, Surya. We can audit custom smart contracts, DApps, NFTs, etc (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/vital_block), Twitter (http://twitter.com/Vb_Audit), or Email (info@vitalblock.org).

Table 1.2: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Methodology (1)

To standardize the evaluation, we define the following terminology based on the OWASP Risk Rating Methodology [4]:

- 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 demonstrates the overall criticality of the risk.

SCOPE OF WORK

Vital Block was consulted by THE SOL TRAIN to conduct the smart contract audit of its. Sol source code. The audit scope of work is strictly limited to mentioned .toml file only:

- **Soltrain.toml**

External contracts and/or interfaces dependencies are not checked due to being out of scope.

Verify audited contract's contract address and deployed link below:

Audited Contract
Anchor.toml
Cargo.toml

AUDIT METHODOLOGY

Smart contract audits are conducted using a set of standards and procedures. Mutual collaboration is essential to performing an effective smart contract audit. Here's a brief overview of Vital Block auditing process and methodology:

CONNECT

- **The onboarding team gathers source codes, and specifications to make sure we understand the size, and scope of the smart contract audit.**

AUDIT

- **Automated analysis is performed to identify common contract vulnerabilities. We may use the following third-party frameworks and dependencies to perform the automated analysis:**
  - **Remix IDE Developer Tool**
  - **Open Zeppelin Code Analyzer**
  - **SWC Vulnerabilities Registry**
  - **DEX Dependencies, e.g., Pancakeswap, Uniswap**
- **Simulations are performed to identify centralized exploits causing contract and/or trade locks.**
- **A manual line-by-line analysis is performed to identify contract issues and centralized privileges. We may inspect below mentioned common contract vulnerabilities, and centralized exploits:**

Centralized Exploits	○ **Token Supply Manipulation** ○ **Access Control and Authorization** ○ **Assets Manipulation** ○ **Ownership Control** ○ **Liquidity Access** ○ **Stop and Pause Trading** ○ **Ownable Library Verification**

Common Contract Vulnerabilities	○ Integer Overflow ○ Lack of Arbitrary limits ○ Incorrect Inheritance Order ○ Typographical Errors ○ Requirement Violation ○ Gas Optimization ○ Coding Style Violations ○ Re-entrancy ○ Third-Party Dependencies ○ Potential Sandwich Attacks ○ Irrelevant Codes ○ Divide before multiply ○ Conformance to Solidity Naming Guides ○ Compiler Specific Warnings ○ Language Specific Warnings

REPORT

- The auditing team provides a preliminary report specifying all the checks which have been performed and the findings thereof.
- The client's development team reviews the report and makes amendments to the codes.
- The auditing team provides the final comprehensive report with open and unresolved issues.

PUBLISH

- The client may use the audit report internally or disclose it publicly.

It is important to note that there is no pass or fail in the audit, it is recommended to view the audit as an unbiased assessment of the safety of solidity codes.

Table 1.0 The Full Audit Checklist

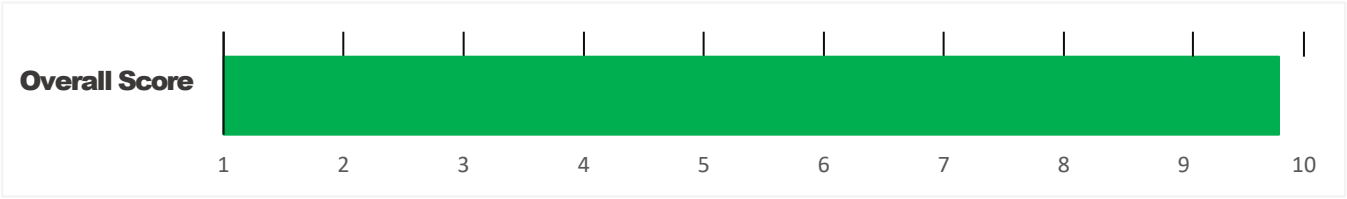
Category	Checklist Items
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	Toml Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

EXECUTIVE SUMMARY

Vital Block Security has performed the automated and manual analysis of the THE SOL TRAIN toml code. The code was reviewed for common contract vulnerabilities and centralized exploits. Here's a quick audit summary:

Status	Critical ! 🔴	Major " 🟠	Medium # 🟡	Minor $ 🟢	Unknown % 🟤
Open	0	0	0	2	0
Acknowledged	0	0	0	0	1
Resolved	0	0	2	0	2
Noteworty onlyOwner Privileges	Set Taxes and Ratios, Airdrop, Set Protection Settings, Set Reward Properties, Set Reflector Settings, Set Swap Settings, Set Pair and Router				

THE SOL TRAIN Smart contract has achieved the following score: 98.0



Please note that smart contracts deployed on blockchains aren't resistant to exploits, vulnerabilities and/or hacks. Blockchain and cryptography assets utilize new and emerging technologies. These technologies present a high level of ongoing risks. For a detailed understanding of risk severity, source code vulnerability, and audit limitations, kindly review the audit report thoroughly.

Please note that centralization privileges regardless of their inherited risk status - constitute an elevated impact on smart contract safety and security.

CENTRALIZED PRIVILEGES

Centralization risk is the most common cause of cryptography asset loss. When a smart contract has a privileged role, the risk related to centralization is elevated.

There are some well-intended reasons have privileged roles, such as:

- **Privileged roles can be granted the power to** pause() **the contract in case of an external attack.**
- **Privileged roles can use functions like,** include(), **and** exclude() **to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale and to list on an exchange.**

Authorizing privileged roles to externally-owned-account (EOA) is dangerous. Lately, centralization-related losses are increasing in frequency and magnitude.

- **The client can lower centralization-related risks by implementing below mentioned practices:**
- **Privileged role's private key must be carefully secured to avoid any potential hack.**
- **Privileged role should be shared by multi-signature (multi-sig) wallets.**
- **Authorized privilege can be locked in a contract, user voting, or community DAO can be introduced to unlock the privilege.**
- **Renouncing the contract ownership, and privileged roles.**
- **Remove functions with elevated centralization risk.**

ℹ️ **Understand the project's initial asset distribution. Assets in the liquidity pair should be locked. Assets outside the liquidity pair should be locked with a release schedule.**

RISK CATEGORIES

Smart contracts are generally designed to hold, approve, and transfer tokens. This makes them very tempting attack targets. A successful external attack may allow the external attacker to directly exploit. A successful centralization-related exploit may allow the privileged role to directly exploit. All risks which are identified in the audit report are categorized here for the reader to review:

Risk Type	Definition
Critical ! 🔴	These risks could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
Major " 🟠	These risks are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to high-risk severity.
Medium # 🟡	These risks should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution. Low-risk re-entrancy-related vulnerabilities should be fixed to deter exploits.
Minor $ 🟢	These risks do not pose a considerable risk to the contract or those who interact with it. They are code-style violations and deviations from standard practices. They should be highlighted and fixed nonetheless.
Unknown % 🟤	These risks pose uncertain severity to the contract or those who interact with it. They should be fixed immediately to mitigate the risk uncertainty.

All statuses which are identified in the audit report are categorized here for the reader to review:

Status Type	Definition
Open	Risks are open.
Acknowledged	Risks are acknowledged, but not fixed.
Resolved	Risks are acknowledged and fixed.

AUDIT SCOPE | THE SOL TRAIN

ID	Repo	Comment	File	SHM321 Checksum
YBY	contracts/solrain/src	Cc512474	Anchor.toml	6788099YIRHVSK853PKTGYHHH67843OKJFGYYY766I09
YBI	contracts/solrain/src	cC512474	Anchor.toml	347520JHDB7549H22HRTFRRE45563DESPDHBVHD655
YBW	contracts/solrain/src	cC512474	Anchor.toml	1988Y73HUGFDINN353840OPUUYTEHHGDTFF9NNDU
YBG	contracts/solrain/src	cC512474	Anchor.toml	4438648TEOHBF6378309EHROECNEPOEJDNETE8EYEU3
YBL	contracts/solrain/src	cC512474	Anchor.toml	66390028765RVNKDBYFTGW553T2KOEREDW78900O7
YBA	contracts/solrain/src	cC512474	Anchor.toml	09825539BDYG543DVNKOMIKEBYRRRE4367DGVR5EUY
YBJ	contracts/solrain/src	cC512474	Anchor.toml	8654RJVT3DWI865YK2643YTRFVDJBOBET8386YF3683G
YBE	contracts/soltrain/cargo	cC512474	Cargo.toml	7763888636TGYGFFTFHBTGDCVSNDO788U59
YBP	Contracts/soltrain/cargo	cC512474	Cargo.toml	88530486494YRHFTEICBGEIEGWTWYWUHEJEHEIE33U3
YBM	contracts/soltrain/cargo	cC512474	Cargo.toml	1209873KHJLKJNFJHGE98763990029774BCUHHDUU239
YBV	contracts/soltrain/cargo	cC512474	Cargo.toml	23456UGFYUHE98756EFHJHE7654ESDFGHGERTYUJ3897
YBQ	contracts/soltrain/cargo	cC512474	Cargo.toml	37889UHBIONEO7TYRDFGVBN5678939IJWSFVDYUHDCI
YBS	contracts/soltrain/cargo	cC512474	Cargo.toml	678903098TFHJKFCPOIUGFGHJKE9865ERGBEIVBHE8767
YBR	contracts/soltrain/Xargo	cC512474	Xargo.toml	98765SDFGBNFCOI56789UIYHGGHEJDIUYTRDCVBN3459
YCD	contracts/soltrain/Xargo	cC512474	Xargo.toml	3348y9808hgtrusvnmu43100ejfojgfnut8496230hb574he
YHU	contracts/soltrain/Xargo	cC512474	Xargo.toml	9864byf5f379eig28ffre64085jv1613251guhkdmue87
YGG	contracts/soltrain/Xargo	cC512474	Xargo.toml	7ej2d8jg765tjfiowg538ij74dwftyv6478ij3gs820
YTR	contracts/soltrain/Xargo	cC512474	Xargo.toml	864fr46de438hdguw903rfdcb246dbuhb2917enk

OPTIMIZATIONS | THE SOL TRAIN

ID	Title	Category	Status
STV	Logarithm Refinement Optimization	Gas Optimization	Acknowledged
SOP	Checks Can Be Performed Earlier	Gas Optimization	Acknowledged
SDP	Unnecessary Use Of SafeMath	Gas Optimization	Acknowledged
SWY	Struct Optimization	Gas Optimization	Acknowledged
SGT	Unused State Variable	Gas Optimization	Acknowledged

General Detectors

Public Functions Should be Declared External

Some functions in this contract should be declared as external in order to save gas

Attention Required

Missing Zero Address Validation

Some functions in this contract may not appropriately check for zero addresses being used.

Attention Required

Numeric Notation Best Practices

The numeric notation used in this contract is unconventional, possibly worsening the reading/debugging experience

Attention Required

- No compiler version inconsistencies found
- No unchecked call responses found
- No vulnerable self-destruct functions found
- No assertion vulnerabilities found
- No old Anchor code found
- No external delegated calls found
- No external call dependency found
- No vulnerable authentication calls found
- No invalid character typos found
- No RTL characters found
- No dead code found
- No risky data allocation found
- No uninitialized state variables found
- No uninitialized storage variables found
- No vulnerable initialization functions found
- No risky data handling found
- No number accuracy bug found
- No out-of-range number vulnerability found
- No map data deletion vulnerabilities found
- No tautologies or contradictions found
- No faulty true/false values found
- No innacurate divisions found
- No redundant constructor calls found
- No vulnerable transfers found
- No vulnerable return values found
- No uninitialized local variables found
- No default function responses found
- No missing arithmetic events found
- No missing access control events found
- No redundant true/false comparisons found
- No state variables vulnerable through function calls found
- No buggy low-level calls found
- No expensive loops found
- No bad numeric notation practices found
- No missing constant declarations found
- No missing external function declarations found
- No vulnerable payable functions found
- No vulnerable message values found

Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 0 high-severity vulnerabilities, 1 medium-severity vulnerabilities, 2 low-severity vulnerabilities, and 1 informational recommendations.

Table 2.1: THE SOL TRAIN Audit Findings

ID	Severity	Title	Category	Status
TST-001	Informational	In Suggested Constant/Immutable Usages For Gas Efficiency	Coding Practice	Informational
TST-002	Medium	Proper Emode Category Use in Pool::Ticket ()	Business Logic	Fixed
TST-003	Low	Possible Underflow Avoidance in Ticket Logic And UserConfiguration	Coding Practices	Confirmed
TST-004	Low	Possible Underflow Avoidance in Ticket Logic And UserConfiguration	Coding Practice	Fixed

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to page 10 for details.

TST-01 Key Findings

Category	Severity	Location	Status
Coding Practices	Low	Multiple Contracts	Informational

Description

In **Suggested Constant/Immutable Usages For Gas Efficiency**

```
pub fn claim_referral(ctx: Context<ClaimReferral>) -> Result<()> {
let userdata = &mut ctx.accounts.userdata;
let dashboard = &mut ctx.accounts.dashboard;
let user = ctx.accounts.user.key.clone();
let user_account = &mut ctx.accounts.user;     // get total balance
let amount = userdata.to_account_info().lamports;
let referral_balance = **amount.borrow_mut();
```

Description

An immutable state variable can only be assigned during contract creation, but will remain constant throughout the life-time of a deployed contract. The main benefit of declaring a state as immutable is that reading the state is significantly cheaper than reading from regular storage, since it is not stored in storage anymore. Instead, an immutable state will be directly inserted into the runtime code.

This feature is introduced based on the observation that the reading and writing of storage-based contract states are gas-expensive. Therefore, it is always preferred if we can reduce, if not eliminate, storage reading and writing as much as possible. Those state variables that are written only once are candidates of immutable states under the condition that each fits the pattern, i

Recommendation

Revisit the state variable definition and make extensive use of constant/ immutable states.

TST-02 Key Findings

Category	Severity ●	Target	Status
Coding Practices	Medium	Pool	Fixed

Description

Proper **Emode Category Use in Pool::Ticket ()**

```
pub fn buy_tickets(ctx: Context<BuyTickets>, amount: u8, referral: Pubkey) -> Result<()> {
    let userdata = &mut ctx.accounts.userdata;
    let dashboard = &mut ctx.accounts.dashboard;
    let user = ctx.accounts.user.key.clone();
    let user_account = &mut ctx.accounts.user;
    let block_timestamp = ctx.accounts.clock.unix_timestamp;

    let total_fee: u64 = u64::from(amount) * dashboard.ticket_price;

    let ix = anchor_lang::solana_program::system_instruction::transfer(
      &user.key(),
      &dashboard.key(),
      total_fee,
    );
      )
```

Description

The SOL TRAIN has a nice feature `credit delegation`, which allows a credit delegator to delegate the credit of their account's position to a Ticket. This feature requires proper accounting of delegation allowance and actual expenditure. While examining its implementation, we notice a key function `Buy Ticket ()` does not properly follow the `credit delegation` logic.

To elaborate, we show Above this `Ticket()` function. This is a core Credit function and is used to funds from the Trasfer protocol. It comes to our attention that the encapsulated `Buy Ticket. (ctx: Context<BuyTickets>,`

Recommendation

Ensure the `credit delegation` feature is consistently honored in all aspects of the Ticket pool.

YDL-07 Key Findings

Category	Severity	Target	Status
Coding Practices	low	Soltrain/src/lib	Confirmed

Description

Possible **Underflow Avoidance in Ticket Logic And** UserConfiguration

```
pub fn initialize_dashboard(ctx: Context<InitializeDashboard>) -> Result<()> {
    let dashboard = &mut ctx.accounts.dashboard;
    dashboard.owner = ctx.accounts.authority.key.clone();
    dashboard.ticket_price = 1_000_000_000;
    dashboard.departure_time = 18 * 3600;
    dashboard.half_rewards_count = 50;
    dashboard.number_of_trains = 1;
    dashboard.max_wallet_size = 20;
    dashboard.max_train_size = 100;
    dashboard.is_initialized = true;
    dashboard.departure_times = Vec::new();
    dashboard.available_tickets = (1..=100).collect();
    Ok(())
  }
```

Description

The SOL TRAIN Plartform makes good use of a number of reference contracts, such as ERC20, SafeERC20, And SafeMath to facilitate its code implementation and organization. For example, the Pool smart contract has so far imported at least Two reference contracts. However, we observe the inclusion of certain unused code or the presence of unnecessary redundancies that can be safely removed.

Recommendation

Revise the above calculation to avoid the unnecessary overflows and under- flows.

YDL-08 Key Findings

Category	Severity ●	Target	Status
Coding Practices	low	Contract/soltrain/cargo	Fixed

Description

Possible **Underflow Avoidance in Ticket Logic And** UserConfiguration

```
// initialize new train
dashboard.departure_times.push(block_timestamp);
dashboard.available_tickets = (1..=100).collect();
dashboard.number_of_trains += 1;

// execute buys for next train
len = dashboard.available_tickets.len();
for i in (1..len).rev() {
    let j = rng.gen_range(0, i + 1);
    dashboard.available_tickets.swap(i, j)
```

Description

For gas efficiency, the SOL TRAIN is engineered with the reserve cache mechanism, which necessi-tates the common steps to be followed when operating with the reserve data in different scenarios, including the cache generation, update, and eventual persistence. However, our analysis shows cer-tain inconsistency in the reserve cache usages and the inconsistency needs to be resolved to avoid confusions and errors.

Recommendation

Revise the above functions to following a consistent approach to use the reserve cache mechanism.

Vulnerability Scan

REENTRANCY

No reentrancy risk found

Severity	Major
Confidence Parameter	Certain

Vulnerability Description

Mintable: More amount of the SOL TRAIN TICKET can **NOT** be minted by a private wallet or contract. (This is Essentially normal for most contracts)

Scanning Line:

```
let len = dashboard.available_tickets.len();
      for i in (1..len).rev() {
          let j = rng.gen_range(0, i + 1);
          dashboard.available_tickets.swap(i, j);
      }

      for _ in 0..amount {
        userdata.tickets.push( Ticket {
          train_number: dashboard.number_of_trains,
          seat: dashboard.available_tickets.pop().unwrap(),
        })
      }

      msg!("available_tickets: {:?}", dashboard.available_tickets);
      msg!("new user data: {:?}", userdata.tickets.clone());
    }
    msg!("tickets_length: {:?}", tickets_left);
```

PROJECT BASIC KNOWLEDGE

Buy a ticket. Drive the train. Earn 2 $Sol for your efforts. All aboard! The Sol Train is departing from the station!

•**All Aboard!** Buy a ticket for 1 $Sol to secure your position on the 100 seat train. When you reach the front of the train, you'll receive 2 $Sol for driving the train. Daily departures mean you'll be in the driver's seat within 100 days at most!

How It Works:
•**Trains:** Each train has 100 seats, including one driver
•**Driver:** As the driver, it's time to collect your initial 1 $Sol investment along with an additional 1 $Sol as a thank you for taking control of the train!
•**Seats:** All seats are randomly assigned. Your seat position determines when on the trip you'll be in the drivers seat.
•**Tickets:** Just 1 $Sol! You can buy up to 20 tickets per wallet.

•3.**Missed the train?**: Don't worry! A new train departs when the current one is full. Fair play for everyone!

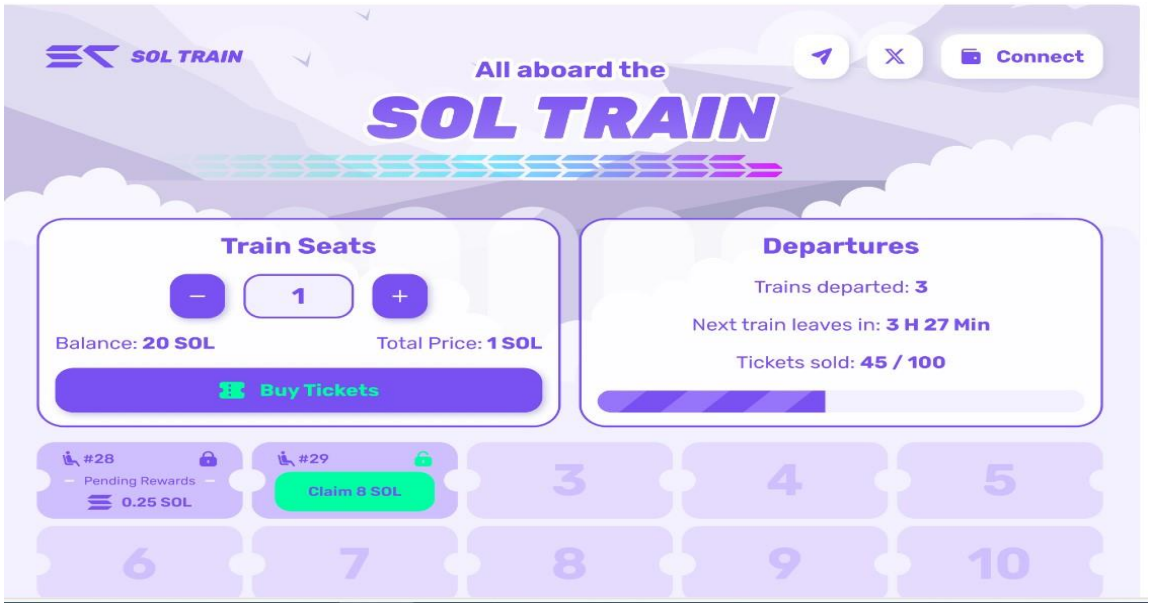
•4.**Train crash:** Coming soon □

•5.**Referrals**: With each ticket sold via your referral link, you'll get an instant 0.1 $Sol bonus that you can claim

PROJECT NAME: The Sol Train
Ticker: TST

Chain/Standard: SOLANA NETWORK

The THE SOL TRAIN Platform Is Launched On Solana Chain



ISSUES CHECKING STATUS

	Issue Description	Checking Status
1.	Compiler errors.	PASSED
2.	Race Conditions and reentrancy. Cross-Function Race Conditions.	PASSED
3.	Possible Delay In Data Delivery.	PASSED
4.	Oracle calls.	PASSED
5.	Front Running.	PASSED
6.	Toml Dependency.	PASSED
7.	Integer Overflow And Underflow.	PASSED
8.	DoS with Revert.	PASSED
9.	Dos With Block Gas Limit.	PASSED
10.	Methods execution permissions.	PASSED
11.	Economy Model of the contract.	PASSED
12.	The Impact Of Exchange Rate On the solidity Logic.	PASSED
13.	Private use data leaks.	PASSED
14.	Malicious Event log.	PASSED
15.	Scoping and Declarations.	PASSED
16.	Uninitialized storage pointers.	PASSED
17.	Arithmetic accuracy.	PASSED
18.	Design Logic.	PASSED
19.	Cross-Function race Conditions	PASSED
20.	Save Upon Rust contract Implementation and Usage.	PASSED
21.	Fallback Function Security	PASSED

AUDIT RESULT

PASSED

RECOMMENDATION

Deployer and/or contract owner private keys are secured carefully.

Please refer to PAGE-09 CENTRALIZED PRIVILEGES for a detailed understanding.

ALLEVIATION

The Sol Train project team understands the centralization risk. Some functions are provided privileged access to ensure a good runtime behavior in the project

References

1 MITRE. CWE-1041: Use of Redundant Code. https://cwe.mitre.org/data/definitions/1041.html.

2 MITRE. CWE-1099: Inconsistent Naming Conventions for Identifiers. https://cwe.mitre.org/data/definitions/1099.html.

3 MITRE. CWE-561: Dead Code. https://cwe.mitre.org/data/definitions/561.html.

4 MITRE. CWE-563: Assignment to Variable without Use. https://cwe.mitre.org/data/definitions/563.html.

5 MITRE. CWE-663: Use of a Non-reentrant Function in a Concurrent Context. https://cwe.mitre.org/data/definitions/663.html.

6 MITRE. CWE-837: Improper Enforcement of a Single, Unique Action. https://cwe.mitre.org/data/definitions/837.html.

7 MITRE. CWE-841: Improper Enforcement of Behavioral Workflow. https://cwe.mitre.org/data/definitions/841.html.

8 MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

9 MITRE. CWE CATEGORY: Business Logic Errors. https://cwe.mitre.org/data/definitions/840.html.

10 MITRE. CWE CATEGORY: Concurrency. https://cwe.mitre.org/data/definitions/557.html.

11 MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

12 OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

Identifier	Definition	Severity
COD-10	Third Party Dependencies	Minor

Smart contract is interacting with third party protocols e.g., Pancakeswap router, cashier contract, protections contract. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised, and exploited. Moreover, upgrades in third parties can create severe impacts, e.g., increased transactional fees, deprecation of previous routers, etc.

RECOMMENDATION

Inspect and validate third party dependencies regularly, and mitigate severe impacts whenever necessary.

DISCLAIMERS

Vital Block Security provides the easy-to-understand audit of Solidity, Move and Raw source codes (commonly known as smart contracts).

The smart contract for this particular audit was analyzed for common contract vulnerabilities, and centralization exploits. This audit report makes no statements or warranties on the security of the code. This audit report does not provide any warranty or guarantee regarding the absolute bug-free nature of the smart contract analyzed, nor do they provide any indication of the client's business, business model or legal compliance. This audit report does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. Cryptographic tokens are emergent technologies, they carry high levels of technical risks and uncertainty. You agree that your access and/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. This audit report could include false positives, false negatives, and other unpredictable results.

CONFIDENTIALITY

This report is subject to the terms and conditions (including without limitations, description of services, confidentiality, disclaimer and limitation of liability) outlined in the scope of the audit provided to the client. This report should not be transmitted, disclosed, referred to, or relied upon by any individual for any purpose without InterFi Network's prior written consent.

NO FINANCIAL ADVICE

This audit report does not indicate the endorsement of any particular project or team, nor guarantees its security. No third party should rely on the reports in any way, including to make any decisions to buy or sell a product, service or any other asset. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. This audit report should not be used in any way

to make decisions around investment or involvement. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort.

FOR AVOIDANCE OF DOUBT, SERVICES, INCLUDING ANY ASSOCIATED AUDIT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

TECHNICAL DISCLAIMER

ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, VITAL BLOCK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO SERVICES, AUDIT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, VITAL BLOCK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM THE COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

WITHOUT LIMITING THE FOREGOING, VITAL BLOCK MAKES NO WARRANTY OF ANY KIND THAT ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET THE CLIENT'S OR ANY OTHER INDIVIDUAL'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE.

TIMELINESS OF CONTENT

The content contained in this audit report is subject to change without any prior notice. Vital Block does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following the publication.

LINKS TO OTHER WEBSITES

This audit report provides, through hypertext or other computer links, access to websites and social accounts operated by individuals other than Vital Block. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such websites and social accounts owners. You agree that Vital block Security is not responsible for the content or operation of such websites and social accounts and that Vital Block shall have no liability to you or any other person or entity for the use of third-party websites and social accounts. You are solely responsible for determining the extent to which you may use any content at any other websites and social accounts to which you link from the report.

ABOUT VITAL BLOCK

Vital Block provides intelligent blockchain Security Solutions. We provide solidity and Raw Code Review, testing, and auditing services. We have Partnered with 15+ Crypto Launchpads, audited 450+ smart contracts, and analyzed 350,000+ code lines. We have worked on major public blockchains e.g., Ethereum, Binance, Cronos, Doge, Polygon, Avalanche, Metis, Fantom, Bitcoin Cash, Aptos, Oasis, etc.

Vital Block is Dedicated to Making Defi & Web3 A Safer Place. We are Powered by Security engineers, developers, UI experts, and blockchain enthusiasts. Our team currently consists of 5 core members, and 4+ casual contributors.

Website: https://Vitalblock.org

Email: info@vitalblock.org

GitHub: https://github.com/vital-block

Telegram (Engineering): https://t.me/vital_block

Telegram (Onboarding): https://t.me/vitalblock_cmo

vital-block

info@vitalblock.org

www.Vitalblock.org

Vital Block Dedicated to securing Public and Private Blockchain Ecosystem