Serai DEX

Security Assessment

April 17, 2025

Prepared for:
Justin Ehrenhofer
MAGIC Grants

Prepared by: **Gustavo Grieco and Opal Wright**

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineering director was associated with this project:

Jim Miller, Engineering Director, Blockchain and Cryptography
james.miller@trailofbits.com

The following consultants were associated with this project:

Gustavo Grieco, Consultant
gustavo.grieco@trailofbits.com

Opal Wright, Consultant
opal.wright@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
March 24, 2025	Pre-project kickoff call
April 4, 2025	Delivery of report draft
April 4, 2025	Report readout meeting
April 17, 2025	Delivery of final comprehensive report

Executive Summary

Engagement Overview

Serai engaged Trail of Bits to review the security of the Ethereum side of its decentralized exchange (DEX). The code in scope included only the Ethereum side of the DEX, where users deposit using the `Router` smart contract and the validators execute transfers out of the system.

A team of two consultants conducted the review from March 24 to April 2, 2025, for a total of three engineer-weeks of effort. Our testing efforts focused on on-chain and off-chain code for the Ethereum side of the DEX, including router operations, ERC-20 handling, and use of cryptography primitives. With full access to source code and documentation, we performed a manual review of the code in scope.

Observations and Impact

We found that the code in scope follows best practices for development. In particular, it is well documented, with comments detailing important design decisions and providing technical explanations of implementation details. We strongly recommend maintaining this standard of documentation as the codebase matures.

The review produced two informational-severity findings and some code quality suggestions, detailed in appendix B.

Recommendations

Based on the findings identified during the security review, Trail of Bits recommends that Serai take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or any refactor that may occur when addressing other recommendations.
- **Perform a security review of the components that were out of scope for this audit.** A good number of important components were not in scope for the audit. It would be beneficial to perform a security review of these components, particularly the validators and the Substrate parachain.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	0
Low	0
Informational	2
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Cryptography	1
Undefined Behavior	1

Project Goals

The engagement was scoped to provide a security assessment of the Serai's DEX. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are deposits and withdrawals to the Ethereum network correctly processed?
- Are the cryptographical primitives used correctly?
- Can an attacker execute privileged operations in the router without requiring signatures?
- Is it possible to front run transactions and negatively affect the system?
- Does the system handle ERC-20 standard tokens correctly?
- Are signatures correctly verified?

Project Targets

The engagement involved reviewing and testing the following target:

serai-dex

Repository	https://github.com/serai-dex/serai
Version	19422de231592690ac324edb57e32ba1517d1db4
Type	Blockchain
Platform	Ethereum, Rust

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **`Router` contract:** This smart contract performs the verification and execution of `InInstructions` and `OutInstructions` in order to process deposits and withdrawals. It uses Schnorr signatures for every operation except external deposits. We reviewed the code to assess whether signatures are correctly parsed and verified. We also tried to find undesired interactions, such as unexpected reverts, reentrancies, or any operation that does not follow the specification.
- **Deployer:** This component provides on-chain and off-chain code to deploy a canonical instance of the router using a keyless procedure. We verified that the procedure is sound and cannot be blocked or front-run by external users.
- **Schnorr library:** This smart contract verifies a Schnorr signature for a specified public key. We reviewed the usage and soundness of the cryptographic operations to determine if they introduce any risk.
- **Ethereum processors:** This off-chain component can craft new valid transactions to interact with the router. It listens to confirmed transactions on the Ethereum mainnet to identify certain operations addressed to the validators (e.g., deposits).
    - **Router:** This code produces valid transactions to relay on-chain and provides a precise estimation of the gas spent in each operation using reth as the execution engine.
    - **ERC-20:** This code analyzes and validates ERC-20 transactions to certain tokens (e.g., DAI) to determine if a transaction is a valid deposit.
    - **Primitives:** This code implements the cryptographic primitives to support off-chain signature creation and verification.

  We also reviewed the off-chain code for correctness issues regarding the usage of cryptographic primitives and interactions with blockchain data, as well as other common Rust-related issues. We also reviewed how gas is accounted for in the simulations to assess whether it follows the expected procedure.

Coverage Limitations

Only the following components/directories were in scope:

- `networks/ethereum/schnorr/contracts/Schnorr.sol`
- `networks/ethereum/schnorr/src`

- `processor/ethereum/router`
- `processor/ethereum/deployer`
- `processor/ethereum/primitives`
- `processor/ethereum/erc20`

As expected, the following components from Serai DEX were marked as out of scope:

- Validators and Substrate code
- Other code related to Ethereum processors except the ones mentioned above

Summary of Findings

The table below summarizes the findings of the review, including details on type and severity.

ID	Title	Type	Severity
1	Disabling validation when dealing with untrusted calldata leads to undefined behavior	Undefined Behavior	**Informational**
2	Public key checks are not explicit	Cryptography	**Informational**

Detailed Findings

1. Disabling validation when dealing with untrusted calldata leads to undefined behavior

Severity: **Informational**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-SERAI-1
Target: `processor/ethereum/erc20/src/lib.rs`	

Description

Disabling validation when decoding calldata can produce undefined behavior when dealing with ERC-20 transactions.

To perform a deposit into the Serai DEX, Ethereum users have multiple options. One involves sending an ERC-20 transaction using certain tokens directly to the router. Such a transaction will have extended calldata encoding the specific information for the deposit, called `InInstruction`.

The Ethereum/ERC-20 processor code iterates over all of the confirmed transactions, locating top-level ERC-20 transactions that can be converted to `InInstructions` deposits. Once a candidate transaction is located, its calldata is decoded, first with the original ERC-20 and then with an alternative ABI called SeraiIERC20 that includes the encoded `InInstruction` value as a list of bytes.

```
 async fn top_level_transfer(
    provider: &RootProvider<SimpleRequest>,
    erc20: Address,
    transaction_hash: [u8; 32],
    transfer_logs: &[Log],
  ) -> Result<Option<TopLevelTransfer>, RpcError<TransportErrorKind>> {
   …
   // Read the data appended after
   let data = if let Ok(call) =
SeraiIERC20Calls::abi_decode(transaction.inner.input(), true) {
     match call {
          SeraiIERC20Calls::transferWithInInstruction01BB244A8A(
            transferWithInInstructionCall { inInstruction, .. },
          ) |
          SeraiIERC20Calls::transferFromWithInInstruction00081948E0(
            transferFromWithInInstructionCall { inInstruction, .. },
          ) => Vec::from(inInstruction),
     }
```

```
    }
```

Figure 1.1: Part of the `top_level_transfer` that decodes ABI from untrusted data

However, an upcoming release of the `alloy-sol-types` library will remove the validation flag. This can be an issue with the current code, as disabling the validation for parsing the SeraiIERC20 ABI can produce unexpected results, such as this crash:

```
---- tests::in_instruction::test_erc20_top_level_transfer_in_instruction stdout ----
thread 'tests::in_instruction::test_erc20_top_level_transfer_in_instruction'
panicked at
/Users/g/.cargo/registry/src/index.crates.io-6f17d22bba15001f/alloy-sol-types-0.8.16
/src/abi/decoder.rs:173:27:
attempt to add with overflow
stack backtrace:
   0: rust_begin_unwind
             at
/rustc/eeb90cda1969383f56a2637cbd3037bdf598841c/library/std/src/panicking.rs:665:5
   1: core::panicking::panic_fmt
             at
/rustc/eeb90cda1969383f56a2637cbd3037bdf598841c/library/core/src/panicking.rs:74:14
   2: core::panicking::panic_const::panic_const_add_overflow
             at
/rustc/eeb90cda1969383f56a2637cbd3037bdf598841c/library/core/src/panicking.rs:181:21
   3: alloy_sol_types::abi::decoder::Decoder::peek_len_at
             at
/Users/g/.cargo/registry/src/index.crates.io-6f17d22bba15001f/alloy-sol-types-0.8.16
/src/abi/decoder.rs:173:27
   4: alloy_sol_types::abi::decoder::Decoder::peek_len
             at
/Users/g/.cargo/registry/src/index.crates.io-6f17d22bba15001f/alloy-sol-types-0.8.16
/src/abi/decoder.rs:179:9
   5: <alloy_sol_types::abi::token::PackedSeqToken as
alloy_sol_types::abi::token::Token>::decode_from
             at
/Users/g/.cargo/registry/src/index.crates.io-6f17d22bba15001f/alloy-sol-types-0.8.16
/src/abi/token.rs:473:21
   6: <(T1,T2,T3) as alloy_sol_types::abi::token::TokenSeq>::decode_sequence
             at
/Users/g/.cargo/registry/src/index.crates.io-6f17d22bba15001f/alloy-sol-types-0.8.16
/src/abi/token.rs:606:27
   7: alloy_sol_types::abi::decoder::Decoder::decode_sequence
             at
/Users/g/.cargo/registry/src/index.crates.io-6f17d22bba15001f/alloy-sol-types-0.8.16
/src/abi/decoder.rs:272:9
   8: alloy_sol_types::abi::decoder::decode_sequence
             at
/Users/g/.cargo/registry/src/index.crates.io-6f17d22bba15001f/alloy-sol-types-0.8.16
/src/abi/decoder.rs:321:18
   9: alloy_sol_types::types::ty::SolType::abi_decode_sequence
             at
/Users/g/.cargo/registry/src/index.crates.io-6f17d22bba15001f/alloy-sol-types-0.8.16
```

```
/src/types/ty.rs:274:9
  10: alloy_sol_types::types::function::SolCall::abi_decode_raw
  ...
```

Figure 1.2: A crash exposed when the validation flag is set to false

Normally, this type of crash will be unreachable if the calldata is coming from confirmed transactions, but in this case, the ABI is never validated on-chain, and the crash can therefore be triggered externally.

Exploit Scenario

The `alloy-sol-types` library is upgraded, introducing a bug that allows any user to remotely crash the processor.

Recommendations

Short term, do not upgrade the `alloy-sol-types` library until you are certain that it is robust enough to handle arbitrary inputs.

Long term, review the security properties and risks introduced by the usage of third-party components across the codebase.

2. Public key checks are not explicit	
Severity: **Informational**	Difficulty: **Not Applicable**
Type: Cryptography	Finding ID: TOB-SERAI-2
Target: `networks/ethereum/schnorr/src/public_key.rs`	

Description

Schnorr signatures are defined such that the point at infinity is a valid public key. However, in most instances, the point at infinity strongly suggests that a serious problem has occurred (e.g., a random number generator is not working correctly). It is usually good to explicitly check for the point at infinity and either log or disallow it.

The Schnorr signature code in the Serai codebase prohibits the point at infinity on the secp256k1 curve, but not due to an explicit check. Instead, when the public key is converted to affine coordinates, the point at infinity is represented with x/y coordinates (0, 0) and a flag to indicate that the point is the point at infinity. The point at infinity is implicitly disallowed because another check ensures that the x-coordinate cannot be zero.

In short, the zero check used in the `PublicKey::new()` function disallows the point at infinity because of an artifact of how it is represented, not because it is explicitly checked.

```
let x_coordinate: [u8; 32] = x_coordinate.into();
// Returns None if the x-coordinate is 0
// Such keys will never have their signatures able to be verified
if x_coordinate == [0; 32] {
  None?;
}
```

Figure 2.1: Zero check on x-coordinate in `PublicKey::new()`

Recommendations

Short term, add an explicit check for the point at infinity.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category does not apply to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

B. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, implementing them can enhance the code's readability and may prevent the introduction of vulnerabilities in the future.

Router

- **Considering updating the documentation in `verifySignature`**. The documentation in `verifySignature` states, "The calldata should be signed with the nonce taking the place of the signature's commitment to its nonce, and the signature solution zeroed." However, there is no field zeroed in the code. Instead, the code uses the chain ID for the signature verification.

- **Consider adding the router address in the minimal signature, since it contains a very small amount of entropy**. The current signature verification works in the case of having no arguments, which means that only a selector (four bytes), a nonce (usually a very small number), and a chain ID (0x1 in case of Ethereum mainnet) are part of the signature.

Testing

- **Consider updating external tool version detection methods**. When compiling contracts for unit tests, the Solidity compiler version is checked using a simple string comparison with a hard-coded value. We recommend a more flexible approach.

C. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On April 14, 2025, Trail of Bits reviewed the fixes and mitigations implemented by Serai for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, the Serai team has resolved the two findings described in this report. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Disabling validation when dealing with untrusted calldata leads to undefined behavior	Resolved
2	Public key checks are not explicit	Resolved

Detailed Fix Review Results

TOB-SERAI-1: Disabling validation when dealing with untrusted calldata leads to undefined behavior

Resolved in commit `184c027`. The Serai team upgraded the library with the appropriate analysis and checks. The client provided the following comments:

> *This moves to Rust 1.86 as were prior on Rust 1.81, and the new alloy dependencies require 1.82.*
> *The revm API changes were notable for us. Instead of relying on a modified call instruction (with deep introspection into the EVM design), we now use the more recent and now more prominent Inspector API. This:*
> *1) Lets us perform far less introspection*
> *2) Forces us to rewrite the gas estimation code we just had audited*
> *Thankfully, it itself should be much easier to read/review, and our existing test suite has extensively validated it.*

TOB-SERAI-2: Public key checks are not explicit

Resolved in commit `33018bf`. The Serai team added an explicit check for the point at infinity.

D. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries and government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on X and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

Trail of Bits performed all activities associated with this project in accordance with a statement of work and an agreed-upon project plan.

Security assessment projects are time-boxed and often rely on information provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test software controls and security properties. These techniques augment our manual security review work, but each has its limitations. For example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. A project's time and resource constraints also limit their use.