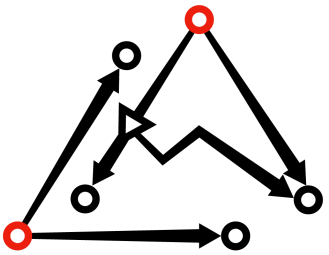

Least Authority

PRIVACY MATTERS

Magic Protocol

Security Audit Report

Final Audit Report: 22 November 2022

Table of Contents

Overview

Background

Trust Machines have requested that Least Authority perform a security audit of the Magic Protocol, an Atomic Swap for the Zest Protocol.

Project Dates

- **August 10 - September 28**: Code review *(Completed)*
- **September 30**: Delivery of Initial Audit Report *(Completed)*
- **November 7:** Verification Review *(Completed)*
- **November 22:** Delivery of Final Audit Report *(Completed)*

Review Team

- Alejandro Flores, Security Researcher and Engineer
- Nikos Iliakis, Security Researcher and Engineer
- Xenofon Mitakidis, Security Researcher and Engineer
- Akunne Obinna, Security Researcher and Engineer
- ElHassan Wanas, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of Magic Protocol followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repositories are considered in scope for the review:

- Magic Protocol:
  https://github.com/magicstx/bridge
- Supplier Job Monitor:
  https://github.com/magicstx/supplier-server

Specifically, we examined the Git revisions for our initial review:

Magic Protocol: `78e8a34eb523ca9ab2cbc51f7e4380a3e8989e63`

Supplier Job Monitor: `e875416a2f5541f563cf48b32c698d12bc6474e9`

For the review, these repositories were cloned for use during the audit and for reference in this report:

Magic Protocol:
https://github.com/LeastAuthority/magic-bridge

Supplier Job Monitor:
https://github.com/LeastAuthority/supplier-server

For the verification, we examined the Git revision:

Magic Protocol: `5b75d8dbe20ebe8ee05aae47a31b9eabbde94f34`

Supplier Job Monitor: `9033f1cb8843bbf22fce06fcec6b326a498aa250`

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Magic Protocol Audit: https://blog.coinfabrik.com/smart-contracts/magic-bridge-audit/
- Magic Protocol: https://magicstx.gitbook.io/magic-protocol/overview/magic-protocol
- Magic Audit Contracts Brief: https://metroid.notion.site/Magic-audit-contracts-brief-shared-002fd6e495724284afb3286110987444
- Supplier Job Monitor Audit Brief: https://metroid.notion.site/Supplier-server-audit-brief-shared-d05fdd7d09b54c29b54ceafcaa74cafb

In addition, this audit report references the following link:

- Stealing Chat Session ID with CORS and Execute CSRF Attack: https://infosecwriteups.com/stealing-chat-session-id-with-cors-and-execute-csrf-attack-f9f7ea229db1

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Common and case-specific implementation errors;
- Adversarial actions and other attacks on the bridge;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service attacks and security exploits that would impact or disrupt the execution of the bridge;
- Vulnerabilities within individual components as well as secure interactions between the components;
- Exposure of any critical information during interactions with any external libraries;
- Proper management of encryption and signing keys;
- Protection against malicious attacks and other methods of exploitation;
- Data privacy, data leaking, and information integrity;
- Inappropriate permissions and excess authority; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Magic Protocol aims to provide a means for holders of BTC to swap for the Wrapped bitcoin token, xBTC, on the Stacks blockchain. Our team performed a security review of the design and implementation of the Magic Protocol and Supplier Job Monitor components. The Magic Protocol governs the swap from BTC to

xBTC and xBTC to BTC, and the Supplier Job Monitor governs the liquidity providers that fund the system with xBTC (Suppliers).

In our review, we investigated Magic Protocol's `escrow` functionality and did not identify any attacks that would result in the loss of funds from Suppliers or users Swapping BTC or xBTC.

We attempted to circumvent the swap expiration mechanisms so we could transfer funds but did not identify any vulnerabilities or issues. We reviewed the `register-supplier` function for susceptibility to sending an invalid amount of funds and could not identify any vulnerabilities.

Although Magic Protocol and Supplier Job Monitor are generally well designed, implemented, and tested, we identified some issues and suggestions to improve the security of the design and the quality of the implementation. Additionally, we recommend that the project documentation and code comments be improved.

System Design

Our team found that the Magic Protocol team has taken security into consideration in the design of the Magic Protocol, as demonstrated by the implementation of Hashed Timelock Contract (HTLC) scripts. The use of the HTLC script ensures each component functions as intended and removes unnecessary layers of trust, especially in the Supplier. Our team also identified areas in which the Magic Protocol team can improve the security of the system design. We found that the Supplier Job Monitor uses basic authentication, which is insufficiently secure, as it could result in an attacker compromising private Supplier data. We recommend that a more secure authentication process be performed (Issue A).

We also found that the Supplier provides a user-selected password that is stored in plaintext, which could be extracted easily in case the system is compromised. We recommend using a memory-hard hash function and that only the hash of user-selected passwords be saved, in accordance with best practice (Issue B).

We found that the registry process for Suppliers does not include a verification that the public key registered is owned by the entity performing the registration. As a result, a user can register a public key that does not belong to them. We recommend that a key ownership verification step be performed in the Supplier registration process (Issue C). In addition, we recommend that Suppliers be enabled to off-board from the system by removing their public key (Suggestion 3).

Code Quality

Our team performed a manual code review of the Magic Protocol and Supplier Job Monitor codebases and found that the code is well organized and generally adheres to best practice. We identified areas of improvement, including instances where type checks can be improved (Suggestion 1).

We found that the use of `ERR_PANIC` in the implementation does not adhere to best practices. `ERR_PANIC` is used widely despite the Magic Protocol team's documentation recommending that it not be thrown. We recommend improving error handling and avoiding the use of `ERR_PANIC` (Suggestion 5).

We recommend using an `es lint` rule for unused imports and variables in the build, as unused imports may introduce security vulnerabilities (Suggestion 4).

Tests

Sufficient test coverage is implemented that tests for the correctness of the implementation and that the implementation functions as intended.

Documentation

The documentation provided insufficiently describes the Magic Protocol and Supplier Job Monitor, as it lacks technical details and descriptive diagrams. The documentation only gives a high level description of the code structure and functionality. The lack of developer documentation made it difficult for our team to assess in-scope components and understand the expected behavior of the Magic Protocol and Supplier Job Monitor. We recommend that the project documentation be improved (Suggestion 2).

Code Comments

The code comments in the codebase describe the inputs and outputs of functions rather than their intended behavior. Although some functions and components have sufficient code comments, we recommend that code comments be improved to include more detailed explanations of the intended behavior of each function and component (Suggestion 2).

Scope

The scope of this security review was sufficient and included all security-critical components.

Dependencies

Our team performed automated scans for vulnerable dependencies and identified several instances where these are implemented. We recommend that vulnerable dependencies be checked and that alternatives to unsafe libraries be used.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Supplier Job Monitor Uses Basic Authentication	Unresolved
Issue B: Supplier Job Monitor Uses Plaintext Passwords for Authentication	Unresolved
Issue C: Registration of Suppliers Does Not Require Verification	Unresolved
Issue D: Insecure Next.js Configuration Setting	Resolved
Suggestion 1: Use Conditional Imports	Resolved
Suggestion 2: Improve Project Documentation	Unresolved
Suggestion 3: Allow Supplier to Unregister from the System	Unresolved
Suggestion 4: Use an es lint Rule for Unused Imports	Resolved
Suggestion 5: Improve Error Handling	Unresolved

Issue A: Supplier Job Monitor Basic Authentication

Location

`supplier-server/src/index.ts#L17`

Synopsis

The Supplier Job Monitor uses basic authentication, which is known to be insufficiently secure.

Impact

Each Supplier who runs this Server is at risk of suffering a successful brute-force attack via the `BasicAuth` login. Even without being connected through a web3 extension or having any access to the private key of the Supplier Job Monitor owner, this could still give an attacker access to the dashboard, which could contain private information.

Feasibility

Performing a brute-force attack against a Basic HTTP Authentication is trivial.

Mitigation

We suggest changing the authentication type to deter the risk of these attacks.

Remediation

We recommend upgrading the codebase from `BasicAuth` to a `JWT` or `OAuth` based authentication via a secure channel, such as HTTPS.

Status

The Magic Protocol team acknowledged that a more secure authentication for the Supplier Job Monitor is ideal, but did not consider it to be critical. As such, the issue remains unresolved at the time of verification.

Verification

Unresolved.

Issue B: Supplier Job Monitor Uses Plaintext Passwords for Authentication

Location

`src/index.ts`

Synopsis

The Supplier Job Monitor compares against plaintext passwords for authentication. This allows other processes to view the password either by checking the command used for starting the Server or accessing the environment file where the password is stored. Additionally, it is easier to mount a brute-force attack in the absence of a slow hashing algorithm used for every attempt.

Impact

The attacker could gain unauthorized access to the Supplier's private information in the dashboard, including any pending operations that are not yet finalized.

Feasibility

An attack is possible if a malicious process has read access to the environment file. Furthermore, remote brute-force attacks are trivial due to the lack of a slow hashing algorithm.

Remediation

We recommend storing the password in hashed form using a slow hashing algorithm, such as Argon2. Additionally, adding another factor during authentication will help protect against stolen credentials.

Status

The Magic Protocol team acknowledged that a more secure authentication for the Supplier Job Monitor is ideal, but did not consider it to be critical. As such, the issue remains unresolved at the time of verification.

Verification

Unresolved.

Issue C: Registration of Suppliers Does Not Require Verification

Location

`contracts/bridge.clar#L121`

Synopsis

A controller may arbitrarily register any valid public key as a Supplier without verifying the ownership of the public key being registered.

Impact

Arbitrarily registering any number of public keys as Suppliers would hinder the performance and prevent users from registering themselves, thus restricting users from being able to use the system as intended.

Remediation

We recommend verifying the ownership of public keys during the Supplier registration process. This could be achieved by requiring a signed message during the Supplier registration or by alternative means.

Status

The Magic Protocol team agreed that the proposed remediation would help ensure that suppliers are configured correctly. Once implemented, this would be classified as “include in any text version” instead of as a critical hotfix. However, at the time of verification, the issue remains unresolved.

Verification

Unresolved.

Issue D: Insecure Next.js Configuration Setting

Location

`/magic-bridge/main/next.config.js#L25`

Synopsis

The function `headers` uses an insecure configuration that allows all HTTP methods from any origin and credentials (cookies) to pass through.

Impact

This type of configuration can allow an attacker to perform Cross-Site Request Forgery (CSRF) attacks by tricking a user to access a malicious URL. If the user has sessions/cookies enabled, this triggers a call to any API route in the Magic Protocol `Next.js` server, which could lead to unauthorized modifications or actions on behalf of real users, without their knowledge.

Feasibility

This attack requires some user interaction. However, a simple request to the `Next.js` server running this configuration would show these response headers, which an attacker could easily exploit to craft a CSRF attack.

Technical Details

This blog post explains a similar scenario involving this type of insecure configuration in which an attacker is able to steal the session ID with a CSRF attack.

Remediation

We recommend implementing the following configuration:

- Header `Access-Control-Allow-Credentials` should be set to `false`
- Header `Access-Control-Allow-Origin` could have a specific set of domains that are allowed to do requests if there is no need for any external domain to perform a request on behalf of the API
- Header `Access-Control-Allow-Methods` should have a defined set of HTTP methods that will be used, preferably to be specific by path, e.g.:
    - `/api/delete` can have a DELETE method
    - `/api/user` can have a GET and POST method

Status

The Magic Protocol team implemented the recommended configuration.

Verification

Resolved.

Suggestions

Suggestion 1: Use Conditional Imports

Location

`main/src/wallet.ts#L139`

`main/src/store.ts#L25`

`main/src/utils.ts#L54`

`main/src/events.ts#L94`

`main/src/stacks-api.ts#L75`

Synopsis

In some instances, the check `typeof x === 'undefined'` is used.

Mitigation

We recommend using == instead of === so the type would be casted and catch both `'undefined'` and `'null'`:

Status

The Magic Protocol team stated that they prefer not to use ==, as it has other side effects and created `helper` functions to check for "nullish."

Verification

Resolved.

Suggestion 2: Improve Project Documentation

Synopsis

Although the general documentation provides a high-level description of the system, the documentation and code comments insufficiently describe the code structure and functionality in detail. For example, there are no mentions of controls in the documentation, but they are implemented in the code. Documentation on how components of the system function serves as a critical reference point that can be compared against what has been implemented in the codebase.

Mitigation

We recommend improving the documentation to include:

- Design specifications that provide detailed and concise information about the system design and requirements. A design specification allows security auditors to check whether the code has been implemented correctly and adheres to the specification, and avoids incorrect assumptions about the expected behavior of the system, which may lead to missed security vulnerabilities;
- Architectural diagrams describing components of the system and their interaction;
- User documentation that ensures users and Suppliers interact with the system correctly and as intended and;
- Code comments that thoroughly describe the intended behavior of each function and components within the Magic Protocol and Supplier Job Monitor.

Status

The Magic Protocol team acknowledged that there are specific areas in which the documentation could be improved and will update it as applicable.

Verification

Unresolved.

Suggestion 3: Allow Supplier to Unregister from the System

Location

`contracts/bridge.clar#L232`

Synopsis

The Supplier can only update their key but cannot delete it. This makes proper maintenance of the system impossible, as even obsolete keys will be kept.

Mitigation

We recommend allowing Suppliers to be unregistered from the system.

Status

The Magic Protocol team stated that they will consider implementing the proposed mitigation in future versions of contracts. As such, the issue remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 4: Use an es lint Rule for Unused Imports

Location

`magic-bridge/blob/main/tsconfig.json`

`supplier-server/blob/main/tsconfig.json`

Synopsis

Imports should only be used for development builds (`_app.tsx`). Unused imports may introduce vulnerabilities or increase the size of the production build.

Mitigation

We recommend using an `es lint` rule for unused imports and variables in `tsconfig`.

Status

The Magic Protocol team implemented an `es lint` rule for unused imports and variables.

Verification

Resolved.

Suggestion 5: Improve Error Handling

Location

Examples (Non-exhaustive):

`contracts/bridge.clar#L141-L143`

`contracts/bridge.clar#L235-L236`

Synopsis

Error handling in the implementation is excessively defensive, with many instances of unnecessary assertions identified by our team that result in an ERR_PANIC as a response. In general, error panics should not be used, but if they are used, it should be in cases where the behavior is possible but unexpected.

There are locations in the codebase where assertions and `ERR_PANIC` are raised, although the assertion will always be `true`. Such assertions add more cost to execution for no gain in reliability.

Mitigation

We recommend avoiding the use of `ERR_PANIC` and implementing more detailed errors for specific scenarios that are possible and that provide useful information on the cause of the error.

Status

The Magic Protocol team stated that if this error is ever thrown, ERR_PANIC is used to clearly identify a logic bug in the Magic Protocol, and in future cases where there are new developed versions of the Magic Protocol contracts, unwrap-panic will be implemented. As such, the issue remains unresolved at the time of verification.

Verification

Unresolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, and zero-knowledge protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.