Protocol Labs

# Filecoin

## Forest Security Assessment

*Version: 1.0*

# Contents

# Introduction

**Protocol Labs** is a research, development, and deployment institution for improving Internet technology. Protocol Labs leads groundbreaking internet projects, such as *IPFS*, a decentralized web protocol; and *libp2p*, a modular network stack for peer-to-peer applications.

**Filecoin** is an open source project led by Protocol Labs which aims at providing a decentralized storage network that turns cloud storage into an algorithmic market. Miners earn the native protocol token by providing data storage and/or retrieval.

**Forest** is an implementation of Filecoin written in Rust, developed and maintained by **ChainSafe**.

Sigma Prime was commercially engaged by Protocol Labs to perform a time-boxed security review of Forest. The review focused solely on the security aspects of the code base within scope, though general recommendations and informational comments are also provided.

## Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the protocol or its implementation. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

## Document Structure

The first section provides an overview of the functionality of the Forest client contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities found within the code.

## Overview

Forest is an implementation of Filecoin, written in Rust and built in two parts:

- **Part I:** Filecoin's security critical systems in Rust from the Filecoin Protocol Specification, namely the virtual machine, blockchain, and node system.
- **Part II:** Functional components for storage mining and storage & retrieval markets to compose a fully functional Filecoin node implementation.

Currently, Forest leverages the Lotus Storage Miner process for block production duties, including computing PoSt and PoReps.

# Security Assessment Summary

This review was conducted on the files hosted on the forest repository and were assesed at commit 84ab31b.

Fuzzing activities leveraging libFuzzer and honggfuzz have been performed by the testing team in order to identify panics within the scope of this assessment. These fuzzing engines are coverage-guided tools which explore different code paths by mutating input to reach as many code paths as possible. The aim is to find memory leaks, overflows, index out of bounds or any other panics.

Along with fuzzing activities, the testing team performed tailored and targeted testing to identify some of the vulnerabilities raised in this report.

## Findings Summary

The testing team identified a total of 21 issues during this assessment. Categorized by their severity:

- Critical: 7 issues.
- High: 2 issues.
- Medium: 6 issues.
- Low: 1 issue.
- Informational: 5 issues.

# Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the scope of this assessment. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the code base, including comments not directly related to the security posture, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

# Summary of Findings

| ID | Description | Severity | Status |
|---|---|---|---|
| FOR-01 | Denial-of-Service via Panic in Block Validation | Critical | Resolved |
| FOR-02 | Inconsistent Deserialisation of `Address` | Critical | Resolved |
| FOR-03 | Inconsistent Deserialisation of `Randomness` | Critical | Open |
| FOR-04 | Inconsistent Code Paths in the Multisig Actor | Critical | Resolved |
| FOR-05 | Inconsistent `ConsensusMinerMinPower` Values to Lotus | Critical | Open |
| FOR-06 | Panics in `indexmap` and `hashbrown` Libraries | Critical | Open |
| FOR-07 | Panic when Unmarshalling String Data Types | Critical | Open |
| FOR-08 | Disabled Gossipsub Scoring Parameters | High | Resolved |
| FOR-09 | Inconsistent Beacon Entry Validation Across Forks | High | Resolved |
| FOR-10 | Inconsistent `CONSENSUS_MINER_MIN_MINERS` Constant Value to Lotus | Medium | Resolved |
| FOR-11 | Incorrect `message_id` in Gossipsub | Medium | Resolved |
| FOR-12 | No Validation Of Genesis And Initial Drand Beacons | Medium | Closed |
| FOR-13 | Potential Reentrancy in the Multisig Actor | Medium | Open |
| FOR-14 | Unencrypted Private Keys | Medium | Resolved |
| FOR-15 | Inaccurate Floating Point Comparison | Medium | Resolved |
| FOR-16 | Unnecessary Extensive Permissions for Private Keys | Low | Resolved |
| FOR-17 | Inconsistent Order of Operations in `apply_block_messages()` | Informational | Open |
| FOR-18 | Check Not Present in Forest | Informational | Open |
| FOR-19 | Superfluous Check in `deal_proposal_is_internally_valid()` | Informational | Open |
| FOR-20 | Potential For Overwrite During Recursion in VM State Transactions | Informational | Open |
| FOR-21 | Miscellaneous General Issues | Informational | Open |

| FOR-01 | Denial-of-Service via Panic in Block Validation | | |
|---|---|---|---|
| Asset | `blockchain/blocks/src/header/mod.rs` | | |
| Status | **Resolved:** See Resolution | | |
| Rating | Severity: Critical | Impact: High | Likelihood: High |

## Description

Certain blocks can cause Forest to panic in `BlockHeader.validate_block_drand()`. A malicious peer can exploit this to repeatedly crash syncing Forest nodes, performing a denial of service (DoS).

Though this crash can be easily triggered by an attacker, it is a relatively simple condition so may even be triggered by a non-malicious peer with an incorrect or buggy `BeaconSchedule`.

The particular crash occurs at line [402], when the block's `beacon_entries` field is empty, but some are expected given the epoch and the `BeaconSchedule`. There is no check earlier in `validate_block_drand()` prior to line [402] that ensures some `beacon_entries` must be present. As such, the `unwrap()` may raise a panic.

```
let max_round = curr_beacon.max_beacon_round_for_epoch(self.epoch);
392 if max_round == prev_entry.round() {
      if !self.beacon_entries.is_empty() {
394     return Err(Error::Validation(format!(
        "expected not to have any beacon entries in this block, got: {:?}",
396     self.beacon_entries.len()
        )));
398   }
      return Ok(());
400 }

402 let last = self.beacon_entries.last().unwrap();
```

Because the panic is not caught or recovered from (this may not be reliable in Rust), the entire Forest process crashes. This crash doesn't result in any peer blacklisting or negative scoring, so the attack may be repeated by the same peer without immediate consequence. Also, because new peers are prioritised somewhat over existing ones, a malicious peer may be able to affect a large number of nodes.

## Recommendations

Appropriately handle the scenario in `validate_block_drand()` where the block contains no `beacon_entries` but some are expected – returning an appropriate error instead of panicking.

Add a relevant unit test case that exercises this.

Consider adding CI checks for panicking functions like `unwrap()` and `expect()`; requiring that, whenever they are used, each has a corresponding comment explaining why it should be safe.

### Resolution

This issue has been mitigated in PR #1188, by changing the `unwrap()` to be a match-statement as seen below.

```
let last = match self.beacon_entries.last() {
    Some(last) => last,
    None => {
        return Err(Error::Validation(
            "Block must include at least 1 beacon entry".to_string(),
        ));
    }
};
```

| FOR-02 | Inconsistent Deserialisation of `Address` | | |
|---|---|---|---|
| Asset | `forest/vm/address/src/lib.rs` | | |
| Status | **Resolved:** See Resolution | | |
| Rating | Severity: Critical | Impact: High | Likelihood: High |

## Description

The object `Address` is used in Filecoin to represent user accounts. Each `Address` can be one of four types; `ID`, `Secp256k1`, `BLS` or `Actor`.

Addresses of type `ID` represent a `u64` in Rust (`uint64` in Golang). They are serialised using Little Endian Base 128 encoding (LEB128), which can represent arbitrarily large integers as bytes.

The LEB128 implementations used in Lotus and Forest handle edge cases for the encoding differently.

The first discrepancy occurs if extra bytes are padded on to the end. For example consider the following bytes of data, which represent a CBOR encoded address, `[67, 0, 1, 2]`. The first byte is a CBOR tag, the second byte specificies the address type `ID` and remaining bytes (`[1, 2]`).

In LEB128 when a byte with value less than 128 is read this signals the end of the input. Thus, when the number one (1) appears no more input is read. Hence, in the above example an additional byte (2) has been appended. Forest will successfully deserialise this as the address `t01`, whereas Lotus will return the error `"different varint length (v:1 != p:2): invalid address payload"`.

The second discrepancy occurs if the data is not minimally encoded. Using the following byte array as a CBOR encoded address, `[67, 0, 130, 0]`. Forest will successfully deserialise to the address `t02` whereas Lotus will return the following error `"could not decode: varint not minimally encoded: invalid address payload"`. The error occurs as the LEB128 data (`[130, 0]`) which may represent the number two should instead be represented as `[2]`.

The impact of these issues are consensus bugs as one client would result in an error when deserialising while the other would successfully process the block or transaction.

## Recommendations

Consider pushing changes upstream which add a `read_exact()` method that will error if any additional bytes are passed to the function but not read. Furthermore, add a check to ensure the final byte is non-zero in the case where the data length is strictly greater than one.

Alternatively, after decoding the `ID`, serialise it in LEB128 and verify that the serialised bytes exactly match the original bytes. This will ensure minimality of the encoding and verify the bytes length.

### Resolution

The development team have resolved this issue by reserialising the LEB128 value after deserialisation, then ensuring the original bytes match the reserialised bytes. The resolution can be seen in PR #1149.

| FOR-03 | Inconsistent Deserialisation of `Randomness` | | |
|---|---|---|---|
| Asset | `forest/vm/actor/src/builtin/miner/` | | |
| Status | **Open** | | |
| Rating | Severity: Critical | Impact: High | Likelihood: High |

## Description

The `Randomness` type, used by the Miner Actor in `SubmitWindowedPoStParams`, is `[u8; 32]` in Forest and `[]byte` in Lotus. This causes a consensus bug as Forest will error on deserialisation if `Randomness` is not exactly 32 bytes, whereas Lotus will allow unmarshalling of arrays of all length.

The impact is that Forest may run a `SubmitWindowedPoStParams` transaction with exit code `ErrSerialization` while Lotus will produce the exit code `ErrIllegalArgument` failing one of the following checks in `specs-actors/actors/builtin/miner/miner_actor.go`.

```
if len(params.ChainCommitRand) > abi.RandomnessLength {
  rt.Abortf(exitcode.ErrIllegalArgument, "expected at most %d bytes of randomness, got %d", abi.RandomnessLength
          , len(params.ChainCommitRand))
}
```

```
if !bytes.Equal(commRand, params.ChainCommitRand) {
  rt.Abortf(exitcode.ErrIllegalArgument, "post commit randomness mismatched")
}
```

## Recommendations

One option to resolve the issue is to convert the Forest `Randomness` type to be a `Vector` rather than fixed size array and then performing the equivalent checks in Lotus.

An alternate option is to add checks to Lotus to ensure the `Randomness` is exactly 32 bytes. This could be done by changing the type to a fixed sized array or by adding length checks when performing unmarshalling.

| FOR-04 | Inconsistent Code Paths in the Multisig Actor | | |
|---|---|---|---|
| Asset | `forest/vm/actor/src/builtin/multisig/mod.rs` | | |
| Status | **Resolved:** See Resolution | | |
| Rating | Severity: Critical | Impact: High | Likelihood: High |

## Description

The Multisig Actor has a set of signers. Each signer can approve a transaction. If more than the `threshold` number of signers approve a transaction, it is executed.

The actor provides the function `remove_signer()` which removes a user from the list of allowed signers. There is a check to ensure if we remove a signer there is still sufficiently many signers to reach the threshold. In Lotus `specs-actors/actors/builtin/multisig/multisig_actor.go` the check is as follows.

```
if !params.Decrease && uint64(len(st.Signers)-1) < st.NumApprovalsThreshold {
```

The equivalent check in Forest is the following line.

```
if !params.decrease && st.signers.len() < st.num_approvals_threshold {
```

The difference is in checking the length of signers. Lotus subtracts one (1) from the length as we have not yet removed a signer from the state. This subtraction does not occur in Forest and thus would allow the case where there is less signers than the threshold.

The impact of this would be a consensus bug as the client would take different code paths: one resulting in an error and the the other a successful execution. The successful execution in Forest would create a state where there are less signers than the threshold so the actor could no longer execute transactions.

## Recommendations

We recommend updating the Forest code to subtract one (1) from the length of signers when performing the check.

## Resolution

The recommendation has been implemented in PR #1152, where one (1) is subtracted from the length of signers when performing the check.

| FOR-05 | Inconsistent `ConsensusMinerMinPower` Values to Lotus | | |
|---|---|---|---|
| Asset | `forest/vm/actor/src/builtin/sector.rs` | | |
| Status | **Open** | | |
| Rating | Severity: Critical | Impact: High | Likelihood: High |

## Description

The function `consensus_miner_min_power()` returns the minimum miner power in units of disk space based on the proof size.

Forest returns a constant value for all proof sizes as 10 TiB ( `10u8 « 40` ). In contrast Lotus would return varying values for each proof size as follows.

- `StackedDRGWindow2KiBV1` : 0 B
- `StackedDRGWindow8MiBV1` : 16 MiB
- `StackedDRGWindow512MiBV1` : 1 GiB
- `StackedDRGWindow32GiBV1` : 10 TiB
- `StackedDRGWindow64GiBV1` : 20 TiB

This would potentially cause a consensus bug if a miner were to submit a `StackedDRGWindow64GiBV1` proof when they had more than 10 TiB of power but less than 20 TiB.

Furthermore, in Forest the function will also return 10 TiB for `StackedDRGWinning` proofs, which errors in Lotus when calling `ConsensusMinerMinPower()` in `specs-actors/actors/builtin/sector.go`. Again, this could result in a consensus fault if a user attempted to create an Miner Actor with a `StackedDRGWinning` proof rather than `StackedDRGWindow` proof.

Similarly, the function `RegisteredPoStProof::window_post_partitions_sector()` will successfully return values for Winning PoSt proofs in Forest but will error in Lotus when calling the function `PoStProofWindowPoStPartitionSectors()` in `specs-actors/actors/builtin/sector.go`.

## Recommendations

This issue related to differing constants may be resolved by updating the constants in either implementation so that they match.

Consider also erroring for Winning PoSt proofs when calling `consensus_miner_min_power()` and `RegisteredPoStProof::window_post_partitions_sector()` in Forest.

## Resolution

Lotus was updated to match the constants in Forest in PR #1434.

The issue related to accepting Winning PoSt proofs remains open.

| FOR-06 | Panics in `indexmap` and `hashbrown` Libraries | | |
|---|---|---|---|
| Asset | `hashbrown-v0.9.1`, `indexmap-v1.6.2` | | |
| Status | **Open** | | |
| Rating | Severity: Critical | Impact: High | Likelihood: High |

## Description

Forest utilises the `indexmap`, `hashbrown` and `std` libraries when unmarshalling IPLD data. The `indexmap` library is responsible for generating a compact hash table of a given length and capacity. Given array data to deserialize, Forest will initially make a call to `indexmap/src/serde.rs:visit_map()` line [42], which makes a string of calls to reach `indexmap/src/map/core.rs:with_capacity()` line [131] as shown below.

```
pub(crate) fn with_capacity(n: usize) -> Self {
  IndexMapCore {
      indices: RawTable::with_capacity(n),
      entries: Vec::with_capacity(n),
  }
}
```

However, there are certain limitations in how the `hashbrown` and `std/vec` libraries deal with `RawTable` and `Vec` allocations respectively. As a result, the `std/alloc` library will crash due to OOM if `length` is not less than or equal to `capacity`. Similarly, the `hashbrown` library will OOM if these invariants are not checked or will panic if there is a capacity overflow in the hash table.

```
impl Fallibility {
  /// Error to return on capacity overflow.
  #[cfg_attr(feature = "inline-more", inline)]
  fn capacity_overflow(self) -> TryReserveError {
      match self {
          Fallibility::Fallible => TryReserveError::CapacityOverflow,
          Fallibility::Infallible => panic!("Hash table capacity overflow"),
      }
  }

  /// Error to return on allocation error.
  #[cfg_attr(feature = "inline-more", inline)]
  fn alloc_err(self, layout: Layout) -> TryReserveError {
      match self {
          Fallibility::Fallible => TryReserveError::AllocError { layout },
          Fallibility::Infallible => handle_alloc_error(layout),
      }
  }
}
```

This is caused by the use of the `hashbrown/src/raw/mod.rs:fallible_with_capacity()` line [427] function, which is called by `hashbrown/src/raw/mod.rs:with_capacity()` line [457] where the fallability enum is set to infallible. As a result, the `hashbrown` library will `abort` or `panic` instead of propagating an error back to the `indexmap` library.

However, it is important to note that the two calls, `indexmap/src/map.rs:with_capacity_and_hasher()` line [167] and `indexmap/src/map/core.rs:with_capacity()` line [131], are not implemented to handle errors.

## Recommendations

There are two potential solutions that can be implemented:

- Fork the `indexmap` library and limit the size of `IndexMap`,
- Fork both the `indexmap` and `hashbrown` libraries and implement error handling by using `hashbrown/src/raw/mod.rs:try_with_capacity()` instead of `hashbrown/src/raw/mod.rs:with_capacity()`.

There should also be some assertion that ensures the capacity of `IndexMap` is always greater than the length of the data being provided.

Add a relevant unit test case that exercises this.

Consider adding CI checks for panicking functions like `indexmap/src/map.rs:with_capacity_and_hasher`; requiring that, whenever they are used, each has a corresponding comment explaining why it should be safe.

| FOR-07 | Panic when Unmarshalling String Data Types | | |
|---|---|---|---|
| Asset | `BlockHeader, GossipBlock, TipsetKeys, Address, SignedMessage` | | |
| Status | **Open** | | |
| Rating | Severity: Critical | Impact: High | Likelihood: High |

## Description

Forest uses a fork of a `cbor` library to marshal and unmarshal byte data. Given string data, the `cbor` library makes a call to parse_str() which utilises an `unsafe` function. This unsafe function, convert_str(), converts a slice of bytes to a string slice without checking if the string is valid UTF-8.

Rust `string` types often expect valid UTF-8 and may panic if invalid bytes are used. Consequently, there are functions that will `panic` due to improper unicode char boundaries.

One such example of a panic that can be reached from unmarshalling invalid UTF-8 is the to_string() function which is called in the error handling process.

An example of the panic can be seen when calling `BlockHeader::unmarshal_cbor()` on the following bytes `0x6ea0800000080000000000000ff00000309000000`.

## Recommendations

The issue may be resolved by rolling back the commit 3e7bf81, which will instead error when deserialising non-UTF8 strings.

The rollback requires the Lotus implementation to error when decoding non-UTF8 strings to prevent a consensus bug. Progress on the Lotus client can be tracked by this issue.

| FOR-08 | Disabled Gossipsub Scoring Parameters | | |
|---|---|---|---|
| Asset | `forest/node/forest_libp2p/src/behaviour.rs` | | |
| Status | **Resolved:** See Resolution | | |
| Rating | Severity: High | Impact: High | Likelihood: Medium |

## Description

The Gossipsub protocol [1], due to a number of various security vulnerabilities, was enhanced to version 1.1. This version was specifically designed to mitigate a wide range of attacks that could be performed on Gossipsub v1.0 networks [2].

The protocol mitigates these attack vectors by introducing a scoring system for nodes on the network. The scoring system requires a set of tuned parameters in order to score peers effectively for a specific network.

Forest has disabled all the scoring parameters, effectively leaving their nodes vulnerable to the known attacks present in the Gossipsub v1.0 protocol.

## Recommendations

Include the scoring parameters when initialising Gossipsub. The parameters are specific to the network, and it is recommended to use the same parameters as in the Golang implementation to avoid potential down-scoring of nodes.

## Resolution

Peer scoring has been enabled in PR #1115.

| FOR-09 | Inconsistent Beacon Entry Validation Across Forks | | |
|---|---|---|---|
| Asset | `forest/blockchain/blocks/src/header/mod.rs` | | |
| Status | **Resolved:** See Resolution | | |
| Rating | Severity: High | Impact: High | Likelihood: Medium |

## Description

There is an inconsistency between Forest and Lotus in how Drand beacon entries are validated in the first block of a fork.

This is found in `validate_block_drand()` at lines [376–389], and the corresponding Lotus function `ValidateBlockValues()` (i.e. when `cb_epoch != pb_epoch`).

In Lotus, the function returns after validating that the block contains two beacon entries and that they follow from each other. However, Forest only returns in this section when the validation fails. When the checks pass, it instead continues past line [388] to perform all the normal validation checks.

In the case of a hard-fork change to the `BeaconSchedule` where it's intended that the beacon chain doesn't directly follow from the pre-fork chain, this will cause a consensus split between the implementations, segmenting the network. Lotus would validate these blocks but Forest would not, instead requiring that the new `curr_beacon` `BeaconSchedule` can accept `prev_entry` as input and that the new entries extend from this.

## Recommendations

Adjust `validate_block_drand()` to match Lotus, such that `Ok(())` is returned after successfully passing the checks at lines [378–388].

If, instead, this is a bug in Lotus, ensure that Lotus and the spec are updated for consistency.

## Resolution

`Ok(())` has been added at lines [389–390] in PR #1206 resolving the issue.

| FOR-10 | Inconsistent `CONSENSUS_MINER_MIN_MINERS` Constant Value to Lotus | | |
|---|---|---|---|
| Asset | `forest/vm/actor/src/builtin/power/policy.rs` | | |
| Status | **Resolved:** See Resolution | | |
| Rating | Severity: Medium | Impact: High | Likelihood: Low |

## Description

The constant `CONSENSUS_MINER_MIN_MINERS` in Forest is three (3). The equivalent value in Lotus is four (4).

The impact in the varying constants is that a different path may be taken in the function `current_total_power()` in the Power Actor. The result would be a consensus bug as different values would be used for the calculation of rewards during the epoch tick.

For this code path to be in effect there would need to be exactly three miners in total who have proven more than the min power amount. This makes it very unlikely to be triggered on mainnet as there are currently over 2,000 active miners.

## Recommendations

We recommend updating the constants in either implementation so that they are the same.

## Resolution

Forest has been updated in PR #1153 such that `CONSENSUS_MINER_MIN_MINERS = 4`.

| FOR-11 | Incorrect `message_id` in Gossipsub | | |
|---|---|---|---|
| Asset | `forest/node/forest_libp2p/src/behaviour.rs` | | |
| Status | **Resolved:** See Resolution | | |
| Rating | Severity: Medium | Impact: Medium | Likelihood: Medium |

## Description

The `message_id` function in Gossipsub is not configured correctly. This creates a significant bandwidth cost to the network for nodes communicating amongst each other with differing `message_id` functions, as nodes advertise seen messages via the `message_id` through `IHAVE` messages. Nodes using a different id function will then constantly and repetitively re-request these messages via `IWANT` messages causing many duplicate sends.

This can also lead to penalties in the new scoring system and causing the Forest implementation to be down-scored and in the worst case ignored from the Gossipsub network.

## Recommendations

Match the `message_id` function to the Golang implementation (blake2b hash of the content) to prevent down-scoring and excessive message duplication on the network.

## Resolution

The recommendation has been implemented in PR #1115 by updating `message_id` to use blake2b.

| FOR-12 | No Validation Of Genesis And Initial Drand Beacons | | |
|---|---|---|---|
| Asset | `forest/blockchain/beacon/src/drand.rs` | | |
| Status | **Closed:** See Resolution | | |
| Rating | Severity: Medium | Impact: Medium | Likelihood: Medium |

## Description

***NOTE:*** This issue also affects the current Lotus implementation.

The current `DrandBeacon.verify_entry()` has a placeholder for validation of the first beacon entry in the chain.

```
// TODO: Handle Genesis better
if prev.round() == 0 {
  return Ok(true);
}
```

The impact of this issue has not been exhaustively evaluated by the testing team. It may allow for invalid beacon entries to be accepted, making it impossible for the chain to proceed without a hard-fork (as Drand will not produce correctly signed beacons that have the invalid value as an ancestor).

## Recommendations

Ensure initial Drand beacon entries are correctly validated against the configured genesis beacon. If this value is not present in the genesis block (in the `genesis.car`), it can be obtained from the `ChainInfo` `group_hash`.

Ensure this is also resolved in Lotus, to avoid a consensus split scenario.

A fix could include populating the `chain_store` with this genesis beacon and modifying `chain_store.latest_beacon_entry()` such that it returns this. Then, `verify_entry()` could proceed as normal.

Implement appropriate tests to ensure verification returns an error when the first beacon doesn't follow from the genesis.

## Resolution

The development have opted not to fix this issue as it will not impact the current network.

| FOR-13 | Potential Reentrancy in the Multisig Actor | | |
|---|---|---|---|
| Asset | `forest/vm/actor/src/builtin/multisig/mod.rs` | | |
| Status | **Open** | | |
| Rating | Severity: Medium | Impact: Medium | Likelihood: Medium |

## Description

When the Multisig Actor receives sufficient approvals from signers for a pending transaction, it will execute the pending transaction. Execution of the transaction will run as a subroutine sandboxed from the original Multisig transaction using the VM `send()` function. After processing of the subroutine, execution will resume and the approved transaction will be marked as executed and removed from the pending transaction list to prevent users replaying this transaction.

Updates to the state occur after the subroutine has finished executing. The implications of this are that if a user gains control of the subroutine they may call `approve()` a second time and this will again execute the pending transaction.

For the user to gain control of the subroutine the simplest way would be to call `approve()` on another Multisig Actor which if it passes the threshold will trigger a subroutine controlled by the attacker.

```
if threshold_met {
    // ...
    match rt.send(txn.to, txn.method, txn.params.clone(), txn.value.clone()) {
        Ok(ser) => {
            out = ser;
        }
        Err(e) => {
            code = e.exit_code();
        }
    }
    applied = true;
    rt.transaction(|st: &mut State, rt| {
        let mut ptx = make_map_with_root::<_, Transaction>(&st.pending_txs, rt.store())
            .map_err(|e| {
                e.downcast_default(
                    ExitCode::ErrIllegalState,
                    "failed to load pending transactions",
                )
            })?;
        ptx.delete(&txn_id.key()).map_err(|e| {
            e.downcast_default(
                ExitCode::ErrIllegalState,
                "failed to delete transaction for cleanup",
            )
        })?;
        st.pending_txs = ptx.flush().map_err(|e| {
            e.downcast_default(
                ExitCode::ErrIllegalState,
                "failed to flush pending transactions",
            )
        })?;
    //...
```

The attack is possible due to the order of operations in the function `execute_transaction_if_approved()`. Since the subroutine is run when `send()` is called before the state updates in `rt.transaction()` then pending

transaction will still exist in the state if the subroutine reenters the `approve()` function.

This issue is present in both Lotus and Forest. An attacker would be able to continue the attack until either the VM max depth (256) is reached or an actor has insufficient balance and a subroutine exits early.

## Recommendations

We recommend updating both Lotus and Forest such that they first update the state in removing the pending transaction before calling `send()`.

| FOR-14 | Unencrypted Private Keys | | |
|---|---|---|---|
| Asset | Forest | | |
| Status | **Resolved:** See Resolution | | |
| Rating | Severity: Medium | Impact: Medium | Likelihood: Medium |

## Description

The default configuration file for Forest has the field `encrypt_keystore` set to `false`. This results in the keystore generated at `~/.forest/keystore.json` being unencrypted.

Any user who gains read access over the file will then be able to read the private key and execute unauthorised operations.

## Recommendations

Modify the default configuration to have `encrypted_keystore = true` this will ensure the keystore is password protected against attackers who have gained access to the system. Furthermore, add a warning for users who disable keystore encryption.

## Resolution

The default configuration for the keystore has been updating such that it is now encrypted and a warning is emitted when using an unencrypted keystore. The changes can be seen in PR #1150.

| FOR-15 | Inaccurate Floating Point Comparison | | |
|---|---|---|---|
| Asset | `forest/blockchain/message_pool/src/msg_chain.rs` | | |
| Status | **Resolved:** See Resolution | | |
| Rating | Severity: Medium | Impact: Low | Likelihood: High |

## Description

`approx_cmp()` is used to compare two floating point numbers ( `f64` ) and intends to treat "close" values as equal to account for the floating point system's inherent approximation of real numbers.[1] However, this is done incorrectly such that it may return inaccurate results — treating very large but comparatively close numbers as different, and very small but relatively different numbers as equal.

`f64::EPSILON` is used incorrectly in this case. It is defined to be "the difference between $1.0$. and the next larger representable number"[2], so the current implementation (which uses the value as a measurement of *absolute* error) will only give accurate results when the numbers involved are close to $1.0$.

```
fn approx_cmp(a: f64, b: f64) -> Ordering {
  if (a - b).abs() < std::f64::EPSILON {
    Ordering::Equal
    } else {
    a.partial_cmp(&b).unwrap()
  }
}
```

Instead, it should be used for *relative* error comparisons, like:

```
if (a - b).abs() <= (a * std::f64::EPSILON).abs() {
```

As `approx_cmp()` appears to only be used for message selection, the impact of this inaccuracy is limited, perhaps resulting in the reduced rewards due to the proposal of sub-optimal messages in a block. However, should message ordering need to be consistent across implementations, this could have a critical impact on consensus.

## Recommendations

Adjust `approx_cmp()` to make appropriate use of `f64::EPSILON` as a measurement of relative error.

Implement appropriate tests to verify that this gives expected results.

Alternative fixes can include:

- Use of a well–maintained and rigorous third-party implementation, perhaps approx.
- Restricting the range of values accepted by `approx_cmp()` such that the inaccuracies are within acceptable bounds.

[1] E.g. for floats $0.1 + 0.2 \neq 0.3$
[2] https://doc.rust-lang.org/std/primitive.f64.html#associatedconstant.EPSILON

- Visibly commenting or documenting any limitations to reduce the risk of this code being reused elsewhere, where the impacts would be more substantial.

## Resolution

The issue has been mitigated by multiplying one of the values by `EPSILON`. The result is that the difference is now checked against an amount proportional to the size of the values. The mitigation was implemented in PR #1160.

| FOR-16 | Unnecessary Extensive Permissions for Private Keys | | |
|---|---|---|---|
| Asset | Forest | | |
| Status | **Resolved:** See Resolution | | |
| Rating | Severity: Low | Impact: Medium | Likelihood: Low |

## Description

The `forest` binary generates elliptic curves private/public keypairs as one of the following types BLS381, SECP256k1 or Ed25519.

When the binary is first run it will create the `~/.forest` directory, with the following keypair files:

- `keypair.json`
- `libp2p/keypair`

The `keypair.json` contains either BLS381 (default) or SECP256K1 private keys. The `libp2p/keypair` contains an `Ed25519` private key. Both files are created and stored with unnecessary extensive file permissions. Indeed, these keys are readable by any user of the system (i.e. `-rw-r--r--` or `0644` permission bits).

Note this most likely falls within the threat model of Filecoin and is therefore raised as a low severity. Indeed, if an attacker gains access to the machine running Forest with an unprivileged account, it is considered a full compromise.

## Recommendations

Change permissions on private key file generation to `-rw-------` (i.e. `0600`).

Refer to the following *Pull Request* from a different Proof-of-Stake Blockchain implementation for an example:

`https://github.com/sigp/lighthouse/pull/551`

## Resolution

The permissions have been updated to `0600` in PR #1151.

| FOR-17 | Inconsistent Order of Operations in `apply_block_messages()` |
|---|---|
| Asset | `forest/vm/interpreter/src/vm.rs` |
| Status | Open |
| Rating | Informational |

## Description

The order of operations in the function `apply_block_messages()` differs between Lotus and Forest potentially causing a consensus bug.

The following is an excerpt from Forest.

```
let ret = self.apply_message(msg)?;

if let Some(cb) = &mut callback {
    cb(&cid, msg, &ret)?;
}

// Update totals
gas_reward += &ret.miner_tip;
penalty += &ret.penalty;
receipts.push(ret.msg_receipt);
```

The equivalent code from the Lotus function `stmgr.ApplyBlocks()` is as follows.

```
r, err := vmi.ApplyMessage(ctx, cm)
if err != nil {
  return cid.Undef, cid.Undef, err
}

receipts = append(receipts, &r.MessageReceipt)
gasReward = big.Add(gasReward, r.GasCosts.MinerTip)
penalty = big.Add(penalty, r.GasCosts.MinerPenalty)

if cb != nil {
  if err := cb(cm.Cid(), m, r); err != nil {
    return cid.Undef, cid.Undef, err
  }
}
```

In both cases the variable `cb` stands for callback and is an optional function that can be applied after each message. In the case where the callback function returns an error, Lotus will return after updating `receipts`, `gasReward` and `penalty`. However, Forest will return before updating the equivalent variables.

The issue is raised as informational as during normal block processing the callback function is not set and thus this code path will not be reached.

The callback is only used during the state `replay()` function in `forest/blockchain/state_manager/src/lib.rs`.

## Recommendations

We recommend updating the order of operations in either Forest or Lotus for the affected functions such that they are equivalent.

| FOR-18 | Check Not Present in Forest |
| --- | --- |
| Asset | `forest/vm/interpreter/src/vm.rs` |
| Status | Open |
| Rating | Informational |

## Description

The function `run_cron()` in Forest does not contain a check that is present in Lotus. Specifically, the check to ensure that the exit code is not `ExitCode::Ok`.

This check is superfluous as the case where `ret.act_error` is `Some` but `ret.msg_receipt.exit_code = ExitCode::Ok` should not exist.

## Recommendations

To improve code maintainability as future works are made on the code base, it is recommended to include the check for `ret.msg_receipt.exit_code == ExitCode::Ok` as in Lotus.

| FOR-19 | Superfluous Check in `deal_proposal_is_internally_valid()` |
|---|---|
| Asset | `forest/vm/actor/src/builtin/market/mod.rs` |
| Status | Open |
| Rating | Informational |

## Description

There is an additional check in the Forest function `deal_proposal_is_internally_valid()`. Specifically the check that the deal end epoch is less than or equal to the start epoch as seen below.

```
if proposal.proposal.end_epoch <= proposal.proposal.start_epoch {
    return Err(actor_error!(
        ErrIllegalArgument,
        "proposal end epoch before start epoch"
    ));
}
```

This check does not exist in Lotus' `dealProposalIsInternallyValid()`.

There is no direct security implication of having this additional check as the check appears again (in both implementations) in the function `validate_deal()` / `validateDeal()` which is the sole caller of `deal_proposal_is_internally_valid()` and all exit codes in this function are `ErrIllegalArgument`.

## Recommendations

Potential future vulnerabilities may be mitigated by removing the additional check from Forest in `deal_proposal_is_internally_valid()`.

| FOR-20 | Potential For Overwrite During Recursion in VM State Transactions |
|---|---|
| Asset | `forest/vm/interpreter/src/default_runtime.rs` |
| Status | **Open** |
| Rating | Informational |

## Description

The runtime has the function `transaction()` which is used to make state mutations inside the VM. This function will mutate the current actor based off some closure that is passed as a parameter.

To prevent a second runtime message for executing during a state mutation `allow_internal` is set to `false`.

The following lines of code represent the setting of `allow_internal` and calling of the passed function `f`.

```
self.allow_internal = false;
let r = f(&mut state, self);
self.allow_internal = true;
```

If, during the execution of `f()`, `transaction()` is called a second time `allow_internal` will be set back to `true` at its completion and thus the original `f()` may then make internal `send()` calls during a state mutation.

This issue is raised as informational as no exploitation scenario could be found which users could use to make recursive calls to `transaction()`.

## Recommendations

Consider adding a check to `transaction()` to error if `allow_internal` is already set to `false` in both Forest and Lotus.

| FOR-21 | Miscellaneous General Issues |
|---|---|
| Asset | `forest/` |
| Status | **Open** |
| Rating | Informational |

## Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

- The function `restore_bytes()` in Verified Actor has a different order of operations for loading the `verified_clients` and `verifiers` maps on line [463] and line [471] compared with Lotus.
- Unused constants in `vm/actor/src/builtin/miner/policy.rs`.
  - `SECTORS_MAX`
  - `MAX_PROVE_COMMIT_SIZE_V4`
  - `MAX_PROVE_COMMIT_SIZE_V5`
  - `NEW_SECTORS_PER_PERIOD_MAX`
- In `forest/vm/actor/src/builtin/paych/mod.rs` on line [139] `rt.verify_signature(&sig, &signer, &sv_bz)` `sig` is already a reference and does not need to be referenced again.
- The naming overlap for `RT` in `forest/vm/interpreter/src/default_runtime.rs` in the function `transaction()` to mean "return type" is easily confused with the class name "runtime".
- Incorrect variable used in the error message for the on line [3307] of `forest/vm/actor/src/builtin/miner/mod.rs`. Specifically, `format!("failed to load sector ", params.sector_number)` should use the variable `params.replace_sector_number`. This issue is also replicated in Lotus.
- A typo in the comment at `blockchain/message_pool/src/msgpool/selection.rs:149` — “Parition” should be “Partition”
- A typo in the comment at `blockchain/message_pool/src/msgpool/msg_chain.rs:27` — “slotamap” should be “slotmap”

## Recommendations

Ensure that the above findings are understood and acknowledged, and consider implementing any relevant suggestions.

# Appendix A Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

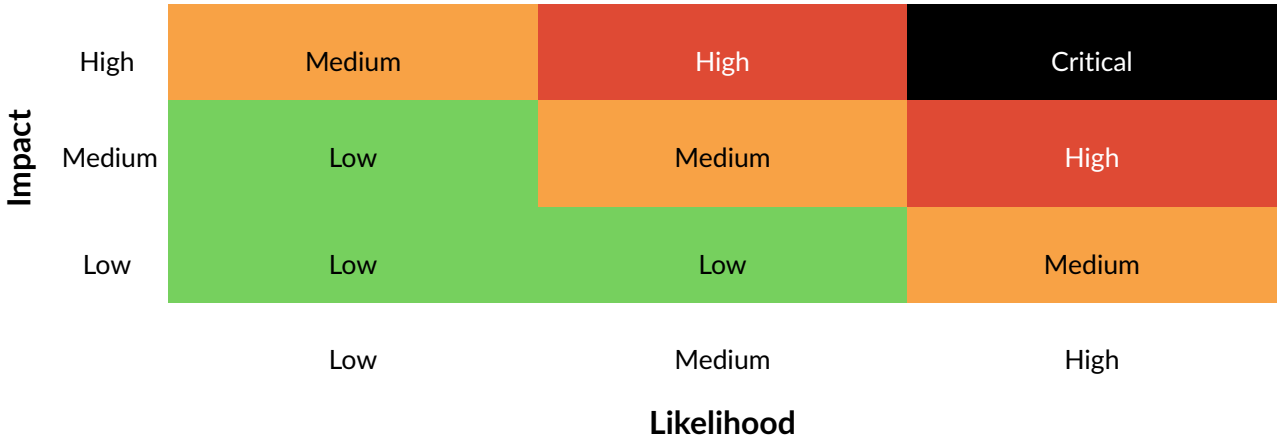


Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

# References

[1] Protocol Labs. gossipsub: An extensible baseline pubsub protocol, Available: https://github.com/libp2p/specs/tree/master/pubsub/gossipsub.

[2] Dimitris Vyzovitis, Yusef Napora, Dirk McCormick, David Dias, and Yiannis Psaras. Gossipsub-v1.1 Evaluation Report. 2020. Available: https://gateway.ipfs.io/ipfs/QmRAFP5DBnvNjdYSbWhEhVRJJDFCLpPyvew5GwCCB4VxM4.