Nomic

Security Assessment

November 19, 2024

Prepared for:
Nomic DAO Foundation

Prepared by: **Anish Naik and Tjaden Hess**

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, as well as 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 Twitter 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.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be 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 the controls and security properties of software. 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. Their use is also limited by the time and resource constraints of a project.

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:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Anish Naik, Consultant
anish.naik@trailofbits.com

Tjaden Hess, Consultant
tjaden.hess@trailofbits.com

Project Timeline

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

Date	Event
September 6, 2024	Pre-project kickoff call
September 18, 2024	Status update meeting #1
September 27, 2024	Status update meeting #2
October 4, 2024	Status update meeting #3
October 14, 2024	Delivery of report draft
October 15, 2024	Report readout meeting
November 19, 2024	Delivery of comprehensive report with fix review addendum

Executive Summary

Engagement Overview

Trail of Bits was engaged to review the security of the Nomic blockchain and `nomic-bitcoin` library. Nomic is a decentralized custody sidechain that allows for the deposit, transfer, and withdrawal of BTC (denominated as nBTC) and integrates with the IBC protocol.

A team of two consultants conducted the review from September 3 to October 11, 2024, for a total of 10 engineer-weeks of effort. Our testing efforts focused on two core areas. First, we investigated whether there was any vector that could lead to a loss or theft of funds from the system. Second, we investigated denial-of-service (DoS) attack vectors that could bring the chain to a halt. With full access to source code and documentation, we performed static and dynamic testing of the targets, using automated and manual processes.

Observations and Impact

The Nomic codebase is well-organized and cleanly implemented. We did not identify any attack vectors that could lead to the theft or destruction of funds. The core distributing team has a strong understanding of the system's underlying invariants, and the code includes a variety of stopgaps to mitigate malicious or unexpected behavior.

However, the code-level documentation provided was insufficient given the complexity of the system. It was difficult to reason through the use-cases of the various macros, the functionality of various modules, and the end-to-end flow of transactions. Thorough documentation aids in maintaining the codebase while also aiding in future security reviews.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends the following steps prior to deployment:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.

- **Improve documentation.** The documentation for the system is insufficient. It would be beneficial to invest time into developing high-level walkthroughs, examples, and diagrams to showcase the interactions between components of the system. Additionally, enriching the inline documentation to highlight each module's and function's expected behavior would be beneficial.

Finding Severities and Categories

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

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	1
Low	0
Informational	0
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Authentication	1

Project Goals

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

- Are there any vectors that can lead to a loss or theft of funds from the system?
- Are there DoS attack vectors that could halt the blockchain?
- Are third-party protocols such as IBC and Tendermint ABCI integrated correctly?
- Do the arithmetic calculations in the system account for potential overflows, precision loss, and/or division by zero?
- Do the generated Bitcoin scripts sufficiently enforce access controls and timelocks?
- Does the emergency recovery flow ensure the correct dispersal of funds when the network halts?

Project Targets

The engagement involved a review and testing of the targets listed below.

nomic

Repository	https://github.com/nomic-io/nomic
Version	`809092f`
Type	Rust
Platform	Linux, macOS, Windows

nomic-bitcoin-js

Repository	https://github.com/nomic-io/nomic-bitcoin-js
Version	`9d8a5d0`
Type	TypeScript
Platform	Linux, macOS, Windows

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:

- **`nomic`.** Nomic is a decentralized custody blockchain that supports the deposit, transfer, and withdrawal of BTC. The blockchain mints nBTC that is backed 1:1 by BTC. Nomic also supports transfers to IBC-compatible chains and EVM-based chains. We performed a manual investigation of this system as follows:
  - We reviewed incoming BTC deposits to the bridge, and whether it is possible to replay deposits or spoof a deposit by providing an invalid proof.
  - We reviewed outgoing nBTC withdrawals, and whether it is possible to steal funds from the system or replay a withdrawal.
  - We reviewed the transfers of nBTC to native accounts and any IBC-compatible chain to identify whether it can lead to a theft of funds or the locking of funds due to a logic error.
  - We reviewed whether it is possible to launch a DoS attack by triggering a panic or bypassing transaction fees.
  - We reviewed the checkpointing logic to assess whether all incoming deposits, withdrawals, and transfers are correctly accounted for; to identify any arithmetic concerns related to miner fees and fee adjustments; and to determine if the requisite state updates are made to the current `Building` and `Signing` checkpoints and if the total reserve output is calculated correctly.
  - We reviewed the emergency disbursal logic to assess whether all funds in the reserve output are accounted for in the outputs to the various nBTC holders (including funds escrowed by IBC or held in Ethereum).
  - We reviewed the Ethereum smart contract bridge and the associated logic in Nomic to assess whether a malicious actor can steal funds by updating the validator set or spoofing a transfer.
  - We reviewed the IBC-related logic to assess whether funds are correctly (un)escrowed and minted/burned.
  - We reviewed the BTC light client and the relayer logic to determine general correctness and resilience against chain reorgs.

- **`nomic-bitcoin`.** The `nomic-bitcoin` library is used to generate and broadcast Bitcoin deposit addresses to the relayer network. We performed a manual review of this system and investigated whether the Bitcoin scripts are correctly generated (given a specific signatory set) and whether the IBC destination is calculated correctly.

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

Tool	Description
Semgrep	An open-source static analysis tool for finding bugs and enforcing code standards when editing or committing code and during build time
Clippy	An open-source Rust linter used to catch common mistakes and unidiomatic Rust code
Cargo Audit	An open-source tool for checking dependencies against the RustSec advisory database

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The arithmetic calculations performed across the various targets are of generally high quality. We did not find any issues related to arithmetic overflows, division by zero, or precision loss. It is important to note that overflow checks are turned on in production, which could result in panics in case of an overflow.	**Strong**
Auditing	There is sufficient logging throughout the system. All ABCI methods emit the expected events, and there is sufficient use of logging to alert on unexpected behavior.	**Satisfactory**
Authentication / Access Controls	We did not identify any attack vectors that could lead to an access control bypass. The critical code paths have sufficient data validation to ensure that only the authorized actors can perform the actions.	**Strong**
Complexity Management	The Nomic blockchain has a complex architecture with a large number of components. Code duplication is minimal and Nomic makes strong use of abstractions, but many modules do not have any top-level documentation and most functions are undocumented.	**Moderate**
Configuration	The system is configured to have a variety of stopgaps to prevent unexpected behavior. For example, the blockchain is resilient to potential blockchain reorgs by requiring a certain number of blocks to be mined before the deposit is accepted as valid. Similarly, there are stopgaps to deter unexpected behavior during checkpointing, withdrawals, signing, and emergency disbursals.	**Strong**
Cryptography	We did not identify any instances of misused	**Moderate**

and Key Management	cryptographic primitives that would cause undefined behavior or trigger an edge case in the state machine. However, we did identify one instance of output collision that could be triggered by a malicious relayer (TOB-NOMIC-1).	
Data Handling	The data validation within the system is generally of high quality. Function inputs are sufficiently validated, and we identified no edge cases in the serialization and deserialization of custom types.	**Satisfactory**
Decentralization	The Nomic blockchain is fully decentralized. No single actor can pause the chain or put user funds at risk. The emergency disbursal process is a powerful stopgap that ensures that user funds can be redeemed if the Nomic blockchain fails to make forward progress. Additionally, user funds transferred to an IBC-supported chain or Ethereum can be redeemed in case of an emergency disbursal.	**Strong**
Documentation	The provided documentation was sufficient to reason through the most critical aspects of the system. The inline comments were thorough in some portions of the codebase but lacking in others (e.g., IBC-related components or emergency disbursals).	**Moderate**
Memory Safety and Error Handling	The Nomic codebase does not use any `unsafe` functionality.	**Satisfactory**
Testing and Verification	The system has unit and end-to-end tests for most flows. However, testing could be improved by including more negative/adversarial test cases.	**Satisfactory**
Transaction Ordering	We did not identify any issues related to front-running or transaction ordering risks.	**Strong**

Summary of Findings

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

ID	Title	Type	Severity
1	Potential output commitment collision	Authentication	**Medium**

Detailed Findings

1. Potential output commitment collision	
Severity: **Medium**	Difficulty: **Medium**
Type: Authentication	Finding ID: TOB-NOMIC-1
Target: `nomic/src/app.rs`	

Description

Deposits to the Nomic sidechain from Bitcoin accounts include a commitment to a destination, indicating where the sidechain funds should be sent. Due to a lack of domain separation, a malicious relayer could cause deposited funds to be redirected to an ETH account rather than a Nomic native sidechain account. Figure 7.1 shows the commitment formats for each type of account:

```
pub fn commitment_bytes(&self) -> Result<Vec<u8>> {
    use sha2::{Digest, Sha256};
    use Dest::*;
    let bytes = match self {
        NativeAccount(addr) => addr.bytes().into(),
        Ibc(dest) => Sha256::digest(dest.encode()?).to_vec(),
        Fee => vec![1],
        EthAccount(addr) => addr.bytes().into(),
        EthCall(call, addr) => Sha256::digest((call.clone(),
*addr).encode()?).to_vec(),
    };

    Ok(bytes)
}
```

Figure 1.1: Commitments for `NativeAccount` and `EthAccount` use the same 20-byte format (`nomic/src/app.rs#1153–1165`)

The `NativeAccount` and `EthAccount` destinations both use arbitrary 20-byte strings, while the `Ibc` and `EthCall` formats use Sha256 digests of arbitrary-length data. A malicious relayer could process a deposit using an `EthAccount` rather than the depositor's `NativeAccount`, in which case the deposited funds would be burned.

Note that the client had already addressed this issue before the start of the review, but the updated code was not included in the commit under review.

Exploit Scenario

A malicious relayer wants to harm depositors or the network's reputation. She relays all incoming native deposits as `EthAccount` deposits, causing the deposited funds to be sent to an Ethereum account for which nobody holds the private key.

Recommendations

Short term, serialize all commitments using unique prefixes to indicate what type of deposit the commitment is for.

Long term, document the formats of all commitments and message hashing schemes for signatures. Ensure that commitments and message encodings are injective by providing a decoding routine that can be used in unit testing.

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.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Configuration	The configuration of system components in accordance with best practices
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Data Handling	The safe handling of user inputs and data processed by the system
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system's resistance to transaction-ordering attacks

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 is not applicable 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.

C. System Invariants

Below is a non-exhaustive list of invariants that we identified during the course of the audit. Note that these invariants can be written programmatically such that they can be tested using a variety of testing methodologies.

Checkpointing Invariants

- There can be only one checkpoint in the `Building` and/or `Signing` state.
- There can be up to `max_unconfirmed_checkpoints` checkpoints in the `Completed` state.
- The fee pool must always have enough funds to pay for all checkpointing fees (i.e., it should never underflow).
- All pending deposits, withdrawals, and transfers (regardless of destination) must be added to the `Building` checkpoint.
- The number of inputs in a checkpoint must be less than or equal to the `config.max_inputs`.
- The number of outputs in a checkpoint must be less than or equal to the `config.max_outputs`.
- The first output of a checkpoint transaction is the current reserve balance.
- The second output of a checkpoint transaction is the state commitment.
- The emergency disbursal outputs must account for all funds within the reserve output (excluding fees and escrowed funds on IBC chains).
- No additional inputs/outputs must be added to a checkpoint that is in a `Signing` state.
- Only checkpoints that are in a `Signing` state can be signed by signatories.
- If a checkpoint is in a `Signing` state, there must also be a checkpoint in a `Building` state.

D. Non-Security-Related Findings

The following recommendation is not associated with specific vulnerabilities. However, it enhances code readability and may prevent the introduction of vulnerabilities in the future.

- **Update the key comparison in `store-diff.rs`.** Currently, the key a is being compared to itself. Instead, the snippet highlighted in figure D.5 should be `iter_b.key().unwrap()`.

```
let key_a = iter_a.key().unwrap();
let key_b = iter_a.key().unwrap();
if key_a != key_b {
    panic!("keys differ. a: {:?}, b: {:?}", key_a, key_b)
}
```

Figure D.1: `nomic/src/bin/store-diff.rs#66–70`

E. Automated Testing

Semgrep

We performed static analysis on the Nomic source code using Semgrep to identify common weaknesses. We used several rulesets, some of which are private. To run a publicly available subset of the rules, execute the commands shown in figure D.1. Note that these rulesets will output repeated results, which should be ignored.

```
semgrep --metrics=off --sarif --config="p/rust”
semgrep --metrics=off --sarif --config="p/trailofbits”
semgrep scan --pro --sarif
```

Figure E.1: Commands used to run Semgrep

Clippy

We use `cargo clippy` with “pedantic” lints to check for areas of interest in our investigation. To replicate this set of lints, you can use the command shown in figure E.2

```
cargo clippy --no-deps --message-format=json -- -W clippy::pedantic | clippy-sarif
> clippy_pedantic.sarif
```

Figure E.2: Commands used to run Clippy

Cargo Audit

We use `cargo audit` to check for known-vulnerable dependencies.

```
cargo install cargo-audit
cargo audit
```

Figure E.3: Commands used to install and run `cargo-audit`

F. 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 a comprehensive analysis of the system.

On October 18, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the client for the issue identified in this report. We reviewed the fix to determine its effectiveness in resolving the associated issue.

In summary, the client has resolved the one issue in this report. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Potential output commitment collision	Resolved

Detailed Fix Review Results

TOB-NOMIC-1: Potential output commitment collision
Resolved in commit 58a3bcd. The destination commitment is computed from the `ed` encoding of the address, which includes a prefix byte that disambiguates the destination types. The client had already made this fix to the codebase before the start of the review, but it was not included in the commit under review.

G. 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.