Fabric Labs Zipper Protocol

Security Assessment

May 5, 2025

Prepared for:
Sergio Tacer

Prepared by: **Quan Nguyen**

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Samuel Greenup, Project Manager
sam.greenup@trailofbits.com

The following engineering director was associated with this project:

Benjamin Samuels, Engineering Director, Blockchain
benjamin.samuels@trailofbits.com

The following consultants were associated with this project:

Quan Nguyen, Consultant
quan.nguyen@trailofbits.com

Project Timeline

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

Date	Event
April 17, 2025	Project kickoff call
April 28, 2025	Delivery of report draft
April 28, 2025	Report readout meeting
May 5, 2025	Delivery of final comprehensive report

Executive Summary

Engagement Overview

Fabric Labs engaged Trail of Bits to review the security of the Zipper protocol smart contracts. Zipper is a cross-chain bridge that facilitates the conversion of external blockchain assets into Fabric-compatible zAssets. The protocol enables assets from Bitcoin, Ethereum, Solana, Dogecoin, and Litecoin to be used within Fabric's decentralized ecosystem.

A team of one consultant conducted the review from April 21 to April 25, 2025, for a total of one engineer-week of effort. Our testing focused on the deposit and withdrawal flows, fee mechanisms, role-based access control system, and error handling mechanisms in the Solidity smart contracts. With full access to source code and documentation, we performed static and dynamic testing of the codebase, using automated tools and manual analysis techniques.

Observations and Impact

The Zipper protocol exhibits well-structured architecture in its implementation of cross-chain asset bridging and privileged access management. Our assessment did not uncover any severe security vulnerabilities that would result in irreversible asset loss for users.

The primary concern we identified relates to external chain transfer failure handling (TOB-FABRIC-1). The protocol lacks a recovery mechanism for withdrawal requests that fail during execution on external chains after entering the `EXECUTION_PROCESSING` state. Although the protocol's controlled environment allows for manual intervention by the team to prevent permanent asset loss, this gap could result in temporarily inaccessible funds and a degraded user experience.

The overall implementation demonstrates thoughtful consideration of security principles, with clearly defined roles and responsibilities across the system. The modular design of the deposit and withdrawal flows enables straightforward verification of state transitions, contributing to the protocol's robustness.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Fabric Labs take the following steps:

- **Implement failure handling for external chain transfers.** Develop a mechanism to detect failed transfers and provide recovery options for withdrawal requests stuck in the `EXECUTION_PROCESSING` state.

- **Add warnings about transfer risks.** Include clear warnings in code comments, documentation, and user interfaces about potential risks when withdrawing to smart contracts, particularly in light of EIP7702.
- **Enhance role naming conventions** to accurately reflect the permissions and responsibilities of each role within the system.

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	3
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Data Validation	1
Denial of Service	1
Undefined Behavior	1

Project Goals

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

- Does the cross-chain bridging mechanism correctly mint and burn zAssets without allowing duplicate minting or unauthorized supply changes?
- Are the validator and executor role permissions appropriately restricted to prevent unauthorized actions?
- Do the deposit and withdrawal flows handle edge cases appropriately, particularly for failed transfers on external chains?
- Is there proper validation of external chain transactions before minting zAssets on the Fabric chain?
- Are fee mechanisms implemented securely with appropriate controls over fee parameters?
- Do the role-based access control systems provide adequate separation of concerns?
- Can users' funds get locked or become inaccessible due to protocol design limitations?
- Are there sufficient recovery mechanisms for interrupted or failed cross-chain operations?
- Can the protocol handle chain-specific edge cases like reorgs, congestion, or temporary unavailability?
- Is there appropriate monitoring and event emission for critical operations?
- Do the contracts implement proper pausability and emergency functions?
- Are there clear upgrade paths that maintain security and user funds?
- Does the codebase conform to industry best practices for cross-chain protocols?

Project Targets

The engagement involved reviewing and testing the following target.

zipper.contracts

Repository	gitlab.com/fabric-dapps/zipper.contracts
Version	`8498a6f38c04bff8a581425f4492f123ec092310`
Type	Solidity
Platform	EVM

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:

- **Cross-chain bridging mechanism:** We assessed the Zipper protocol's core bridging functionality that enables assets from external chains to be wrapped as zAssets on the Fabric chain. Our review focused on the deposit and withdrawal flows, examining how the system validates external chain transactions, mints and burns zAssets, and ensures that asset backing remains 1:1 throughout operations.

- **Role-based access control system:** We evaluated the permission model that governs the protocol's operations, including validator, executor, and administrative roles. We assessed the separation of concerns, permission restrictions, and potential privilege escalation vectors that could compromise the system's security.

- **Error handling and recovery mechanisms:** We examined how the protocol handles failed operations, particularly focusing on withdrawal requests that encounter issues during execution on external chains. We identified limitations in the recovery options for transactions that fail after entering the `EXECUTION_PROCESSING` state.

- **Fee management system:** We analyzed the fee collection and distribution mechanisms, including how fees are calculated, collected from users, and allocated to cover gas costs for external chain operations. We evaluated the controls over fee parameters and the handling of fees in various operational scenarios.

- **Signature validation:** We reviewed the signature validation mechanisms used in the protocol for authorizing critical operations, ensuring that only properly signed transactions from authorized entities can execute sensitive functions such as approving deposits and withdrawals.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we were unable to perform comprehensive testing of the following system elements, which may warrant further review:

- While we evaluated the core bridging functionality of the Zipper protocol, we did not perform extensive testing of its integration with all supported external chains. Different chains have unique characteristics and edge cases that may require specific consideration.

- Our review focused on the Solidity smart contracts and did not extend to the off-chain components such as relayers, validators, and executors that interact with the protocol. These components play critical roles in the system's operation and security.
- We did not assess the implementation of zToken contracts that represent the wrapped assets on the Fabric chain. A separate review of these token contracts would be beneficial to ensure that they properly integrate with the Zipper protocol and follow security best practices.

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 codebase implements only minimal arithmetic operations, primarily simple addition and subtraction for fee calculations and balance tracking. All arithmetic operations are straightforward with no complex calculations, minimizing the risk of numerical errors.	**Satisfactory**
Auditing	The contracts emit appropriate events for all key state changes, including deposit/withdrawal status transitions, configuration updates, and role assignments. These events provide good visibility for tracking cross-chain asset movements and administrative actions.	**Satisfactory**
Authentication / Access Controls	The role-based access control system implements robust authentication for administrative actions. However, role naming conventions could be improved for clarity.	**Satisfactory**
Complexity Management	The contracts use the well-established OpenZeppelin libraries for access control and proxy upgradeability. The code is modular with clear separation of concerns between different components.	**Satisfactory**
Decentralization	The protocol implements a role-based access control system that distributes responsibilities across various roles. However, the system lacks a clearly documented path for users to opt out of upgrades or exit the system during critical changes. The system would benefit from implementing these user protection mechanisms and providing clearer documentation about the intended responsibility of each role.	**Moderate**
Documentation	The protocol provides good high-level documentation through official docs and diagrams that explain the	**Moderate**

	system architecture. However, the codebase itself lacks thorough NatSpec comments for functions, state variables, and contracts. There is also insufficient documentation about the specific responsibilities of each role in the role-based access control system, making it difficult to understand the intended permissions model.	
Low-Level Manipulation	The contracts do not perform any low-level manipulation.	**Not Applicable**
Testing and Verification	The project has unit tests that provide reasonable coverage of core functionality. However, there are notable limitations: the test suite lacks integration tests with off-chain components, and it focuses primarily on Ethereum as an external chain while neglecting other supported networks. More comprehensive testing across all supported chains and with integration scenarios would strengthen the protocol's reliability.	**Moderate**
Transaction Ordering	We did not identify any significant vectors that could be exploited through front-running or similar attacks.	**Satisfactory**

Summary of Findings

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

ID	Title	Type	Severity
1	Missing recovery mechanism for failed external chain transfers	Denial of Service	**Informational**
2	Misleading role name for validator management	Undefined Behavior	**Informational**
3	Missing validation for feeTo parameter in setupFee function	Data Validation	**Informational**

Detailed Findings

1. Missing recovery mechanism for failed external chain transfers	
Severity: **Informational**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-FABRIC-1
Target: `ZipperEndpoint.sol`	

Description
The protocol lacks a recovery path for withdrawal requests when transfers fail on external chains after entering `EXECUTION_PROCESSING` status. This creates a potential fund-locking scenario that requires manual intervention.

Withdrawal requests transition from `EXECUTION_PENDING` to `EXECUTION_PROCESSING` when the executor calls the `executeRequest` function. After this status change, the executor attempts to transfer tokens to the recipient on the external chain.

The contract has a `refundWithdrawRequest` function, but it works only for requests rejected during validation. When external transfers fail after validator approval (due to smart contract rejections, token callbacks, or post-EIP7702 EOA conversions), the withdrawal remains in `EXECUTION_PROCESSING` with no automated way to refund users or retry the transaction.

Exploit Scenario
A user initiates a withdrawal to what appears to be a standard EOA address. After the withdrawal is approved by validators and enters `EXECUTION_PROCESSING` status, the recipient leverages EIP7702 to convert their EOA into a contract without a `receive` function. When the executor attempts to transfer ETH to this address, the transaction fails because the newly created contract cannot receive ETH. The withdrawal remains stuck in `EXECUTION_PROCESSING` status with no automated way for the user to receive a refund, requiring manual intervention from the team.

Recommendations
Short term, add explicit warnings in codebase comments, developer documentation, and user interfaces about potential risks of failed transfers, especially when withdrawing to smart contracts or in light of EIP7702. Document the process for users to contact support if their withdrawal gets stuck.

Long term, implement a comprehensive failure handling mechanism that can detect external chain transfer failures, transition requests to an `EXECUTION_FAILED` status, and provide options for automated retries or manual refunds by administrators.

2. Misleading role name for validator management

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-FABRIC-2
Target: `ZipperEndpoint.sol`	

Description

The `VALIDATOR_ROLE` name is misleading because it suggests that it is assigned to validators themselves, when it is actually for addresses that manage validators.

The system maintains a list of validators that need to be added or removed by a manager/admin. The function `addValidator` is restricted to be callable only by addresses with the `VALIDATOR_ROLE`, and this function is used to add validators to the system.

```
function addValidator(address[] calldata _validators) external onlyRole(VALIDATOR_ROLE)
{
  for (uint256 i = 0; i < _validators.length; i++) {
    address _validator = _validators[i];
    require(_validator != address(0), "Invalid validator address");
    require(!validators[_validator], "Validator already exists");

    validators[_validator] = true;
    validatorCount += 1;
  }
}
```

Figure 2.1: The `addValidator` function in `ZipperEndpoint.sol`

Despite being used to manage validators rather than being assigned to validators themselves, the role name implies the opposite, creating potential confusion about the role's purpose and permissions.

Recommendations

Short term, rename the role to `VALIDATOR_MANAGER_ROLE` or `VALIDATOR_ADMIN_ROLE` to accurately reflect its purpose in the system.

Long term, create comprehensive documentation for all roles in the system, clearly defining their purposes, permissions, and relationships, to avoid similar confusion in the future.

3. Missing validation for feeTo parameter in setupFee function

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-FABRIC-3
Target: `ZipperFactoryFee.sol`	

Description

The `setupFee` function lacks validation for the `feeTo` parameter, which could potentially allow setting a zero address as the fee recipient.

The contract has two functions for managing fee recipients: `setupFee` for initial configuration and `updateFeeTo` for subsequent changes. The `updateFeeTo` function validates that `feeTo` is not the zero address, indicating that sending fees to the zero address is considered invalid. However, this same validation is missing in the `setupFee` function despite setting the same critical parameter.

This inconsistency creates a security gap during the initial fee setup while preventing the same issue during updates, suggesting an oversight rather than a deliberate design decision.

Exploit Scenario

An administrator with the `EDITOR_ROLE` accidentally sets the `feeTo` parameter to the zero address when initially configuring fees for a new chain using the `setupFee` function. All fees collected for transactions on that chain are sent to the zero address and become permanently inaccessible.

Recommendations

Short term, add the missing validation check to `setupFee` to match the validation in `updateFeeTo`.

Long term, implement comprehensive test coverage for initialization scenarios. Create test cases that explicitly check for proper validation of critical parameters during initial configuration and ensure consistency between initial setup and update functions.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity, 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
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
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
Low-Level Manipulation	The justified use of inline assembly and low-level calls
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 exceeded 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.

C. 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.

1. **Adding a constructor with `_disableInitializers`.** All upgradeable contracts in the system (`ZipperEndpoint`, `ZipperFactoryChain`, `ZipperFactoryFee`, `ZipperFactoryKey`, `ZipperFactoryToken`, and `ZipperFactoryVault`) lack a constructor that calls `_disableInitializers`. This leaves the implementation contracts vulnerable to initialization attacks where an attacker could initialize the implementation directly before the proxy is deployed. Adding a constructor with `_disableInitializers` to each contract would prevent this attack vector and align with OpenZeppelin's recommended best practices.

ZipperEndpoint

1. **Removing dead code in comments.** `ZipperEndpoint` contains a commented-out `initialize`V2 function that creates confusion during reviews. Remove unused code to improve readability and document future functionality elsewhere.

D. 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 30, 2025, Trail of Bits reviewed the fixes and mitigations implemented by the Fabric team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, Fabric has resolved all three issues disclosed in this report. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	Missing recovery mechanism for failed external chain transfers	**Informational**	**Resolved**
2	Misleading role name for validator management	**Informational**	**Resolved**
3	Missing validation for feeTo parameter in setupFee function	**Informational**	**Resolved**

Detailed Fix Review Results

TOB-FABRIC-1: Missing recovery mechanism for failed external chain transfers

Resolved in commit `39900b04c83`. The team added a new status, `EXECUTION_FAILED`, to accurately represent when external chain transfers fail.

TOB-FABRIC-2: Misleading role name for validator management

Resolved in commit `39900b04c83`. The team renamed the `VALIDATOR_ROLE` variable to `VALIDATOR_MANAGER_ROLE` for better clarity.

TOB-FABRIC-3: Missing validation for feeTo parameter in setupFee function

Resolved in commit `39900b04c83`. The team implemented a zero address check for the `feeTo` parameter in the `setupFee` function.

E. 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.