Icarus Wallet Security Audit

Final Report, 2018-09-18

FOR PUBLIC RELEASE

Contents

1 Summary

IOHK hired Kudelski Security to perform a security assessment of the "Icarus" Cardano wallet, providing access to source code, documentation, and review guidelines including references to the most critical components. The repositories concerned are https://github.com/input-output-hk/rust-cardano/ and https://github.com/input-output-hk/js-cardano-wasm as well as private git repositories including the front-end and backend-end code.

This document reports the security issues identified and our mitigation recommendations, as well as our general assessment of the wallet implementation and architecture. A "Status" section reports the feedback from IOHK developers, and includes a reference to the patches related to the issues reported. One issue present in the initial report was removed from the final report, after developers noticed that we had misunderstood the expected functionality.

This first version of the report includes:

- 3 security issues of low severity
- 11 observations related to general code safety

The audit was lead by Dr. Jean-Philippe Aumasson, VP Technology, jointly with Yolan Romailler, Cryptography Engineer, and involved 6 person-days of work.

2 Findings

This section reports security issues found during the audit.

The “Status” section includes feedback from the developers received after delivering our draft report.

Findings are codenamed IC-XXX-F-NNN with IC for Icarus, where

- XXX can be CRY (Cryptoxide), CAR (Cardano), CJW (Cardano JS wasm), WFE (wallet front-end), WBE (wallet back-end).
- NNN is a counter in 001, 002, etc.

2.1 IC-CRY-F-001: Missing sanity checks in Ed25519 API

Severity: Low

Description

In `ed25519.rs`, `ed25519`’s `signature()` does not validate `secret_key`’s length (64 bytes), hence the program will panic if a shorter value is received.

`sign_extended()` has a similar issue.

Likewise, `verify()` should validate `public_key`’s length (32 bytes), otherwise the program will panic from `Fe::from_bytes()`, and `exchange()` should verify both keys’ lengths.

Recommendation

Add `assert_eq!()` statements to ensure that the received keys have the right size.

Status

Addressed in https://github.com/input-output-hk/rust-cardano/issues/189.

2.2 IC-CRY-F-002: Unsafe ReadBuffer methods implementation

Severity: Low

Description

In `buffer.rs`, some of the `ReadBuffer` trait methods' implementations could potentially create an integer underflow or overflow:

```
fn position(&self) -> usize { self.capacity() - self.remaining() }

(...)

fn rewind(&mut self, distance: usize) {
    self.pos -= distance;
    self.parent.len -= distance;
}

(...)

fn take_next<>(&mut self, count: usize) -> &mut [u8] {
    let r = &mut self.parent.buff[self.pos..self.pos + count];
    self.pos += count;
    self.parent.len += count;
    r
}
```

If triggered by the caller, these will result in a panic in debug mode (though not in release mode).

Callers of these functions in Cardano don't seem to create any risk.

Recommendation

Verify that the arguments are of sizes that won't underflow/overflow the arithmetic operations.

Status

Addressed in `https://github.com/input-output-hk/rust-cardano/issues/190`.

2.3 IC-CAR-F-003: Panic upon deserialization of malformed blocks

Severity: Medium

Description

In `RawBlock::to_header()` method unwraps a `Result` from the `decode()` method, which gets this result from `deserialize()` as implemented for `Block` values:

```
impl RawBlock {
    pub fn from_dat(dat: Vec<u8>) -> Self { RawBlock(dat) }
    pub fn decode(&self) -> cbor_event::Result<Block> { RawCbor::from(&self.0).deserialize() }
    pub fn to_header(&self) -> RawBlockHeader {
        // TODO optimise if possible with the CBOR structure by skipping some prefix and some suffix ..
        let blk = self.decode().unwrap();
        blk.get_header().to_raw()
    }
}
```

However, `deserialize()` can return an `Err` value, for example if the second byte of the block is not 0 or 1. The program will then panic when calling `unwrap()`, which expects an `Ok` value.

Recommendation

The potential error should be handled, for example using a `math` or `unwrap_or_else()`.

Status

Addressed in https://github.com/input-output-hk/rust-cardano/issues/191.

2.4 IC-CJW-F-005: Potential out-of-bound read

Severity: Low

Description

`fromSeed()` takes a seed and gets wallet data from it:

```
export const fromSeed = (module, seed) => {
    const bufseed = newArray(module, seed);
    const bufxprv = newArray0(module, 96);
    module.wallet_from_seed(bufseed, bufxprv);
    let result = copyArray(module, bufxprv, 96);
    module.dealloc(bufseed);
    module.dealloc(bufxprv);
    return result;
    };
```

Here `wallet_from_seed()` calls the unsafe `read_seed()`, which reads `SEED_SIZE=64` bytes from the seed's address:

```rust
unsafe fn read_seed(seed_ptr: *const c_uchar) -> hdwallet::Seed {
        let seed_slice = std::slice::from_raw_parts(seed_ptr, hdwallet::SEED_SIZE);
        hdwallet::Seed::from_slice(seed_slice).unwrap()
}

..

#[no_mangle]
pub extern "C" fn wallet_from_seed(seed_ptr: *const c_uchar, out: *mut c_uchar) {
    let seed = unsafe { read_seed(seed_ptr) };
    let xprv = hdwallet::XPrv::generate_from_seed(&seed);
    unsafe { write_xprv(&xprv, out) }
}
```

Therefore, if the `seed` passed to `fromSeed()` is less than 64 bytes, the program will read out of the argument's bounds.

This seems difficult to exploit to leak memory from the wallet application, however: `fromSeed()` is called in the wallet client's `getCryptoWalletFromSeed`, where the seed is obtained from decrypting an encrypted seed:

```javascript
export function getCryptoWalletFromSeed(
  walletSeed: WalletSeed,
  password: string
): CryptoWallet {
  const seed = decryptWithPassword(password, walletSeed.encryptedSeed);
  const seedAsArray = Object.values(seed);
  const wallet = Wallet.fromSeed(seedAsArray).result;
  wallet.config.protocol_magic = protocolMagic;
  return wallet;
}
```

Since the encrypted seed is not authenticated, an attacker could truncate the encrypted data in order to force the usage of a shorter seed. Indeed, the `encryptedSeed` is stored in the browser’s localStorage, and may therefore be manipulated by attackers.

Similar out-of-bound reads may happen when calling `sign()` and `verify()`, which call unsafe Rust code under unchecked length assumptions, in respectively `wallet_tx_sign()` and `wallet_tx_verify()`.

Recommendation

Validate the size of the seed in `fromSeed()` at least, and preferably in `decryptWithPassword()` as well.

Do similar validations to eliminate the risk in `sign()` and `verify()`, and check for similar unchecked length in other exposed functions.

Status

Addressed in https://github.com/input-output-hk/js-cardano-wasm/issues/25.

2.5 IC-CJW-F-007: Lack of parameter validation in signing and verify functions

Severity: Informational

Description

The following wasm wrapper functions call the signing and verification logic given their arguments and configuration, but do not explicitly validate the parameters’ validity, and will for example panic if an `unwrap()` fails:

```rust
    #[no_mangle]
    pub extern "C" fn wallet_tx_sign(cfg_ptr: *const c_uchar, cfg_size: usize,
    ↪ xprv_ptr: *const c_uchar, tx_ptr: *const c_uchar, tx_sz: usize, out: *mut
    ↪ c_uchar) {
        let cfg_bytes : Vec<u8> = unsafe { read_data(cfg_ptr, cfg_size) };
        let cfg_str = String::from_utf8(cfg_bytes).unwrap();
        let cfg : Config = serde_json::from_str(cfg_str.as_str()).unwrap();
        let xprv = unsafe { read_xprv(xprv_ptr) };
        let tx_bytes = unsafe { read_data(tx_ptr, tx_sz) };

        let tx : tx::Tx =
    ↪ raw_cbor::de::RawCbor::from(&tx_bytes).deserialize().unwrap();

        let txinwitness = tx::TxInWitness::new(&cfg, &xprv, &tx.id());

        let signature = match txinwitness {
            tx::TxInWitness::PkWitness(_, sig) => sig,
            _ => unimplemented!()
    ↪ // this should never happen as we are signing for the tx anyway
        };
        unsafe { write_signature(&signature, out) }
    }

    #[no_mangle]
    pub extern "C" fn wallet_tx_verify(cfg_ptr: *const c_uchar, cfg_size: usize,
    ↪ xpub_ptr: *const c_uchar, tx_ptr: *const c_uchar, tx_sz: usize, sig_ptr: *const
    ↪ c_uchar) -> i32 {
        let cfg_bytes : Vec<u8> = unsafe { read_data(cfg_ptr, cfg_size) };
        let cfg_str = String::from_utf8(cfg_bytes).unwrap();
        let cfg : Config = serde_json::from_str(cfg_str.as_str()).unwrap();
        let xpub = unsafe { read_xpub(xpub_ptr) };
        let signature = unsafe { read_signature(sig_ptr) };

        let tx_bytes = unsafe { read_data(tx_ptr, tx_sz) };
        let tx : tx::Tx =
    ↪ raw_cbor::de::RawCbor::from(&tx_bytes).deserialize().unwrap();

        let txinwitness = tx::TxInWitness::PkWitness(xpub, signature);

        if txinwitness.verify_tx(&cfg, &tx) { 0 } else { -1 }
    }
```

These are called in the wallet's `Tx.js`'s `sign()` and `verify()` functions.

Recommendation

Explicit input validation checks should be added, and the function should return an appropriate error code upon failure.

Status

Icarus is not concerned by this as this function is not being used at all by Icarus front end. They instead use the `Wallet.js`'s `spend()` function, which does provide check and report error via JSON output.

(Severity was downgraded from Medium to Informational after this feedback.)

3 Observations

This section reports various observations that are not security issues to be fixed.

Observations are codenamed IC-XXX-O-NNN with IC for Icarus, where

- XXX can be CRY (Cryptoxide), CAR (Cardano), CJW (Cardano JS wasm), WFE (wallet front-end), WBE (wallet back-end).
- NNN is a counter in 001, 002, etc.

3.1 IC-CRY-O-001: Zeroization of memory holding sensitive data

For extra safety, the crypto library could erase sensitive data (such as secret keys) from heap or stack memory after being used.

Status

Addressed in `https://github.com/input-output-hk/rust-cardano/issues/192`.

3.2 IC-CAR-O-002: Potentially unsafe `unwrap()` calls

The code includes many `unwrap()` calls to extract a valid result from a `Result`, but will panic upon an `Err` value.

IC-CAR-F-003 is an example of security issue whose root cause is an `unwrap()`.

Status

Addressed in https://github.com/input-output-hk/rust-cardano/issues/193.

3.3 IC-CAR-O-003: `Wallet` seed derivation method does not check mnemonic length

`Wallet:from_bip39_mnemonics()` does not check the number of words nor the validity of the words in `mnemonic_phrase`. The methods called in this function do not perform any check either. However this function does not seem to be used in the current version of the wallet.

Status

Mnemonic length is actually checked, `MnemonicString` can only be constructed with a valid mnemonic phrase of valid length. If the length is not valid, the type cannot be constructed.

3.4 IC-CAR-O-004: Lack of unit tests of BIP44 logic

Neither `bip/bip44.rs` nor `wallet/bip44.rs` include unit tests, which seem necessary unless done elsewhere.

Status

Addressed in https://github.com/input-output-hk/rust-cardano/issues/194.

3.5 IC-CAR-O-005: Clippy warnings

The Clippy linter reports tons of possible improvements, none of which seems related to a security issue.

Status

Clippy warnings and compilation warnings will be addressed over time.

3.6 IC-CAR-O-006: No explicit handling of failed decryption

In `Input::get_derivation_path()`, `HDKey::decrypt_path()` is called:

```rust
pub fn get_derivation_path(&self, key: &hdpayload::HDKey) -> Option<hdpayload::Path> {
    match &self.value.address.attributes.derivation_path {
        &Some(ref payload) => { key.decrypt_path(payload) },
        &None              => { None }
    }
}
```

However `decrypt_path()` will return `None` if decryption failed, in particular if the authentication tag is not validated:

```rust
pub fn decrypt_path(&self, payload: &HDAddressPayload) -> Option<Path> {
    let out = self.decrypt(payload.as_ref())?;
    Path::from_cbor(&out).ok()
}
```

That is, `derivation_path()` returns `None` both when `&self.value.address.attributes.derivation_path` is `None` and when decryption failed.

We suggest to return a `Result` rather than an `Option` and return the relevant error in case of a failure.

Status

Addressed in https://github.com/input-output-hk/rust-cardano/issues/195.

3.7 IC-CAR-O-007: Potential risk of integer overflow

In

```rust
pub fn sum_coins(coins: &[Coin]) -> Result<Coin> {
    coins.iter().fold(Coin::new(0), |acc, ref c| acc.and_then(|v| v + *c))
}
```

The sum could overflow the `u64` type in extrme cases, for example if summing hundreds of coins whose value is close to `MAX_COIN`, which is unlikely to happen. This edge case may be covered nonetheless and the overflow prevented.

Likewise, when `Coin` values are added, the result is not checked to be below `MAX_COIN`.

Status

There is a check on the `Add` trait implementation of the coin. This is why the `acc` in the `fold` is used with `acc.and_then`, which unwraps the result and checks that it is `Result::Ok` before applying the lambda function (which perform an addition and returns a `Result`).

- `Coin::new()` performs a check and returns a `Result<Coin>`
- `Add::<Coin>::add()` performs the check that two given coins can add up and will be checked with reusing the `Coin::new(lhs.0 + rhs.0)`

- `Coin::MAX_COIN` is 45000000000000000. `Coin::MAX_COIN * 2 < u64::MAX`
- All the inputs of the `sum()` function are already valid `Coin` and are below the `Coin::MAX_COIN`.

3.8 IC-WFE-O-008: No certificate/key pinning

Requests to `backendUrl` via `axios` will presumably be done to an HTTPS host for production (cf. the placeholder `"backendUrl": "https://DEFINE:443"` in `production.json`). However no certificate pinning is enforced to ensure that the client talks to the right host, and the client will rely on the browser's certificate store. This is not really a security issue, but could be an extra layer of defense if doable.

3.9 IC-WFE-O-009: Dependencies

When attempting to build the project, `npm` reports that some dependencies include vulnerable ones:

```
added 599 packages from 621 contributors and audited 6374 packages in 20s
found 40 vulnerabilities (6 low, 30 moderate, 4 high)
run ‘npm audit fix‘ to fix them, or ‘npm audit‘ for details
```

(These vulnerabilities do not necessarily apply to the project, however, as it may not use the vulnerable components from these libraries, or may not expose an exploitation vector.)

Furthermore, `npm dview` reports that a number of dependencies are not up-to-date:

```
    Module Name                  Requested          Remote

    autoprefixer                 7.2.5              9.1.0
    bignumber.js                 4.0.0              7.2.1
    bluebird                     3.3.4              3.5.1
```

`cbor`	`4.0.0`	`4.1.1`
`classnames`	`2.1.3`	`2.2.6`
`humanize-duration`	`3.12.0`	`3.15.1`
`mobx`	`3.1.7`	`5.0.3`
`mobx-react`	`4.1.5`	`5.2.3`
`mobx-react-form`	`1.32.2`	`1.35.1`
`mobx-react-router`	`3.1.2`	`4.0.4`
`moment`	`2.21.0`	`2.22.2`
`prop-types`	`15.6.1`	`15.6.2`
`react`	`15.4.2`	`16.4.2`
`react-copy-to-clipboard`	`4.2.3`	`5.0.1`
`react-dock`	`0.2.3`	`0.2.4`
`react-dom`	`15.4.2`	`16.4.2`
`react-intl`	`2.2.3`	`2.4.0`
`react-markdown`	`2.5.0`	`3.4.1`
`react-number-format`	`^3.3.0`	`3.5.1`
`react-polymorph`	`0.6.5`	`0.7.1`
`react-router`	`3.0.3`	`4.3.1`
`react-svg-inline`	`^2.1.0`	`2.1.1`
`safe-buffer`	`5.1.1`	`5.1.2`
`validator`	`6.3.0`	`10.5.0`

3.10 IC-WBE-O-010: API input validation

Address and transaction validation, as done by `validateAddresReq()` and `validateSignedTransactionReq()`, do not make a proper validation of their inputs, as marked as `TODO` in the code.

We suggest that address validation be done using regexes (unless a simpler complete check is possible).

We suggest to verify that a transaction, i.e. each (sub)field of it, may be checked for syntactic correctness and limit values (for example, a number's range). The signature may or may not be verified at this point: if it is, it allows to process invalid requests faster by avoiding a network round-trip; if the verification is not done, it saves in theory the time of one signature verification for valid transactions. But this saving will be negligible, given the speed of the signature verification algorithm.

3.11 IC-CJW-O-011: Paper wallet incorrect documentation

The comment "`* @param iv: Uint8Array - 4 random bytes of entropy`" defines the `iv` parameter of paper wallet's `scrambleStrings()`, but the underlying functions actually need an IV of 8 bytes (`IV_SIZE`), as enforced in `scramble()`.

The comment should be fixed accordingly.

4 About

Kudelski Security is an innovative, independent Swiss provider of tailored cyber and media security solutions to enterprises and public sector institutions. Our team of security experts delivers end-to-end consulting, technology, managed services, and threat intelligence to help organizations build and run successful security programs. Our global reach and cyber solutions focus is reinforced by key international partnerships.

Kudelski Security is a division of Kudelski Group. For more information, please visit https://www.kudelskisecurity.com.

Kudelski Security
route de Genève, 22-24
1033 Cheseaux-sur-Lausanne
Switzerland