IOHK

Software Trust Review
Phase 1+2 Final Report

Published March 5, 2020

CONFIDENTIAL INFORMATION ENCLOSED

Contents

Introduction

The Research and Development (RD) branch of root9B (R9B) conducted a static Software Trust Review of a snapshot of the specified IOHK repositories, broken into two phases. R9B sought to identify common classes of vulnerabilities that might compromise the confidentiality, integrity, or availability of the software. We cloned the following repositories at the beginning of Phase 1 and evaluated them at the following commits:

- cardano-base: 1c9e91cc8cc1deecdd7696d0f7b0b74b8984aa93
- cardano-crypto: 4590efa638397e952a51a8994b5543e4ea3c1ecd
- cardano-ledger: 5be59bf1e3aa4c35603c8c84add0a6a074bcaff8
- cardano-node: 4f770763bfb787232902d05146d425e38241712f
- cardano-wallet: d29879d75c6a4ae8c515c869b6b112479ad95b8b
- daedalus: f61a20762451e31ea1bbf73bdf8028ce85408b6b
- ouroboros-network: 797dd912a3ac99179569ced2e010968306c51483

As detailed in the Statement of Work (SOW), R9B's audit did not include dynamic analysis or potential risk validation, which could include creation of proof-of-concept attacks. As a result, areas of potential risk identified in the code should be considered starting points for validation.

In our Software Trust Review, we identified many classes of potential weaknesses that were entirely avoided or well-handled by IOHK and some that posed further potential risks.

Positive Findings

A. One finding notwithstanding, we found the examined repositories properly employed secure random sources (e.g., Crypto.Random) selecting insecure choices (e.g., System.Random) only for test code.
B. Language guarantees largely ensured memory safety, including such items as the prevention of buffer overruns and use after free.
C. The Haskell repos avoided the risky practices of evaluating interpreted code.
D. No inappropriate use of static keys or embedded private keys (excluding test keys).
E. We searched for signs of classic backdoors that could provide Remote Code Execution (RCE) capabilities to intruders and found no such functionality.
F. The repos requiring external Node.js packages, such as Daedalus, are up to date, with versions of libraries that have had fairly recently released RCE vulnerabilities and Cross-Site Scripting (XSS) vulnerabilities.
G. The repos requiring OpenSSL Haskell bindings have the most recent version, which mitigates recent OpenSSL vulnerabilities and bugs.
H. We did not find insecure SQL queries in the SQLite interface.

Suspicious Items

1. Critical Insecure Genesis Key Generation

Vulnerability in Genesis Block Generation. Cardano node CLI (cardano-node/app/cardano-cli.hs) defines arguments for genesis block generation. All arguments are standard parameters except for "secret-seed." Seed is either as specified in argument (example given in scripts/genesis.sh is 2718281828) or generated in cardano-node/src/Cardano/CLI/Genesis.hs. This file generates a seed with runSecureRandom but the seed is only 32 bits. Key generation runs under a deterministic RNG from the seed in cardano-ledger/src/Cardano/Chain/Genesis/Generate.hs. This leaves only 2^{32} possibilities for each set of private keys; runs with the same parameters create the same keys and seeds. Anyone with a copy of a blockchain's genesis block can brute force the seed.

One Full Genesis Generation in a VM on an i7-8750H took 0.167s. To try all possible 32-bit seeds would require 4,294,967,296 tries, while the expected value of number of tries before success is 2,147,483,648. Given that running in parallel on all 12 threads of the i7-8750H is about 10x faster than serially, this single laptop will expect to find the secret seed and all keys in about 415 days without optimization (or equivalently, an attacker can expect to succeed in a brute force attack today for about $5,000 using AWS cloud resources). Further optimizations are possible. An attacker who successfully recovers the seed can generate the genesis block and all associated files. Generation reveals the secret keys of all initial nodes holding all cryptocurrency at start of blockchain in the seed and key files. An adversary would be able to steal all cryptocurrency remaining in or transferred to the initial addresses.

We recommend fixing the generation process in the code to guard against future insecure blockchain creation. For any existing blockchains, we also recommend transferring all value out of initial addresses/vouchers or verifying that the seeds were not generated with the code here.

2. Minor Code Practice – ReadFile

ouroboros-network/Win32-network/cbits/Win32Async.c in HsAsyncRead calls ReadFile specifying both an lpNumberOfBytesRead parameter and an lpOverlapped parameter. Per documentation at https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-readfile "Use NULL for this parameter [lpNumberOfBytesRead] if this is an asynchronous operation to avoid potentially erroneous results." If synchronous operation is desired instead, the value assigned to that parameter (numBytesRead) should be returned to the caller instead of discarded and lpOverlapped should not be specified. If asynchronous operation is desired, only the lpOverlapped parameter should be set.

3. Minor Code Practice and Potential Resource Usage – Async Read

ouroboros-network/Win32-network/src/System/Win32/Async.hs implements readHandle by initiating an async read then calling waitForCompletion, which will, via an MVar, wait for c_GetQueuedCompletionStatus to complete in another thread, which passes in maxBound for timeout, which will be passed to HsGetQueuedCompletionStatus and then GetQueuedCompletionStatus, which can result in an infinite wait. ouroboros-network/Win32-network/test/Test/Async.hs includes a test interrupting this operation by calling killThread, which in combination with the surrounding code will close all the pipe and port handles. Although closing all related handles will break the infinite wait, it is a blunt hammer. Instead, prefer CancelIoEx to cancel a request individually or use timed instead of infinite waits.

4. Minor Potential Resource Usage/Denial of Service (DoS)

ouroboros-network/src/Ouroboros/Network/Socket.hs implements lower levels of the network stack, including socket creation and connection primitives in connectToNode. When used as a client, it sets the ReuseAddr and if applicable, ReusePort socket options, calls Socket.connect, wraps in socketAsMuxBearer, and proceeds to runPeerWithByteLimit. The mux bearer acts as a thin wrapper of Socket.sendAll and Socket.recv calls. Without setting a socket timeout or using poll or select, these may block forever, as the remote side may simply continue to keep a connection alive without sending or receiving data. This will tie up resources on the node.

5. Minor Potential Protocol Incompletion – Static Node Set

cardano-node/app/chairman.hs implements main, which parses ChairmanArgs, which includes core node IDs, and calls runChairman, passing those IDs. The runChairman code in cardano-node/src/Cardano/Chairman.hs then uses mapConcurrently to spawn off threads connecting to each core node, running connectTo against each. Other uses of connectTo in this repository appear to be just for client/wallet/TX submission, suggesting a cardano node does not start any new node connections after its first start. However, as time goes on, new nodes may join the network, old nodes may leave, and node connections may not be initially successful or may disconnect. When proper, the node should routinely re-establish connections.

6. Minor Primitive Usage – Mock Crypto

In ouroboros-network/ouroboros-consensus/src/Ouroboros/Consensus/Protocol/Praos.hs, PraosMockCrypto (using MD5, MockVRF, and MockKES) and PraosStandardCrypto (SHA256, SimpleVRF, and SimpleKES Ed448DSIGN) are defined. PraosStandardCrypto is not referenced outside of this file, while PraosMockCrypto is used elsewhere in the node and protocol implementation. SimpleStandardCrypto also uses MD5. We recommend consolidating on only the release implementation.

7. Moderate Weakened Protections – CSP in Electron App Daedalus

The Daedalus wallet has the following code in the index.html base page:

```
<meta
  http-equiv="Content-Security-Policy"
  content="
    default-src 'self';
    script-src 'self' 'unsafe-eval' 'unsafe-inline';
    style-src 'self' 'unsafe-inline';
    img-src 'self' data:;
  "
>
```

The unsafe-* portions effectively remove any protections against XSS (i.e., script injection) or other injected content Content Security Policy (CSP) directives provide. We recommend taking those unsafe-* items out in future releases. **Please Note:** R9B did not find any exploitable XSS vulnerabilities. However, if the way Daedalus renders components changed, introducing such a vulnerability or an attacker discovers such a vulnerability in an underlying library, CSP would not defend against exploitation.

8. Minor – Blake Hash Function Only Performed Once When Applying a Spending Password

We recommend increasing the number of rounds the hash function is performed in "function changeSpendingPassword(config, _ref)" in Daedalus. A key derivation function (KDF) from a password should not be a single round of a hash; ideally it should include a large work factor to limit brute force attacks. In this case, the risk is low, but we still recommend best practices like a standard KDF.

9. Potential Address Randomization Suggestion

Cardano-wallet includes a --random-port option. We did not see a reason for this stated in comments; one potential reason is simply to avoid port conflicts, for which the current code is sufficient. But another reason for port randomization is to provide another layer of defense against cross-site request forgery, XSS, and DNS rebinding. For those purposes, a simple port randomization (with a maximum of 65k possibilities for an attacker to try) is less likely to be effective unless also combined with localhost IP randomization; i.e., randomize the last 3 octets of the IPv4 localhost address: 127.X.Y.Z as well as the two bytes of the port. This approach results in up to 2^{40}, that is, over 10^{12} potential address/port pairs, which adds a strong defense against such attacks. Please note that we did not see any such vulnerabilities present in the code. This recommendation is only a potential hardening feature to defend against them should they arise.

10. Potential Future Issue - Payment URI (daedalus:// or similar)

Uniform Resource Identifier (URI) handling errors are frequent in mobile applications and can lead to the intercept of data OR used to load malware on a device. If this is slated for a future release, consideration should be given to ensuring the URI handling is done appropriately.

A URI is a string of characters that can be used to identify a location, resource, or protocol. Cryptocurrencies use these URIs to pass payment details or wallet information. Improper handling of a Daedalus URI could result in being redirected to some other fraudulent web app or page. If the user were to submit a payment to the incorrect portal, the funds could be unrecoverable. A current search of CVEs on CVEDetails.com reveals hundreds of different URI-related issues.

Specific to cryptocurrencies, Bitcoin Wiki addresses potential URI abuse in https://en.bitcoin.it/wiki/Weaknesses. Referred to as "App Links" and "Intent Links," the Android ecosystem has had URI problems where other, malicious programs were able to register a different handler and intercept details about the transaction which normally would have been protected via HTTPS or gone directly to the correct application (https://people.cs.vt.edu/gangwang/deep17.pdf). Consideration should be given to whether this is a required feature, measured against the potential for abuse or injected vulnerabilities.

11. Theoretical DoS Vulnerability

a. Theoretical DoS Vulnerability

As excluded in the SOW, R9B did not audit the cryptographic proofs in the relevant Ouroboros whitepapers and did not audit whether the implementation matched the theoretical foundations in such works. However, we did consult such papers at a surface level as a reference to expand upon code comments. This included assisting in defining terms, identifying components of the source code, and understanding the threats the code sought to be secure against. The first we consulted, “Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol” dated July 20, 2019, found at https://eprint.iacr.org/2016/889.pdf, reinforces our confidence in our findings.

Section 7 of this Ouroboros paper demonstrates how “Anonymous Communication” protocols are necessary to be secure against a non-delayed adversary corruption model. Although the paper suggests mix networks and dining cryptographer-based protocols, we do not see any implementation of these protocols. Nevertheless, suppose such protocols are implemented.

Mix networks and dining cryptographer-based protocols offer, at best, guarantees an adversary cannot tell what node originates a message among participating nodes. However, in each leader selection round, Ouroboros requires all stakeholders to broadcast a message (all elected stakeholders, but the improved-anonymity version elects all stakeholders, e.g., “we also change the environment behavior by requiring that it activates all users at every slot”).

We first assume an adversary can learn the IP addresses of participating nodes. In practice, adversaries may launch a Distributed Denial of Service (DDoS) attack against one node during one leader selection round, then observe which stakeholder's message did not get through to deduce the identity of that stakeholder. We will label an adversary with this capability a "DDoS-A-Few-At-A-Time adversary" and such an adversary can rapidly learn the whole set, one identity for each slot. This type of adversary is explicitly not in the Ouroboros threat model, since the paper limits the number of parties an adversary can *ever* corrupt ("a slot is marked as adversarial if the owner of that slot ever fell under adversarial control during the protocol" also "being offline during periods when the shareholder is supposed to participate" is called a "desynchronization attack" and "Our model allows parties to become desynchronized by incorporating them into the adversary."). Real adversaries with botnets are usually only limited by the number of nodes they can DDoS *at the same time*.

"Booter" or DDoS-for-hire services are commonly provided by botnet owners. A survey of prices from various sources (e.g., https://archive.is/cmWar or https://archive.is/iCUeB) shows powerful DDoS capabilities commonly run $5-50 per target. It is quite feasible for nearly any attacker to be able to perform a DDoS attack at will against at least one or two targets at a time and rapidly change those targets if desired, eventually targeting all users at some point, although not a significant percentage at the same time.

As an aside, the activate-all-users modification is necessary for an effective anonymous protocol under these conditions. If it is not in place, a passive observer can, in each round, identify which nodes are transmitting. All elected stakeholders must be in that set, which reveals about one bit of information (in the set or not) for all nodes. By keeping track of these set memberships, the nodes will be quickly identified. This attack will proceed at optimal speed if size of committee (m) is roughly half of all nodes and stake is roughly evenly distributed, in which case the speed will be comparable to a binary search. After roughly $\log_2$(total nodes) iterations, most nodes will be identified.

Either way, once an attacker completes such de-anonymization, the attacker will then be able to knock offline the selected leader in each slot and violate the Existential Chain Quality (∃CQ) security property, effectively disabling the blockchain.

In summary, we show that although the Ouroboros paper proves key security properties (CP, HCG, ∃CQ) against a class of adversaries, we described an adversary (DDoS-A-Few-At-A-Time) that is not in that class, showed that most adversaries in the real world are likely to be able to obtain this capability, key security assumptions of the theorems do not hold for these adversaries, and such an attacker has a strategy to deny the key security properties.

b. Applicability

We next referenced the "Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain" paper dated November 14, 2017 at https://eprint.iacr.org/2017/573.pdf. A brief review of this paper shows that if the key properties of its components hold as described, Praos does not share this flaw. Prior to generating and diffusing a block, only the slot leader knows it will be a slot leader, meaning ordinary users (and the attacker) are unable to predict which nodes/users will be leaders and attack them. However, this attacker model is still not in the Praos threat model, so we have not seen proof that Praos maintains its security properties against such an attacker.

As far as implementation goes, source comments in the ouroboros-network repository at ouroboros-consensus/src/Ouroboros/Consensus/Protocol/Praos.hs and ouroboros-consensus/src/Ouroboros/Consensus/Ledger/Mock/Block/Praos.hs indicate “this Praos is merely a proof of concept.” After Phase 2, we suggest a full engagement to complete the assessment of threat models and cryptologic theory and application and verify implementation.

Phase 2

Following Phase 1, R9B RD conducted a static Software Trust Review of a snapshot of the remaining specified IOHK repositories. RD sought to identify common classes of vulnerabilities that might compromise the confidentiality, integrity, or availability of the software. RD cloned the repositories at the beginning of the engagement, and evaluated them at the following commits:

- cardano-prelude at c40175657bc8134e0561d9b6e6334728c19112e4
- cardano-shell at bc3563c952d9f3635e1c76749b86b0a24f7e4b83, and
- iohk-monitoring-framework at 5ca0f73db28132181eee6b402db8d84d3f6f51ee

In Phase 2, we also identified additional classes of potential weaknesses that were entirely avoided or well-handled by IOHK and some that posed further potential concern.

Positive Findings

A. We found no security vulnerabilities or even areas in need of improvement in cardano-prelude.
B. Web monitoring frontend avoids XSS and CSRF issues through framework operation.
C. Web monitoring frontend avoids arbitrary deserialization and code evaluation bugs through interfaces.
D. Web monitoring frontend only listens on 127.0.0.1, avoiding remote attacks.
E. Web monitoring configuration editing does not seem to allow arbitrary paths to be set, avoiding arbitrary file write vulnerabilities.

Suspicious Items

1. Update Process

The cardano-shell repository includes various files relating to its update process. PLAN.md and the current state of the code imply much of the update work is yet to be completed. (LAUNCHER.md says the “source of most of that update logic can be found in cardano-auxx" but cardano-auxx is not in the repositories we assessed.) The code in cardano-launcher/src/Cardano/Shell/Update/Types.hs nevertheless includes some functionality, even if only at a draft level. It matches descriptions in docs/update-system.md and associated diagrams. Most of the code and the documentation focus on the mechanics of running an updater and restarting into an updated version of the code. These seem generally solid, with a low likelihood of introducing path traversal or command injection flaws.

One important consideration is to ensure the download code does verify the blake2b hash of the update file. We did not see code to perform the download in this repository. A second and larger concern is that the documentation and skeleton code referenced pulling update versions and hashes from blocks in the blockchain. Not documented are what criteria should be sufficient to determine a provided hash is valid, and how those are verified. For example, can any stakeholder advertise a new hash that all nodes should download and update to? This would be a significant vulnerability. Maybe instead, like other software, only IOHK should be able to distribute updates. If so, there could be a master asymmetric key that IOHK closely guards that must sign any valid updates.

2. IOHK Monitoring Web Frontend Exposure

The IOHK Monitoring Framework includes a web front end that allows a significant amount of modification of logging and backend configuration while a process using it is running. Among other capabilities, it includes the ability to filter log messages by a wide variety of customizable criteria, view buffers, export configuration, and enable/disable traces. It also includes the ability to set the minimum severity level to be reported. This interface is open on a localhost port but does not implement any other access control. This allows local attackers in another process, potentially running with less privileges, to both observe information and modify configuration in ways that would not be possible otherwise.

a. Exposing log data

Logged data can be viewed by anyone who can connect to the port by various techniques. Even those without permission to directly open and read the output files can obtain logged data by creating specific filtering and aggregation rules and viewing the buffer tab in the web interface. The scope of this audit does not include all of the potential uses of this framework so we will not list specific data at risk of leaking, but debug-level logging as a rule commonly includes significant sensitive information.

b. Preventing logging

By editing filters or raising the minimum severity level, attackers could instead prevent critical logging information from being recorded. These actions could be part of a larger attack, to cover up tracks, or hide alerts from being discovered, for example.

c. Configuration Export

Configuration export functionality reveals the full server path of the IOHK monitoring code. While this is normally not a security vulnerability, it is considered a best practice to not do so. Full paths reveal information about the server that is not necessary and may aid the exploitation of path-related vulnerabilities. Ideally only the file name or request path should be visible to clients.

Issues with Analysis – All Phases

a. Unimplemented Items

At multiple points we encountered areas where code was either a simple stub or did not represent the methodology within the given whitepapers, frequently annotated by comments explaining work to be done. We recommend leaving in-progress code to non-master branches and removing code when replaced to ensure no dead or non-functional code lies within the master branch.

b. Exclusions

This was a static-only assessment. A full assessment requires bringing in dynamic analysis, such as traffic generation, node syncing, and wallet creation to verify assumptions made about how the code functions. Library usage remained out of scope even though module-to-module interfaces, whether to IOHK or third-party dependencies, often pose risks. A full assessment would likewise fully analyze and validate threat models and cryptographic theory and practice.