Least Authority
PRIVACY MATTERS

Holochain Deterministic Integrity

Security Audit Report

Holo Ltd.

Final Audit Report: 24 January 2023

Table of Contents

Overview

Background

Holo Ltd. has requested that Least Authority perform a security audit of the Holochain Deterministic Integrity crate.

Project Dates

- **September 5 - October 21:** Initial Code Review of Part 1 & Part 2 *(Completed)*
- **October 26:** Delivery of Initial Audit Report *(Completed)*
- **January 23:** Verification Review *(Completed)*
- **January 24:** Delivery of Final Audit Report *(Completed)*

Review Team

- Shareef Maher Dweikat, Security Researcher and Engineer
- Nicole Ernst, Security Researcher and Engineer
- DK, Security Researcher and Engineer
- Dylan Lott, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Holochain Deterministic Integrity crate followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repository is considered in scope for the review:

- https://github.com/holochain/holochain/tree/develop/crates/hdi

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

- `1a291fb210f5e9e506339721f3a8a9d5760f3af6`

For the verification, we examined the Git revisions:

- `972ded527128175c9afaac5bebe3f45974342d55`
- `aa72f71305cc0d14ceb4869a3587408855ea27bc`

For the review, this repository was cloned for use during the audit and for reference in this report:

- https://github.com/holochain/holochain/tree/develop/crates/hdi

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:

- Holochain Security Review document: https://hackmd.io/rfotIeMlT8SXbVwxEFOyyg

Areas of Concern

Our investigation focused on the following areas:

- Alignment of the design of the model with the Holochain team's project goals;
- Correctness of the implementation and adherence to best practices;
- Common and case-specific implementation errors in the code;
- Exposure of any critical information during interactions;
- Adversarial actions and other attacks, such as the manipulation of data;
- Networking and communication with external data;
- Vulnerabilities in the code and whether the interactions between the related and network components are secure;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Our team performed a comprehensive design review and audit of the Holochain Deterministic Integrity (HDI) crate, a critical component of the Holochain system. Overall, we found the HDI crate to be highly modular and organized in a logical, compartmentalized manner.

Our team investigated the areas of concern listed above, in addition to examining input validation implementation, error handling, as well as general implementation security: use of third-party dependencies, configuration management, secure coding and best practices. We derived several attack scenarios targeting eventual-consistency, validation, and capabilities mechanisms. We checked for correct encryption and hash use, and analyzed the serialization and deserialization logic of network messages, of which we did not identify any issues.

Although our team did not identify any security vulnerabilities in the design and implementation of the HDI crate, we found that the lack of up-to-date design documentation increased the difficulty of auditing and reasoning about the security of the system (Issue A). We also found that determining an appropriate scope for this audit was difficult given the tightly dependent nature of Holochain on the HDI crate.

System Design

The Holochain codebase is written in Rust, a language with good performance and memory safety characteristics, and utilizes WebAssembly (Wasm) as an execution environment. The Holochain team cites Wasm support and options as a major argument for their choice of Rust. We agree that Rust and Wasm are a rational choice of technologies for this application.

Holochain can be described as a content-addressable storage (CAS) system backed by a Distributed Hash Table (DHT) that allows independent validation of mutations to be carried out in the network by individual nodes called "agents." These independent validations are carried out by Wasm environments in so-called `zome` validation functions that each agent maintains. The HDI crate is built for developing and testing these `zome` validation functions. The network is built to gossip changes to a node's view of the network in discrete packages called "actions" that each agent can execute independently and deterministically. Agents must arrive at the same deterministic view of that action or else that action is considered invalid and thus rejected. This way, participants "discover their own truth" by independently walking the DHT and rendering the entries they find into a coherent tree.

HDI Crate

We examined how the HDI crate fits into the rest of the Holochain system, in addition to its usage, construction, and implementation. The HDI crate is primarily responsible for developing with `zomes` and validating the data that updates a Holochain application at runtime. The HDI crate contains several long-form code comments, which are helpful at understanding how the code relates to outside concepts. Because the HDI crate is meant to be a building block, it is tightly scoped and intentionally depends on a small number of other packages. Although this lends it to being a good unit of composition, it can make reasoning about the crate more difficult. While the code comments aid in gaining this understanding, the lack of more technical documentation to compare against the implementation is problematic (Issue A).

Zome Functions

In Holochain, changes to a given integrity `zome`'s content result in a change to its hash. Consequently, changes to that layer result in a new set of peers, which presents a security property of the Holochain network unique to content-addressed systems. Changes to the functions dictate how content is identified and thus where it is stored. This could be positively applied to introduce natural churn in an application's lifecycle in order to reduce the effectiveness of identity-based attacks, but also negatively applied by an attacker to game the system with the same category of identity attacks, such as Sybil or Eclipse attacks. Additionally, it might be a factor to consider during "migrations," or where chains need to be moved along and upgraded simultaneously. This is a common entry point for attack vectors in content-addressed systems, so we recommend close scrutiny to this interaction in the system.

Actions

Holochain promotes a local "agent-centric" view of the network. State is entirely local and there is no expectation of a consistent global state, thus categorizing Holochain as an eventually-consistent system. Agents update their view of their local state by way of actions. Actions are distinct units of change in Holochain, and they represent the core unit of gossip. Actions are defined as objects with at least an author, a timestamp, an action sequence, the action type, and the previous action. Actions can have other additional fields, but they must have at least these fields to be considered valid. These actions can be modeled as a directed acyclic graph by nature of the previous action `prev_action` and action sequence `action_seq` fields, which tie them into a chain. Agents collect these actions and build a conflict-free replicated data type (CRDT) out of them. Specifically, Holochain implements an OR Set CRDT, which maintains a tombstone set to determine the removal of members from the set. We looked at how the DHT lifecycle of actions applies to the tombstone set as it relates to the HDI crate and did not identify any issues or problems with the correctness of the implementation. Additionally, we looked for attack vectors related to manipulating the ordering or application of actions and identified no vulnerabilities using this strategy.

Zome Lifecycle

We also looked for specific places where the lifecycle of a given `zome` function, links, or entries could be abused. Since the system is eventually consistent, we were primarily searching for places where time latency might be manipulated based on a single agent's view of the network. We did not identify any issues with the DHT lifecycle but suggest further scrutiny of the networking and peer-to-peer system and recommend that close attention be paid to the eventual consistency guarantees offered by Holochain (Suggestion 3). Additionally, we recommend creating a formal specification of the eventual consistency guarantees that are described by the system (Issue A).

We closely examined the functions that HDI declares and exposes, in addition to the data validation cycle as implemented in the rest of the Holochain workflow. It is clear that security has been considered in the definition and usage of this part of the system, with comments in the code describing eclipse attack vectors and their effects on the agent's view of their own state. We did not identify any issues or vulnerabilities in the lifecycle and guarantees around delivery and validation.

Though our primary focus was the HDI crate, it was necessary for our team to examine other components to holistically understand the entire Holochain system and fully assess the security of the HDI crate. Below, we comment on some of those sections.

P2P Networking

We analyzed the P2P and networking stack, as it is a vital part of the HDI crate and its associated guarantees. Holochain's eventual consistency guarantees are heavily built around the peer-to-peer networking layer and the DHT's gossip interactions. As a result, it was necessary to understand the DHT at a high level in order to fully understand the HDI crate's place in the system. We analyzed these components and how they relate to the HDI crate and could not identify any issues or implementation errors. However, we do recommend an independent audit of the networking components in the future. (Suggestion 3).

Hashes

Hashes are a key component of the Holochain system and, thus, the HDI crate. We examined how Holochain uses and implements the hashes in the HDI crate. Holochain uses hashes for signing, encryption, and as content identifiers. We found that the Holochain team selected hashes with appropriate properties for these use cases. Specifically, we examined the use of the `blake2` hash algorithm and Holochain's hash construction for content addressing. Holochain uses `multihash` to future-proof these hashes and encodes them as `base64` strings for reliable transfer over the network. We looked for locations where any cryptographically insecure hashes were used, and instances where a secure hash was used incorrectly, and could not find any areas where hashes were used in a way that would introduce an issue or vulnerability.

WebAssembly

A key part of the Holochain system design is the use of WebAssembly to run the validation functions developed with the HDI crate. Agents run validation functions in Wasm to update their local state. These Wasm functions are metered and arbitrarily limited to 10,000 cycles to prevent situations where a validation callback could enter an indefinitely recursive state, an issue known as the Halting Problem. We examined the Wasm environment, along with how validation functions and the HDI crate as a whole interact with it and did not identify any issues or vulnerabilities.

Fuzzing

Fuzz testing is the process of using stochastic inputs to test boundary conditions within a function. Though Rust typically makes the class of bugs that fuzz tests are best at catching a non-issue, there are situations where fuzz testing can be valuable and reveal serious issues. Our team fuzz tested a set of functions from the HDI crate. We used `cargo-fuzz` to run our fuzz tests and looked for panics or unexpected errors within a given set of operating constraints. We ran each fuzz test for a minimum of 12 hours against both a corpus of known issues and an empty corpus. We did not identify any issues or panics while running these fuzz tests. However, we recommend that the Holochain team implement fuzz testing into their standard development framework (Suggestion 1).

Code Quality

The Holochain team makes heavy use of Rust crates as units of organization and separation within the codebase. We examined the structure and character of the codebase and found it to be well organized and written. The Holochain team uses a modular pattern of highly nested crates within the code to manage boundary layers between different components of the system, which promotes a cleaner system implementation.

We found evidence of test-driven design behaviors, and the codebase has comprehensive code coverage across vital components. Additionally, we looked for any usage of `unsafe` and its related traits and concurrency primitives.

The project uses Nix for managing development and production builds. There are docker containers for convenience, but Nix is the primary and preferred build system, as the docker containers simply invoke the Nix scripts. During the audit, we verified that the project code at the specified commit builds and compiles as it should from the source on a first clone. We ran into no major issues with installation and compilation and were able to install the Holochain CLI and related Holochain tooling on our local machines.

Tests

We closely examined the tests provided in the HDI crate, which are composed of a set of integration tests and unit tests that check functionality. The tests showed high coverage of important logic and covered both errors and happy paths with a high degree of specificity. The test code is clean and well organized and shows that the team has built these components from the perspective of their use and not just to accomplish a specific task. Additionally, the tests exhibit the highly composable nature of the HDI crate and its parts.

Throughout the rest of the repository, we found a similar pattern of good code coverage, unit and integration tests, and comprehensive test harnesses. The main Holochain crate includes the Sweettest package for mocking and testing the major components of their system.

Documentation

We found that the documentation provided for this review was insufficient in describing the architecture of the system, each of the components, and the interactions between those components. Our team received a set of documentation at the beginning of the audit but was subsequently informed that the documentation was out of date and that the code should be the source of truth.

Our team finds it valuable to compare an implementation against its technical specification to identify security vulnerabilities or problematic patterns. The lack of a design specification or up-to-date project documentation made it difficult to assess the state of the project and compounded the time necessary to understand the system and explore adversarial attack vectors (Issue A).

Code Comments

The repository has sufficient code comments, which were helpful and detailed.

Scope

The scope of our audit was limited to the HDI crate, which is a comparatively small area of the overall Holochain codebase. We felt it necessary to understand the Holochain system as a whole in order to sufficiently audit the HDI crate. As such, we recommend that all modules composing the Holochain system be reviewed by a third-party security team (Suggestion 3).

Dependencies

We ran a dependency analysis on the HDI crate with `cargo-audit` and did not identify any out-of-date dependencies that `cargo-audit` considers high priority. We recommend that the Holochain team implement automatic dependency checks into their development workflow. Our team also checked Ed25519 curve dependencies for vulnerabilities but did not identify any. We recommend that the Holochain team continue to monitor the `libsodium` library for security updates (Suggestion 2).

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: No Design Specification	Resolved
Suggestion 1: Add Fuzz Testing to CI/CD Pipeline	Resolved
Suggestion 2: Monitor libsodium_sys Safety	Resolved
Suggestion 3: Audit the Holochain Repository	Unresolved

Issue A: No Design Specification

Synopsis

Our team found that there is no up-to-date design specification for HDI or the Holochain system in general, which makes the system more difficult to reason about and audit. A design specification identifies each component and describes why it is used and how it contributes to achieving the objectives of the system.

At present, to recreate the design specification, one must reverse engineer based on the codebase or ask team members questions regarding the design. Both approaches are highly ineffective and inefficient and should not be used in the development process of systems where security is a critical priority.

Although there are no guaranteed ways to build a safe and secure system, a design specification provides a frame of reference that enhances the security of the design and implementation of a system, and maximizes the effectiveness of third-party security audits.

Impact

The lack of a design specification may lead to flaws and weaknesses being introduced during the design phase that would be expensive and difficult to fix at a later point. It also creates ambiguities and vulnerabilities in the implementation of the system since there is not a single reliable source of information that all developers agree on.

Mitigation

We recommend that the Holochain team create a comprehensive design specification and keep it up to date as the project evolves. This document should contain the following sections: design goals, assumptions, threat model, system architecture, protocol specifications, secure design considerations, and implementation considerations.

Status
The Holochain team created a design specification as recommended.

Verification
Resolved.

Suggestions

Suggestion 1: Add Fuzz Testing to CI/CD Pipeline

Location
GitHub and the `.github/` folder in the provided repository, if the team chooses to use GitHub Actions.

Synopsis
Fuzz testing has become a standard tool of advanced network protocols like Holochain to test for obscure and difficult-to-find bugs. Moreover, tooling exists that allows fuzz tests to be run on each pull request (e.g., with GitHub Actions).

Mitigation
We recommend creating a GitHub Action for running fuzz tests, or employing a similar tool for running fuzz tests on each pull request or full release.

Status
The Holochain team used GitHub actions to introduce fuzz testing to two Holochain repositories, `holochain-wasmer` and `holochain-serialization`, and noted that they will further implement it in other areas in Holochain in the future.

Verification
Resolved.

Suggestion 2: Monitor libsodium_sys Safety

Location
`hdi/src/hdi.rs#L24`

Synopsis
An insecure implementation of the Ed25519 algorithm has left tens of cryptography libraries vulnerable to attacks. The implementation allows attackers to steal private cryptographic keys. Some libraries have patched the vulnerability, but others have not. The Ed25519 algorithm is being utilized in the project through the `libsodium_sys` dependency.

Mitigation
Although we could not identify any vulnerabilities affecting `libsodium_sys` at this time, we suggest staying up to date with the latest news and security developments relevant to this dependency, and functions relating to the Ed25519 algorithm.

Status
The Holochain team stated that they are monitoring `libsodium_sys` in their development pipeline with Rust dependency checker.

Verification
Resolved.

Suggestion 3: Audit the Holochain Repository

Location
`Holochain-Determinstic-Integrity/`

Synopsis
The scope was a difficult problem for this audit because of how dependent other parts of the system are on the HDI crate. To fully understand the security implications of the HDI crate, we had to look outside of it and examine how it fit into the rest of the system.

Mitigation
We recommend that the Holochain team audit the entire Holochain repository, ideally after the completion of a formal specification of the system.

Status
The Holochain team stated that there are planned audits of other areas of the Holochain system, which are still being defined in the specification of the system and the formal design elements.

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.