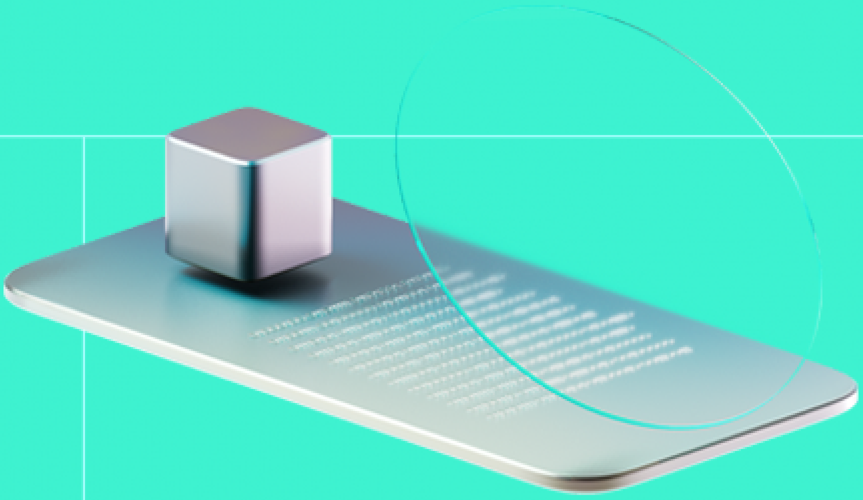
Smart Contract Code Review And Security Analysis Report



Customer: Shuttle Labs

Date: 27/01/2025

We express our gratitude to the Shuttle Labs team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

The Shuttle Labs is a groundbreaking solution, synthesizing advanced cross-chain liquidity management and order execution methodologies into a seamless, high-performance cross-chain asset transfer system.

Document

Name	Smart Contract Code Review and Security Analysis Report for Shuttle Labs
Audited By	Przemyslaw Swiatowiec
Approved By	Grzegorz Trawinski
Website	bridgesmarter.com
Changelog	16/01/2025 - Preliminary Report, 27/01/2025 - Final Report
Platform	Solana
Language	Rust
Tags	Liquidity, Orders, Bridge
Methodology	https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/Shuttle-Labs/genius-contracts-solana
Commit	1ca628357d27f652d6e16e00cda53e36697b1a07

Audit Summary

The system users should acknowledge all the risks summed up in the risks section of the report

3	3	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	1
High	0
Medium	1
Low	1

Vulnerability	Severity
F-2025-8250 - Fee Amount Not Validated, Allowing Users to Create Orders with Insufficient Fee	Critical
F-2025-8249 - Missing Minimum Order Amount Validation Can Lead to System Inefficiency and Off-Chain DoS Risks	Medium
F-2025-8187 - Lack of Order Cancellation Mechanism Forces Protocol to Cover Processing Costs for Mismanaged Fees	Low

Documentation quality

- Functional requirements are provided.
- Technical description is provided.

Code quality

- The development environment is configured.

Test coverage

Several tests are provided including both success and negative test cases. Due to the limitation of tools in the Solana ecosystem, the test coverage could not be calculated.

Table of Contents

System Overview

The Shittle Labs is a cross-chain asset transfer and liquidity management system designed to facilitate seamless token swaps and asset movement between multiple blockchain networks. Users can deposit tokens into a vault to create orders for cross-chain execution. Orchestrators, operated by the protocol, are responsible for processing these orders and ensuring the accurate transfer of assets to the target chain.

Key components of the system include:

1. **Global State**: Maintains critical configuration and operational parameters, such as admin keys, fee settings, and system status (e.g., frozen or active).
2. **Orchestrators**: Specialized accounts that handle order execution and liquidity management on the protocol's behalf. Orchestrators process orders, claim fees, and manage bridge liquidity.
3. **Vault**: The central repository for user deposits and liquidity management. It securely holds assets until orders are processed or withdrawn.
4. **Fee Management**: Enforces a minimum fee structure for cross-chain transactions, ensuring orchestrators are compensated for their work. This includes dynamic fee adjustments to adapt to varying gas costs across chains.
5. **Order Lifecycle**: Supports order creation and order fill.

Privileged roles

Admin

- Responsible for initializing the protocol, managing orchestrators, updating global parameters, and nominating new admins.

Orchestrators

- Execute cross-chain orders, claim processing fees, and manage bridge liquidity.
- Operated by the protocol and must remain authorized to function.

Freeze/Thaw Authorities

- Control the operational state of the protocol by freezing or thawing the global state to manage sensitive operations.

Users

- Create and manage orders by depositing tokens into the vault for cross-chain execution.

Potential Risks

- **Off-Chain Orchestrator Vulnerability -** Many processes are handled off-chain, making orchestrators a potential weak point. Malicious actors could target orchestrators to disrupt protocol operations.
- **Centralization Risk -** Since orchestrators are operated by a single company, there is a risk of centralization, which may reduce the protocol's trustworthiness and resilience to failure or malicious behavior.
- **Lack of Documentation on Key Processes -** Processes such as swapping all tokens back and forth to USDC are not well-documented. This lack of clarity can lead to misunderstandings, improper implementation, or unintentional errors by orchestrators and developers.

Findings.

Vulnerability Details

F-2025-8250 - Fee Amount Not Validated, Allowing Users to Create Orders with Insufficient Fee - Critical

Description:

The protocol does not validate whether the `fee` specified by the user when creating an order is less than the `amount` of the order. This allows users to specify a `fee` greater than the actual order amount, resulting in the protocol accumulating a fee balance that does not exist. Consequently, when orchestrators attempt to withdraw accumulated fees, they may encounter a situation where the total fee balance exceeds the actual funds available, causing withdrawals to fail and halting protocol operations.

```
impl CreateOrder<'_> {
    pub fn process_instruction(
        ctx: Context<Self>,
        ...
    ) -> Result<()> {
        ...
        let min_fee = ctx.accounts.target_chain_min_fee.min_fee;
        if min_fee > fee {
            return err!(GeniusError::InsufficientFees);
        }

        ...

        ctx.accounts.asset.unclaimed_fees += fee;
        ctx.accounts.asset.total_fee_collected += fee;

        // Transfer USDC from orchestrator to vault
        token_transfer_user(
            ctx.accounts.ata_trader.to_account_info().clone(),
            ctx.accounts.trader.to_account_info().clone(),
            ctx.accounts.ata_vault.to_account_info().clone(),
            ctx.accounts.token_program.to_account_info().clone(),
            amount,
        )?;

       ...

        Ok(())
    }
}
```

Assets:

- programs/genius/src/instructions/create_order.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]

Status: Fixed

Classification

Impact: 5/5

Likelihood: 5/5

Exploitability: Independent

Complexity: Simple

Severity: Critical

Recommendations

Remediation: Ensure that the `fee` specified by the user is less than or equal to the `amount` of the order and reject orders where `fee > amount`.

Resolution: The issue was fixed in the commit `dec7f285450ee16be4b674711f7b4e2199c1bb1d` by validating the amount and fee values.

Evidences

Steps to reproduce

Reproduce:

Create an Order with Insufficient Funds to Cover the Fee:

- A malicious user creates an order with a small `amount` but specifies a larger `fee`. For example:
  - `amount = 0.05 SOL`
  - `fee = 100 SOL`
- The protocol accepts the order, assuming the fee is valid, and adds `100 SOL` to the accumulated fee balance.

Repeat the Process to Inflate Fee Balances:

- The user repeats the above step multiple times, creating numerous orders with fees larger than the order amounts. Each time, the protocol incorrectly accumulates fees that do not exist in the vault.

Withdraw Fees by Orchestrators:

- Orchestrators attempt to withdraw accumulated fees. However, the vault balance is insufficient to cover the inflated fee balance.
- This results in withdrawal failures, disrupting protocol operations and potentially halting orchestrator activities.

F-2025-8249 - Missing Minimum Order Amount Validation Can Lead to System Inefficiency and Off-Chain DoS Risks - Medium

Description: The protocol does not enforce a minimum order amount when creating orders. This allows users to create a large number of orders with negligible amounts, such as near-zero token values. Such behavior can lead to significant inefficiencies and potential denial-of-service (DoS) risks for the off-chain orchestrators responsible for processing these orders.

If orchestrators must process thousands of trivial orders, their computational resources and bandwidth could be overwhelmed, leading to delays in legitimate transactions and degraded performance of the off-chain network. This lack of validation creates a vulnerability that could be exploited by malicious users to disrupt the system.

```
impl CreateOrder<'_> {
    pub fn process_instruction(
    ...
    ) -> Result<()> {
        let min_fee = ctx.accounts.target_chain_min_fee.min_fee;
        if min_fee > fee {
            return err!(GeniusError::InsufficientFees);
        }
        ctx.accounts.order.amount_in = amount;
        ctx.accounts.order.seed = seed;
        ctx.accounts.order.trader = trader_key_bytes;
        ctx.accounts.order.receiver = receiver;
        ctx.accounts.order.src_chain_id = src_chain_id;
        ctx.accounts.order.dest_chain_id = dest_chain_id;
        ctx.accounts.order.token_in = token_in;
        ctx.accounts.order.fee = fee;
        ctx.accounts.order.status = OrderStatus::Created;
        ctx.accounts.order.min_amount_out = min_amount_out;
        ctx.accounts.order.token_out = token_out;

        ctx.accounts.asset.unclaimed_fees += fee;
        ctx.accounts.asset.total_fee_collected += fee;

        // Transfer USDC from orchestrator to vault
        token_transfer_user(
            ctx.accounts.ata_trader.to_account_info().clone(),
            ctx.accounts.trader.to_account_info().clone(),
            ctx.accounts.ata_vault.to_account_info().clone(),
            ctx.accounts.token_program.to_account_info().clone(),
            amount,
        )?;
```

Assets:

- programs/genius/src/instructions/create_order.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]

Status: Fixed

Classification

Impact: 3/5

Likelihood: 4/5

Exploitability: Independent

Complexity: Simple

Severity: Medium

Recommendations

Remediation: Introduce a minimum order amount validation to ensure that only meaningful orders are created. This prevents resource wastage and mitigates the risk of orchestrator overload.

Resolution: The issue was fixed by the `F-2025-8250` - the the `min_fee` requirement protects against potential Denial of Service (commit `dec7f285450ee16be4b674711f7b4e2199c1bb1d`).

F-2025-8187 - Lack of Order Cancellation Mechanism Forces Protocol to Cover Processing Costs for Mismanaged Fees - Low

Description:

The program does not provide an order cancellation mechanism, leaving the protocol vulnerable to scenarios where funds become stuck if an order cannot be processed. Currently, orchestrators, which are run by the protocol, are required to process orders. If the protocol sets an insufficient minimum fee for processing an order, orchestrators must cover the fee difference to process the order. This can lead to financial losses for the protocol, especially in chains where gas prices vary significantly and are difficult to predict accurately.

Without an order cancellation mechanism, the protocol bears the risk of mismanaging fees, forcing it to subsidize the cost of processing orders where the minimum fee was underestimated. Over time, this could lead to unsustainable costs and reduce protocol reliability.

```
impl SetTargetChainMinFee<'_> {
    pub fn process_instruction(ctx: Context<Self>, dest_chain_id: u32, min_fee: u64) -> Result<()> {
        // shouldn't be verified against fee in cross_chain_fee_bps?
        let target_chain_min_fee = &mut ctx.accounts.target_chain_min_fee;
        target_chain_min_fee.dest_chain_id = dest_chain_id;
        target_chain_min_fee.token_in = ctx.accounts.usdc_mint.key().to_bytes();
        target_chain_min_fee.min_fee = min_fee;

        Ok(())
    }
}
```

Status: Fixed

Classification

Impact: 3/5

Likelihood: 2/5

Exploitability: Semi-Dependent

Complexity: Medium

Severity: Low

Recommendations

Remediation: Introduce an order cancellation mechanism that allows users to cancel their unprocessed orders. This would prevent the protocol from being forced to process orders at a loss in cases of mismanaged fees or volatile gas prices.

Resolution: The issue was fixed in `dec7f285450ee16be4b674711f7b4e2199c1bb1d` commit by introducing the `RevertOrder` instruction.

Observation Details

F-2025-8181 - Use Anchor Events Instead of msg! for Structured Logging - Info

Description:

The program extensively uses `msg!` across various instructions to log critical events like order management, While `msg!` is effective for simple debugging, it does not provide a structured format that can be easily indexed or consumed by off-chain applications. This reliance on unstructured logging makes it challenging for developers and indexers to extract meaningful data programmatically. The lack of structured events limits the ability to filter or query specific actions, such as tracking orders or orchestrator actions, and results in additional overhead for parsing raw logs.

Code example:

```
msg!(
    "OrderCreated: {{\
    \"seed\":\"{:?}\",\
    \"trader\":\"{:?}\",\
    \"receiver\":\"{:?}\",\
    \"token_in\":\"{:?}\",\
    \"token_out\":\"{:?}\",\
    \"amount_in\":{},\
    \"min_amount_out\":{},\
    \"src_chain_id\":{},\
    \"dest_chain_id\":{},\
    \"fee\":{},\
    \"status\":\"{}\"\
    }}",
    seed,
    trader_key_bytes,
    receiver,
    token_in,
    token_out,
    amount,
    min_amount_out,
    src_chain_id,
    dest_chain_id,
    fee,
    "Created"
);
```

Status: Accepted

Recommendations

Remediation: Replace `msg!` logs with Anchor events. Define events for key actions and emit them during instruction execution.

Resolution: The client accepted the issue.

F-2025-8182 - Missing Event Emission for Critical Actions - Info

Description: The program lacks event emission for critical actions such as state updates, role changes, and key transactions (e.g., admin nomination, orchestrator addition/removal, and updates to global parameters). Without events, off-chain applications and indexers cannot efficiently track these changes, making it harder to monitor program behavior and build reliable integrations. This absence increases the difficulty of debugging and auditing state changes, while also limiting transparency for users and developers.

Status: Fixed

Recommendations

Remediation: Introduce Anchor events for all critical actions to provide structured and traceable updates.

Resolution: The issue was fixed in the commit `dec7f285450ee16be4b674711f7b4e2199c1bb1d` by introducing event emissions in the form of `!msg` messages.

F-2025-8183 - Orchestrator Accounts Are Not Closed Upon Removal, Leading to Wasted Rent - Info

Description: The `RemoveOrchestrator` instruction deactivates the orchestrator by setting the `authorized` field to `false` but does not close the associated account. As a result, the SOL rent used to maintain the account remains locked, leading to inefficient use of resources. Over time, this can cause unnecessary on-chain clutter and wasted rent for unused accounts.

```rust
    //  Stores orchestrator info
    #[account(
        mut,
        seeds = [orchestrator.key().as_ref(), ORCHESTRATOR_SEED],
        bump
    )]
    pub orchestrator_state: Box<Account<'info, OrchestratorState>>,
}

impl RemoveOrchestrator<'_> {
    pub fn process_instruction(ctx: Context<Self>) -> Result<()> {
        let orchestrator_state = &mut ctx.accounts.orchestrator_state;

        orchestrator_state.authorized = false;

        Ok(())
    }
```

Status: Accepted

Recommendations

Remediation: Implement account closure for orchestrators upon removal. Transfer the remaining lamports from the account back to the admin or a designated recipient, and close the account to reclaim storage rent.

Resolution: The client accepted the issue.

F-2025-8186 - Incorrect Length Validation in AddGlobalStateAuthority Allows Adding One Extra Authority - Info

Description:

The `AddGlobalStateAuthority` instruction uses a `<=` comparison when validating the number of authorities in the `freeze_authority` and `thaw_authority` vectors. This allows one extra authority to be added beyond the maximum defined limit (`MAX_FREEZE_AUTHORITY_LENGTH` or `MAX_THAW_AUTHORITY_LENGTH`). Consequently, this results in fewer slots available for other types of authorities. For example, if the user adds 11 freeze authorities (1 more than the limit), only 9 thaw authorities can be added, reducing the overall capacity and flexibility of the program.

```
impl AddGlobalStateAuthority<'_> {
    pub fn process_instruction(
        ctx: Context<Self>,
        authority_address: Pubkey,
        is_freeze_authority: bool,
    ) -> Result<()> {
        let global_state_authority = &mut ctx.accounts.global_state_authority;
        // let mut global_state_authority_clone = global_state_authority.clone();
        if is_freeze_authority {
            require!(
                !global_state_authority.freeze_authority.contains(&authority_address),
                GeniusError::AuthorityAlreadyExists
            );
            require!(
                global_state_authority.freeze_authority.len() <= MAX_FREEZE_AUTHORITY_LENGTH,
                GeniusError::MaxAuthoritiesAlreadySet
            );
            global_state_authority.freeze_authority.push(authority_address);
        // global_state_authority.set_inner(global_state_authority_clone.into_inner());
        } else {
            require!(
                !global_state_authority.thaw_authority.contains(&authority_address),
                GeniusError::AuthorityAlreadyExists
            );
            require!(
                global_state_authority.thaw_authority.len() <= MAX_THAW_AUTHO
RITY_LENGTH,
                GeniusError::MaxAuthoritiesAlreadySet
            );
            global_state_authority.thaw_authority.push(authority_address);
            // global_state_authority.set_inner(global_state_authority_clone.
into_inner());
        }

        Ok(())
    }
}
```

Status: Fixed

Recommendations

Remediation: Correct the validation logic by replacing the `<=` comparison with `<`. This ensures that the program strictly enforces the maximum limits for both freeze and thaw authorities, preserving the intended balance between the two types of operators.

Resolution: The issue was fixed in the commit `dec7f285450ee16be4b674711f7b4e2199c1bb1d` by replacing `<=` comparison with `<`.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Definitions

Severities

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution.

Potential Risks

The "Potential Risks" section identifies issues that are not direct security vulnerabilities but could still affect the project’s performance, reliability, or user trust. These risks arise from design choices, architectural decisions, or operational practices that, while not immediately exploitable, may lead to problems under certain conditions. Additionally, potential risks can impact the quality of the audit itself, as they may involve external factors or components beyond the scope of the audit, leading to incomplete assessments or oversight of key areas. This section aims to provide a broader perspective on factors that could affect the project's long-term security, functionality, and the comprehensiveness of the audit findings.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details	
Repository	https://github.com/Shuttle-Labs/genius-contracts-solana
Commit	1ca628357d27f652d6e16e00cda53e36697b1a07
Whitepaper	Provided as files.
Requirements	Provided as files.
Technical Requirements	Provided as files.

Asset	Type
programs/genius/src/constant.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/error.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/accept_authority.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/add_global_state_authority.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/add_orchestrator.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/borrow.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/claim_fees.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/create_order.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/fill_order.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/freeze_thaw_global_state.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/initialize.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/mod.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/nominate_authority.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/remove_bridge_liquidity.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/remove_global_state_authority.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract

Asset	Type
programs/genius/src/instructions/remove_orchestrator.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/repay.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/set_traget_chain_min_fee.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/update_global_state_params.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/instructions/withdraw_stable_coin.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/lib.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/state.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract
programs/genius/src/util.rs [https://github.com/Shuttle-Labs/genius-contracts-solana]	Smart Contract

Appendix 3. Additional Valuables

Additional Recommendations

The smart contracts in the scope of this audit could benefit from the introduction of automatic emergency actions for critical activities, such as unauthorized operations like ownership changes or proxy upgrades, as well as unexpected fund manipulations, including large withdrawals or minting events. Adding such mechanisms would enable the protocol to react automatically to unusual activity, ensuring that the contract remains secure and functions as intended.

To improve functionality, these emergency actions could be designed to trigger under specific conditions, such as:

- Detecting changes to ownership or critical permissions.
- Monitoring large or unexpected transactions and minting events.
- Pausing operations when irregularities are identified.

These enhancements would provide an added layer of security, making the contract more robust and better equipped to handle unexpected situations while maintaining smooth operations.