sigma prime

Protocol Labs

Security Assessment

Filecoin Proving Subsystem

Version: 2.1

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.

Sigma Prime was approached by Protocol Labs to perform a security assessment of the Filecoin Proving Subsystem which provides the storage proofs required by the Filecoin protocol. This is implemented in Rust, with an executable specification developed in Golang.

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 smart contract. 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 Filecoin Proving Subsystem 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

At the highest level, Filecoin Proving System aims to implement three types of protocols, Proof of Space-Time (PoST), Proof of Retrievability (PoR) and Proof of Replication (PoRep). These protocols are based upon zk-SNARKS which require a hashing protocol.

zk-SNARKs require an initial setup phase such that all participants may agree on a set of public parameters. These parameters only need to be generated once for each protocol before any proofs may be generated or verified.

Once the public parameters have been agreed upon, proofs may be generated for each of the protocols. PoR proofs verify that a piece of data currently exists in the users storage. PoRep proofs demonstrate that a piece of data has been duplicated and if there are multiple replications each duplication is unique. Finally, PoST proofs validate that a piece of data has indeed been stored for a given period of time.

The Filecoin Proving Subsystem exposes an API where the core functionality can be summarised as follows:

- Generating the initial protocol parameters
- Generating proofs
- Verifying proofs.

Security Review Summary

This review was initially conducted on the following commits:

- `rust-fil-proofs` : e8d4475
- `filecoin-ffi` : 870251c

The second round of this assessment targeted release `v4.0.0` .

Fuzzing activities leveraging libFuzzer have been performed by the testing team in order to identify panics within the code in scope. libFuzzer is a coverage-guided tool which explores different code paths by mutating input to reach as many code paths possible. The aim is to find memory leaks, overflows, index out of bounds or any other panics.

Specifically, the testing team produced the following fuzzing targets:

- `rust-fil-proofs` :
  - `compute_d.rs`
  - `finalize_ticket.rs`
  - `generate_candidates.rs`
  - `generate_candidates_with_conf.rs`
  - `generate_post.rs`
  - `generate_post_with_conf.rs`
  - `generate_then_verify_post.rs`
  - `get_unsealed_range.rs`
  - `seal_commit_phase1.rs`
  - `seal_commit_phase2.rs`
  - `seal_pre_commit_phase1.rs`
  - `seal_pre_commit_phase2.rs`
  - `validate_cache_for_precommit_phase2.rs`
  - `verify_batch_seal.rs`
  - `verify_post_with_conf.rs`
  - `verify_seal.rs`
  - `blake2s_function.rs`
  - `fuzz_blake2b.rs`
  - `fuzz_blake2s.rs`
  - `fuzz_pedersen.rs`
  - `pedersen_function.rs`
  - `poseidon_function.rs`

  - `sha256_function.rs`
- `filecoin-ffi`:
  - `fuzz_fil_write_with_alignment.rs`
  - `fuzz_fil_write_without_alignment.rs`
  - `fuzz_full_cycle.rs`

These fuzzing targets have been shared with the development team.

The testing team identified a total of thirteen (13) issues during the first round of this assessment (`FPS-01` to `FPS-13`), of which:

- One (1) is classified as high risk,
- One (1) is classified as medium risk,
- Eleven (11) are classified as informational.

The testing team identified a total of seven (7) issues during the second round of this assessment (`FPS-14` to `FPS-20`), all of which are classified as informational.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within Filecoin Proving Subsystem. 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 of Filecoin Proving Subsystem, 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
FPS-01	Potentially Dangerous Unwraps on `from_repr()`	High	Resolved
FPS-02	Index Out of Bounds	Medium	Resolved
FPS-03	Failed Assertions #1	Informational	Resolved
FPS-04	Failed Assertions #2	Informational	Resolved
FPS-05	Piece Sizes Overflows	Informational	Resolved
FPS-06	Potential Challenge Count Denial-of-Service	Informational	Resolved
FPS-07	Potential Denial-of-Service on `compute_comm_d`	Informational	Resolved
FPS-08	Merkle Tree Leaves	Informational	Resolved
FPS-09	Merkle Tree Infinite Loop	Informational	Resolved
FPS-10	Configuration Parameters Abuse	Informational	Resolved
FPS-11	Unchecked Stack Accesses	Informational	Resolved
FPS-12	Hash Message Sizes Panics	Informational	Resolved
FPS-13	Miscellaneous General Comments	Informational	Resolved
FPS-14	SDR Spec `feistel()` Infinite Loop	Informational	Resolved
FPS-15	Feistel Permutation Tests Invalid Range	Informational	Resolved
FPS-16	`SectorId::as_fr_safe()` uses 31 Bytes for Field Representatives	Informational	Resolved
FPS-17	SDR Invalid Constants	Informational	Resolved
FPS-18	SDR Spec Edge Cases	Informational	Resolved
FPS-19	Unnecessary `memcopy` in `generate_labels()`	Informational	Resolved
FPS-20	Miscellaneous General Comments - Round 2	Informational	Resolved

FPS-01	Potentially Dangerous Unwraps on `from_repr()`		
Asset	Multiple locations		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

The function `Fr::from_repr()` takes a value as input and converts it to a field value. If the input value is greater than the field modulus then the function will error.

There are numerous occurences of `Fr::from_repr().unwrap()` , `.expect()` or other panic happy accesses of `Results` that can be seen in the codebase. The implication is that, panics will occur if the value is greater than the field modulus. This pattern is a regular occurence with respect to hash function domains.

Additionally, if the `randomness` used in a Winning PoST or Window PoST is greater than the field modulus then both generation and verification will error due to `as_safe_commitment()` . The generation of randomness must ensure that the output is less than the field modulus (similar observation for `prover_id`).

Recommendations

Each occurrence where `from_repr()` is accessed without handling the error case needs to be manually reviewed to ensure it cannot be reached from malicious user input. The cases where it can be reached from malicious user input should instead propogate the error.

Care needs to be taken when generating randomness to ensure that the output is less than the field modulus or proofs cannot be created.

Resolution

The issue has been resolved in the specifications by dictating the type of byte arrays which have been checked as $\mathbb{B}^{[32]}_{safe}$ where `from_repr()` is ensuring the return value is `Result::Ok()` . Alternatively, $\mathbb{B}^{[32]}$ is used for arrays which have not been checked, in these situations the error case in `from_repr()` is handled.

FPS-02	Index Out of Bounds		
Asset	`storage-proofs/src/compound_proofs.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

In the `bellman` dependency, the function `create_proof_batch_priority()` will index out of bounds when `provers` is of length zero, as it is indexed at zero.

The function is called from the function `circuit_proofs()` which is called from `generate_post()`.

Recommendations

We recommend returning an error when `provers` is of length zero.

Resolution

The development team has resolved this issue in commit 8d6c3f2.

FPS-03	Failed Assertions #1
Asset	`storage-proofs/src/circuit/stacked/proof.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

In the function `circuit()` each of the three assertions may be triggered, resulting in a panic.

The assertions may be reached from the function `generate_post()` for varying input. Note that these assertions may be reached when `PoStConfig` configurations are restricted to valid inputs.

The three assertions are:

- `assert!(!vanilla_proof.is_empty(), "Cannot create a circuit with no vanilla proofs");`
- `assert!(vanilla_proof.iter().all(|p| p.comm_r_last() == &comm_r_last));`
- `assert!(vanilla_proof.iter().all(|p| p.comm_c() == &comm_c));`

Recommendations

Consider changing the assertions to the `ensure!` macro such that an error will be returned as opposed to resulting in panics.

Resolution

The development team has resolved this issue in commit 5ac8e2e.

FPS-04	Failed Assertions #2
Asset	`Multiple locations`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The following assertions may fail during execution of `seal_commit_phase2()` :

- `storage-proofs/src/circuit/insertion.rs` : line [22]
- `storage-proofs/src/circuit/insertion.rs` : line [81]
- `storage-proofs/src/por.rs` : line [197]
- `storage-proofs/src/circuit/stacked/hash.rs` : line [41]

Note that these assertions may be reached when `PoRepConfig` configurations are restricted to valid inputs.

Recommendations

Consider changing the assertions to return and propagate errors as opposed to panicking.

Resolution

The desired behaviour is for the program to panic under these conditions. The panics will be caught by the calling repository thus preventing a full program crash.

FPS-05	Piece Sizes Overflows
Asset	`filecoin-proofs/src/api/seal.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

A range of overflows are possible when `piece_info.size` is large, in the functions `seal_pre_commit_phase1()` and `seal_commit_phase1()`.

The following is a list of possible vulnerabilities when the `piece_infos.size` is not restricted:

- `filecoin-proofs/src/pieces.rs`: line [98]:
  Addition overflow if the array of `piece_info` have a summed total size greater than 2^{64}.
- `filecoin-proofs/src/fr32.rs`: line [324]:
  Multiply overflow if `pos * 8` is greater than 2^{64}, where `pos = piece_info.size`.
- `filecoin-proofs/src/fr32.rs`: line [314]:
  Multiply overflow if `full_elements * to_size`, where `full_elements = pos / 254`, `to_size = 256` and `pos = piece_info.size`.
- `filecoin-proofs/src/pieces.rs`: line [235]:
  Additional overflow `left.size + right.size` when the two sizes add to more than 2^{64}, where each `size` is a `piece_info.size`.

Recommendations

Consider returning an error when a piece size is greater than the sector size.

Resolution

Piece lengths are verified by the network service when they are received and thus cannot be larger than the sector size. The max sector size is 64GB and thus `piece_info.size` will not be able to overflow in addition or multiplication.

FPS-06	Potential Challenge Count Denial-of-Service
Asset	`filecoin-proofs/src/api/post.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The function `generate_candidates()` takes the following parameters:

- `post_config`
- `randomness`
- `challenge_count`
- `replicas`
- `prover_id`

If `generate_candidates()` is called with a high `challenge_count`, the time taken to generate the sector challenges will be infeasible for all modern computers.

As a result, there is a denial of service on the client as the resources are tied up attempting to generate an excessive number of sector challenges.

Recommendations

We recommend either:

- Setting a `MAXIMUM_CHALLENGE_COUNT`;
- Restricting access to `generate_candidates()`.

Resolution

The development team is aware of this potential Denial-of-Service vector and have ensured that `challenge_count` is restricted by the network protocol.

FPS-07	Potential Denial-of-Service on `compute_comm_d`
Asset	`filecoin-proofs/src/api/seal.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The function `compute_comm_d()` calls a function of the same name in `filecoin-proofs/src/pieces.rs`. These functions take `sector_size` as input.

Specifying a large `sector_size` will result in `filecoin-proofs/src/pieces.rs` creating a reader of length `sector_size` such that in line [68], `io::copy(...)` will iterate over the entire reader.

If `sector_size` is sufficiently large the copy will iterate over the entire reader which is infeasible for all common machines (consumer hardware), thereby allowing for a Denial-of-Service attack.

Recommendations

We recommend either:

- Converting `SectorSize` to an enum thereby restricting possible values;
- Ensuring `sector_size` is restricted in the calling repository.

Resolution

Piece sizes are restricted to the size of a sector by the network protocol, therefore providing a restriction on the amount of data that can be copied.

FPS-08	Merkle Tree Leaves
Asset	`filecoin-proofs/src/api/post.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Inside the `merkletree` dependency, the function `get_merkle_tree_len(leafs, branches)` will give an invalid answer if `leafs` is not a valid power of branches.

A valid power of branches should be $leafs = branches^x$ for a given x.

Similarly, the function `get_merkle_tree_leafs(len, branches)` will result in a subtraction overflow and the assertion will fail causing a panic if `len` is not a the size of a full tree.

A full tree will have size:

- $len = \sum_{i=0}^{n} branches^i$ for some n.

The combination of these functions are used in `generate_post()` and `generate_candidates()`.

First, `tree_size = get_merkle_tree_len(leafs, branches)` is called with:

- `leafs = post_config.sector_size / Domain::byte_len()`
- `branches = OCT_ARITY = 8`

Second, `leafs = get_merkle_tree_leafs(len, branches)` is called with:

- `len = tree_size`
- `branches = OCT_ARITY = 8`

Thus, if `post_config.sector_size / Domain::byte_len()` is not a valid multiple of 8 the `tree_size` will be invalid and `get_merkle_tree_leafs()` will panic.

Effective exploitation of this bug requires sending an invalid `post_config.sector_size` which is restricted through the use of `enums` in the calling repository `filecoin-project/rust-filecoin-proofs-api`.

Recommendations

To increase the overall robustness of the proof susbsytem, we recommend handling the invalid cases by returning errors for both `get_merkle_tree_len()` and `get_merkle_tree_leaves()` in `merkletree` and handling these errors in `generate_post()` and `generate_candidates()`.

Resolution

The development team has modified `get_merkle_tree_len()` to return a `Result` in this issue in commit a7073ff.

Additionally, `get_merkle_tree_leafs()` has been updated to return a `Result` in commit 9d0f4f2.

FPS-09	Merkle Tree Infinite Loop
Asset	`filecoin-proofs/src/api/post.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Inside the `merkletree` dependency, in the function `get_merkle_tree_leafs(len, branches)`, an infinite loop will occur if `len = 0`.

The function is called from `generate_post()` and `generate_candidates()`.

However, effective exploitation requires `post_config.sector_size = 0` which is restricted through the use of `enums` in the calling repository `filecoin-project/rust-filecoin-proofs-api`.

Recommendations

We recommend handling the zero case by either returning zero or returning an error.

Resolution

The code will now return an error in the loop if `len < leafs` which occurs when `len = 0`. Therefore an infinite loop cannot occur. See commit 9d0f4f2 for more details.

FPS-10	Configuration Parameters Abuse
Asset	`filecoin-proofs/src/api/post.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

A range of overflows and an excessive memory allocations are plausible when `PostConfig` or `PoRepConfig` are able to be set by the user.

The following is a list of possible vulnerabilities when the configuration files are not restricted:

- `storage-proofs/src/circuit/election_post.rs` : line [166]:
  Multiplication overflow in `pub_params.challenged_nodes * pub_params.challenge_count` . Where `challenged_nodes` and `challenged_count` come from `post_config.challenged_nodes` and `post_config.challenged_count` .
- `storage-proofs/src/stacked/graph.rs` : line [125]:
  Multiplication overflow in `expansion_degree * nodes` . Where `expansion_degree` is set to `EXP_DEGREE = 8` and `nodes` comes from post_config.challenged_nodes.
- `storage-proofs/src/circuit/election_posts.rs` : line [171]:
  Capacity overflow if `challenged_nodes * challenge_count` is large then initiaisation of vectors will require excessive memory and will panic.

Recommendations

Care should be taken in the calling repository `filecoin-project/rust-filecoin-proofs-api` to ensure that the configuration parameters cannot be manipulated.

Resolution

The development team has resolved the multiplication overflow in `storage-proofs/src/stacked/graph.rs:125` in commit 47c880d.

The remaining issues will not be fixed as these will be public parameters that are set by the network. In deployment they are set as constant which will not overflow or use excessive memory.

FPS-11	Unchecked Stack Accesses
Asset	`filecoin-proofs/src/pieces.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The struct `Stack` uses an underlying vector to immitate a stack.

The struct is implemented with two funcitons `peek()` and `peek2()` which provide a references to the first and second items in the stack respectively.

However, the functions access the underlying vector without first checking the number of elements.

For example, calling `peek()` on an empty stack will execute `&self.0[self.0.len() - 1]` which will cause a subtraction overflow and the index will be $2^{64} - 1$. The index is out of bounds and will therefore panic.

Similarly, calling `peek2()` on a stack with one element will execute `&self.0[self.0.len() - 2]` thereby accessing the index $2^{64} - 1$. The index is out of bounds and will again panic.

No direct exploitation of this vulnerability could be found based on the current usage of these functions.

Recommendations

Consider changing the function to return an `Option<PieceInfo>` and return `None` when there are insufficient elements.

Resolution

The development team has reviewed all uses of `peek()` and `peek2()` and confirm there are no possible uses resulting in a index out of bounds.

FPS-12	Hash Message Sizes Panics
Asset	`storage-proofs/src/hash/`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Numerous hash implementations, including `SHA256`, `Blake2s`, `Pedersen` and `Poseiodon`, have been implemented. These hash functions take a message as input and output a message digest.

The following implementations of `HashFunction` will panic given a certain message size.

- `PedersenFunction`: if the message is length zero, the iterator in `storage-proofs/src/crypto/pedersen.rs` will fail on the first call to `next()` as it will index out of bounds.
- `PedersenFunction`: if the message length is greater than about 120 bytes, then not enough Pedersen generators would be created.
  Hence, an assertion in the dependency `fil_sapling_crypto` `src/pedersen_has.rs` on line [100] or line [194] will be triggered.
- `PoseidonFunction`: if the message is length zero then the match statement at `storage-proofs/src/hasher/poseidon.rs` on line [250] will panic.
- `PoseidonFunction`: if the message is not 32 bytes nor zero, then `PoseidonDomain::from_slice(data)` will panic.

Additionally, `PoseidonFunction` will panic if the given data is not less than the field modulus as `from_repr()` is unwrapped in `storage-proofs/src/hasher/poseidon.rs` on line [218].

Recommendations

We recommend ensuring all calls to `PedersenFunction::hash()` and `PoseidonFunction::hash()` only pass 32 bytes messages.

The case in `PoseidonFunction` where the `from_repr()` may fail needs to be handled, consider returning a result.

Resolution

All calls to `PedersenFunction::hash()` and `PoseidonFunction::hash()` are made using `FrRepr` checked points and thus will not panic.

FPS-13	Miscellaneous General Comments
Asset	`filecoin-proofs/src/fr32.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

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

- `filecoin-proofs/src/fr32.rs`: consider changing `padded_bytes()` to `to_padded_bytes()` and `unpadded_bytes()` to `to_unpadded_bytes()`

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team has resolved the issue in commit 031b6fb.

FPS-14	SDR Spec `feistel()` Infinite Loop
Asset	SDR Spec
Status	**Resolved:** See Resolution
Rating	Informational

Description

The function `feistel()`, in the SDR Spec section Feistel Network PRP, has a loop that will run indefinitely in 50% of cases.

The loop is deterministic based off the values of $left_r$ and $right_r$ in lines 3 and 4. These values are based off `input` and the constants `RightMask` and `LeftMask`. Hence, $left_r$ and $right_r$ will not change between each iteration of the loop and thus the value of `output` will not change from the first iteration. If the value of `output` does not satisfy the condition of the while loop after the first iteration, execution will continue indefinitely.

Note that this issue does not exist in the Rust implementation.

Recommendations

Either update the value of `input` to match `output` at the end of the loop. Otherwise, use a temporary variable which is initially set to the value of `input` and updated with `output` at the end of each loop iteration.

Resolution

The value of `input` is set to the value of `output` as the last statement in the loop thereby preventing an infinite loop. This can be seen in the updated spec.

FPS-15	Feistel Permutation Tests Invalid Range
Asset	`storage-proofs/core/src/crypto/feistel.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The function `feistel()` requires the output permutation p in the range $p < n$ or $[0, n)$ where n is the size of the permutation.

The Rust tests for `feistel()` in `storage-proofs/core/src/crypto/feistel.rs` in line [128] and line [224] check the output permutation $p \leq n$. Tests may incorrectly pass for the case where $p = n$.

Note that examination of the code shows the case $p = n$ will not occur in the function.

Recommendations

Update the tests to check that p is strictly less than the size of the permutation.

Resolution

This issue has been resolved in the pull request #1196.

FPS-16	`SectorId::as_fr_safe()` uses 31 Bytes for Field Representatives
Asset	`storage-proofs/core/src/sector.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Checked field representatives, $\mathbb{B}_{safe}^{[32]}$, are 32 byte arrays with the most significant two bits in the most significant byte set to 0. On the other hand, `SectorId::as_fr_safe()` outputs a 31 byte array.

Recommendations

The function is not used in `rust-fil-proofs` hence, to avoid accidental programming errors, it may be removed. Alternatively, if the function is required by an external library, consider updating the function to match $\mathbb{B}_{safe}^{[32]}$ by adding a 0 byte as the most significant byte.

Resolution

This issue has been resolved in the pull request #1196.

FPS-17	SDR Invalid Constants
Asset	SRD Notation, Constants, and Types
Status	**Resolved:** See Resolution
Rating	Informational

Description

The constant d_{exp} appears twice (with different values) while d_{drg} does not appear.

The constant $N_{fesitel\_rounds} = 4$ is now 3 Feistel rounds. Note the number four is also used in SDR Spec section Feistel Network PRP. Furthermore, *"fesitel"* is spelt incorrectly in numerous locations in SDR Spec and SDR Notation, Constants, and Types.

Additionally, $FeistelKeys_{PorepID}$ can updated to be an array of length 3. The Rust implementation in `storage-proofs/porep/src/stacked/vanilla/graph.rs` may now also generate 3 keys.

The constants $N_{porep\_partitions}$ and $N_{porep\_challenges}$ are used in SDR Spec but not defined in SDR Notation, Constants and Types.

All constants are defined for the 32GiB sector size. The constants for other sector sizes are have not been included in SRD Notation, Constants, and Types.

Recommendations

Consider the following:

- Change $d_{exp} = 6$ to $d_{drg} = 6$;
- Change $N_{fesitel\_rounds} = 4$ to $N_{feistel\_rounds} = 3$ and fix any occurrences of "fesitel", in both SDR Spec and SDR Notation, Constants, and Types. Additionally, consider using the `FEISTEL_ROUNDS` constant to generate the required number of keys in the rust implementation;
- Add the constants $N_{porep\_partitions}$ and $N_{porep\_challenges}$;
- Add the constants for other sector sizes.

Resolution

The recommendations have been implemented except the constants for the 64 GiB sector size have not been added. The updated spec can be seen here.

FPS-18	SDR Spec Edge Cases
Asset	SRD Spec
Status	**Resolved:** See Resolution
Rating	Informational

Description

There are a range of edge cases that may arise as part of normal code execution, considering normal execution to be the execution paths that occur with a high probability in valid use cases.

For example the cases for `get_drg_parents(v)` where `v = 0 or 1` must be handled differently during the code execution. These cases will always arise during replication and so are part of normal execution.

Recommendations

Consider adding the code paths that may be reached as part of the normal execution of the program.

Resolution

The edges cases for `v = 0 or 1` have been updated in `get_drg_parents(v)` in the most recent spec update.

FPS-19	Unnecessary `memcopy` in `generate_labels()`
Asset	`storage-proofs/porep/src/stacked/vanilla/proof.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The function `generate_labels()` iterates though the layers of the Stacked DRG, creating labels for each node based off their parents in both this layer and the previous one.

The function stores an array of the labels consisting of the current layer and the previous layer as `labels_buffer` of length $n * 2$ where n is the size of a layer. The first n nodes (i.e. $[0, n)$) represent the current layer and the second n (i.e. $[n, 2n)$) nodes represent the previous layer.

At the end of the iteration of each layer there is a `memcopy` of the current nodes to the previous nodes (i.e. the first n nodes are copied over the second n nodes). Subsequently, the next iteration will overwrite the current nodes (i.e. the first n nodes) with the new current layer nodes.

These new current layer nodes overwrite the first n nodes without reading them. Thus, the `memcopy` can be avoided.

It can be avoided by, at the end of the first iteration, switching the previous layer nodes to be the first n nodes and the current layer nodes to be the second n nodes. After the second iteration, switch back such that the first n are the current layer nodes and the second n nodes are the previous layer nodes. This process can be repeated to reduce the need for a `memcopy`.

This is a significant optimisation as a layer will contain a sector size worth of nodes which may be as large as 64GiB.

Recommendations

Consider implementing the optimisation described above to reduce copying large quantities of memory.

Resolution

This issue has been resolved in the pull request #1198.

FPS-20	Miscellaneous General Comments - Round 2
Asset	Multiple
Status	**Resolved:** See Resolution
Rating	Informational

Description

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

- `rust-fil-proofs/README.md` :
  - line [20]: The link `go-filecoin` is `https://github.com/filecoin-project/go-filecoin’` . Consider removing the trailing apostrophe.
  - Build Section: Consider including other dependencies such as `gcc/clang` `wall` , `cmake` .
  - Benchmarking Section: `bencher` binary no longer exists.
  - line [99]: The link `stacked` is broken.
  - line [280]: The link `sector-base` is broken.
  - line [287] & line [288]: The four links on these lines are broken.
- **SDR Notation, Constants, and Types**:
  - General Notation Section: $\mathbb{F}_q$ would be clearer to say q is the curve subgroup order.
  - Protocol Constants Section: `"either Winning of Window PoSt, determined by context"` should be `Winning or Window PoSt` .
  - Protocol Constants Section: `The Groth16 keypair sued to generate ...` should be `The Groth16 keypair used to generate ...` .
  - Protocol Constants Section: The constant $N_{buckets}$ is not needed in SDR Notation, Constants, and Types as it is calculated as a different value in SDR Spec.
- **SDR Spec**:
  - Merkle Proofs Section: The link `storage_proofs::merkle::MerkleTreeWrapper::gen_proof()` is broken.
  - BinTreeProofs Subsection: There is a closing $ missing in, `or the` $BinTreeProof_c.leaf$ `if $1 = 0` .
  - BinTreeProofs Subsection: The function `create_proof()` is referred to as both `BinTreeProof.create_proof()` and `BinTree.create_proof()` .
  - OctTreeProofs Subsection: Similarly, the function `create_proof()` is referred to as both `OctTreeProof.create_proof()` and `OctTree.create_proof()` .
  - DRG Subsection: $dist_{min,b} = max(d_{max,b}/2, 2)$ should be $dist_{min,b} = max(dist_{max,b}/2, 2)$.
  - Expander Subsection: d_E apprears twice, it should be d_{exp} .

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The comments have been understood and acknowledged, and suggestions have been applied where required.

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.

Impact			
High	Medium	High	Critical
Medium	Low	Medium	High
Low	Low	Low	Medium
	Low	Medium	High
		Likelihood	

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

References