Audit of POA's HBBFT Implementation

Jean-Philippe Aumasson

Version 20/12/18

1 Summary

The POA Network project implemented the *Honey Badger of BFT Protocols* (HBBFT), as defined by Miller et al. (after a small fix). HBBFT is essentially an asynchronous version of the well-known Byzantine Fault Tolerant (BFT) class of protocols, created thanks to an atomic broadcast protocol achieving relately low communication complexity, compared to previous similar protocols.

POA's implementation is in Rust and implements HBBFT from scratch—except for dependencies realizing basic operations. It relies on a threshold cryptography library already written by POA, and which we audited in a previous engagement. The goal of the present project is to identify any security shortcoming in this implementation, with a focus on logic bugs and cryptographic issues. We also attempted to match the implementation with the logic defined in the specification. However, this project is not an assessment of HBBFT's security (already analyzed in detail in the original paper.)

(We'll note that HBBFT's assumptions, though more realistic than classical BFT's timing assumptions, still assume that a secure channel is established between all pairs of nodes and that all messages are delivered.)

We considered version `b3c6377` (Nov 8) of the hbbft repository, and reviewed individual components through three phases according to a bottom-up approach:

1. Synchronous key generation, threshold signature and decryption operations
2. Binary agreement, asynchronous broadcast, and reliable broadcast
3. Honey badger (basic and queued version, and parts of the "dynamic" version)

The points reviewed include:

- Randomness quality and entropy sources
- Risky Rust code patterns (unsafe, unwraps, etc.)
- Parsing of untrusted input (e.g. Part and Ack messages)
- Behavior on invalid input (e.g. number of shares)
- Dependencies' safety
- Usage of threshold cryptography API
- General matching of the intended behavior from specifications
- Replay of messages
- Sensitivity to spoofed time values
- Potentially weak parameters
- General software bugs such as integer overflow, division by zero, and so on

The section below describe the issues discovered, in perceived order of severity (starting with the highest severity), and proposes mitigations.

The audit was carried out in 19.5 (including reporting and patches review), and consisted in documentation review, manual code audit, as well as dynamic analysis based on the test suite.

2 Findings

Below we list potential security issues discovered. Our assessment of the potential impact is based on our (limited) understanding of the project, so we may underestimate or overestimate a finding's impact. Moreover, a limited time audit is unlikely to discover all possible issues, so we may have missed some things. We're nonetheless confident that the code is free of blatant parsing or cryptographic bugs, and that any issue missed would likely be related to edge cases of the protocol logic and of its state machine. To find such bugs, formal methods are sometimes useful, but require time and capabilities unavailable for the present audit.

Note that we didn't report the deprecated version of `rand`, as it is evidently known to the authors, and not straightforward to update because of the new API (migration to 0.6 is discussed in https://github.com/poanetwork/hbbft/pull/368).

2.1 Secret values not all erased from memory

2.1.1 Description

Most secret values are zeroized in memory when going out of scope, via the `ContainsSecret` trait, for example for `SecretKey` objects.

However, other intermediate values may be sensitive and not erased. In our understanding, in threshold crypto decryption, the variable `g` in `decrypt()` will not be zeroized, yet it contains a secret value:

```
pub fn decrypt(&self, ct: &Ciphertext) -> Option<Vec<u8>> {
    if !ct.verify() {
        return None;
    }
    let Ciphertext(ref u, ref v, _) = *ct;
    let g = u.into_affine().mul(*self.0);
    Some(xor_with_hash(g, v))
}
```

We have not identified other potential leaks, but recommend that developers have another look.

2.1.2 Status

This issue was added to https://github.com/poanetwork/threshold_crypto/issues/24.

2.2 Potentially unsafe PRNG (ISAAC)

2.2.1 Description

The issue seems known to developers (see comment in the snippet below, from util.rs), yet we believe it should be reported here: the ISAAC PRNG (a variant of RC4) is used in Honey Badger, although ISAAC may not provide strong crypto graphic safety:

```
fn sub_rng(&mut self) -> Box<dyn rand::Rng + Send + Sync> {
    // Currently hard-coded to be an `Isaac64Rng`, until better options emerge. This is either
    // dependant on `rand` 0.5 support or an API re-design of parts of `threshold_crypto` and
    // `hbbft`.
    let rng = self.gen::<rand::isaac::Isaac64Rng>();
    Box::new(rng)
}
```

2.2.2 Status

Related improvements are proposed in https://github.com/poanetwork/hbbft/pull/357 and in https://github.com/poanetwork/hbbft/issues/356.

2.3 Potential panic upon usize underflow in `num_correct()`

2.3.1 Description

The method `num_correct()` of `NetworkInfo` would be safer if it verified that `num_faulty` is smaller than `num_nodes` (otherwise it would underflow the `usize` and panic at runtime):

```
pub fn num_correct(&self) -> usize {
    self.num_nodes - self.num_faulty
}
```

In the current version of the code, `num_faulty` is set to `(num_nodes - 1) / 3`, which eliminates the risk.

2.3.2 Status

An additional assert statement was added for extra safety, in https://github.com/poanetwork/hbbft/pull/364.

2.4 Binary agreement: coin value per epoch computed differently than in specs

2.4.1 Description

In our understanding, Figure 11 in the HBBFT paper does not fully match the logic implemented in binary_agreement/mod.rs, specifically regarding the following computation of a new epoch's coin value, as commented in mod.rs:

```
//! In epochs that are 0 modulo 3, the value `s` is `true`. In 1 modulo 3, it is `false`. In the
//! case 2 modulo 3, we flip a coin to determine a pseudorandom `s`.
```

It seems that instead the specification sets the coin to a random value (modulo 2). The implications of this discrepancy—if correct—are unclear to us.

2.4.2 Status

This is a documented deviation of the original protocol in order to improve performance, as described and analyzed by Andrew Miller in https://github.com/amiller/HoneyBadgerBFT/issues/63.

2.5 Inconsistent epoch type (`u64` vs `u32`)

2.5.1 Description

The epoch is generally 64-bit (`u64`) but 32-bit (`u32`) in the binary agreement protocol. Since these variable refer to the same value in the protocol, we recommend that it be 64-bit everywhere.

2.5.2 Status

This issue is no longer relevant for the latest version in the master branch.

2.6 Use of `unwrap()` and potential panics

2.6.1 Description

The use of `unwrap()` may lead to a panic if it fails to process a `Some` or `Ok` object, and is use in a number of places including threshold signature, key generation, binary agreement, and broadcast.

We could not determine an attack vector that would trigger a panic, however it is generally recommended to manually handle failures through pattern matching.

2.6.2 Status

The PR https://github.com/poanetwork/hbbft/pull/362

2.7 Encryption schedule allows unencrypted rounds in HB

2.7.1 Description

The `EncryptionSchedule` parameter in HB allows to only encrypt certain epochs or not to encrypt at all. The rationale behind this parameter are unclear to us (performance?), but in production one should ensure that the highest security parameter is set.

2.7.2 Status

This modification was done on purpose, based on a proposal of Andrew Miller (see https://github.com/amiller/HoneyBadgerBFT/issues/9), to improve performance. The implications have not been formally analyzed but seem well understood.

2.8 Consensus-node example panic

2.8.1 Description

Not a security issue, but something that should be fixed in the example program in examples/consensus-node.rs:

```
$ RUST_BACKTRACE=1 ./target/debug/examples/consensus-node -h
thread 'main' panicked at 'called `Result::unwrap()` on an `Err` value: AddrParseError(())',
libcore/result.rs:1009:5
stack backtrace:
   0: std::sys::unix::backtrace::tracing::imp::unwind_backtrace
             at libstd/sys/unix/backtrace/tracing/gcc_s.rs:49
   1: std::sys_common::backtrace::print
             at libstd/sys_common/backtrace.rs:71
             at libstd/sys_common/backtrace.rs:59

(...)

  15: std::sys_common::bytestring::debug_fmt_bytestring
             at libstd/panicking.rs:289
             at libstd/panic.rs:392
             at libstd/rt.rs:58
  16: std::rt::lang_start
```

```
           at libstd/rt.rs:74
17: <consensus_node::Args as core::fmt::Debug>::fmt
```

2.8.2 Status

Fixed in https://github.com/poanetwork/hbbft/pull/363.