Recall Labs

Recall Review

Version: 2.2

June, 2025

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Recall Labs components. The review focused solely on the security aspects of the Solidity implementation of the contract, 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 components in scope. 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 Recall Labs components 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 within the Recall Labs components in scope.

Overview

Recall is an L2 built on Filecoin using the Interplanetary Consensus Framework. Recall aims to offer data availability services by storing data offchain and making it available onchain. By using a hierarchical subnet architecture Recall delivers scalability and fast consensus.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the recallnet/contracts and the recallnet/ipc repositories.

The scope of this time-boxed review was strictly limited to files at commit 5a67104 and commit d08b279. Additionally, a refactor of `ValidatorRewarder.sol` was reviewed at commit fe4d3b4 The fixes of the identified issues were assessed at commit a72edb8 and commit fa69b46.

Note: third party libraries and dependencies were excluded from the scope of this assessment.

Approach

The security assessment covered components written in Solidity and Rust.

For the Solidity components, the manual review focused on identifying issues associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout).

Additionally, the manual review process focused on identifying vulnerabilities related to known Solidity anti-patterns and attack vectors, such as re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers.

For a more detailed, but non-exhaustive list of examined vectors, see [1, 2].

To support the Solidity components of the review, the testing team also utilised the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`
- Aderyn: `https://github.com/Cyfrin/aderyn`

For the Rust components, the manual review focused on identifying issues associated with the business logic implementation of the components in scope. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Rust language.

Additionally, the manual review process focused on identifying vulnerabilities related to known Rust anti-patterns and attack vectors, such as unsafe code blocks, integer overflow, floating point underflow, deadlocking, error handling, memory and CPU exhaustion attacks, and various panic scenarios including index out of bounds, panic!(), unwrap(), and unreachable!() calls.

To support the Rust components of the review, the testing team also utilised the following automated testing tools:

- Clippy linting: `https://doc.rust-lang.org/stable/clippy/index.html`

- Cargo Audit: `https://github.com/RustSec/rustsec/tree/main/cargo-audit`
- Cargo Outdated: `https://github.com/kbknapp/cargo-outdated`
- Cargo Geiger: `https://github.com/rust-secure-code/cargo-geiger`
- Cargo Tarpaulin: `https://crates.io/crates/cargo-tarpaulin`

Output for these automated tools is available upon request.

Coverage Limitations

Due to the time-boxed nature of this review, all documented vulnerabilities reflect best effort within the allotted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 19 issues during this assessment. Categorised by their severity:

- Critical: 1 issue.
- High: 4 issues.
- Medium: 3 issues.
- Low: 5 issues.
- Informational: 6 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Recall Labs components in scope. 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 contracts, including gas optimisations, 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
RECL-01	Blob Read Request Callback Is Performed By `SYSTEM_ACTOR`	Critical	Resolved
RECL-02	Blob Read Request ID Collision For BLS & Delegated Addresses	High	Resolved
RECL-03	No Access Control On `update_object_metadata()`	High	Resolved
RECL-04	Unspent Tokens Are Sent To Origin	High	Resolved
RECL-05	`whenActive` Performs A Silent Return	High	Resolved
RECL-06	Expired Approval Is Considered Valid In `push()`	Medium	Resolved
RECL-07	Approval Expiry Can Overflow	Medium	Resolved
RECL-08	`debit_accounts()` May Run Out Of Gas	Medium	Resolved
RECL-09	`rt.message().origin()` Is Used For Authentication	Low	Resolved
RECL-10	No Validation For `RecallConfig`	Low	Resolved
RECL-11	Overflow In `capacity_available()`	Low	Resolved
RECL-12	User Does Not Receive Blob Read Request ID	Low	Resolved
RECL-13	Missing Constructor For `ValidatorGater` & `ValidatorRewarder`	Informational	Resolved
RECL-14	`recipient.transfer()` Is Used	Informational	Resolved
RECL-15	Transfers Will Emit An `Approval` Event	Informational	Resolved
RECL-16	Pausing & Unpausing Have The Same Role	Informational	Resolved
RECL-17	Miscellaneous General Comments	Informational	Closed
RECL-18	`hash_rm()` Returns Succesfully on Error	Low	Closed
RECL-19	Trivial Hash Collisions For `RuntimeHasherWrapper`	Informational	Closed

RECL-01	Blob Read Request Callback Is Performed By `SYSTEM_ACTOR`		
Asset	`fendermint/actors/blob_reader/*, fendermint/vm/interpreter/src/chain.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

When opening a read request in `blob_reader` a user can specify a callback address and method. Next, this request is picked up by `get_open_read_requests()` in `chain.rs` and processed. Later, the callback is fetched from `read_request_callback()` and the return value of the request is executed.

However, as seen in `create_implicit_message()` the callback is executed with `from` as the `SYSTEM_ACTOR` address. This is problematic since the `SYSTEM_ACTOR` is privileged and many functions use `rt.validate_immediate_caller_is(std::iter::once(&SYSTEM_ACTOR_ADDR))` as access control.

Therefore, by using the callback, an attacker may call any function as the system address, circumventing access control.

fendermint/vm/interpreter/src/chain.rs

```
1445 fn create_implicit_message(to: Address, method_num: u64, params: RawBytes) -> Message {
         Message {
1447         version: Default::default(),
             from: system::SYSTEM_ACTOR_ADDR, //@audit callback is sent from `SYSTEM_ACTOR`
1449         to,
             sequence: 0,
1451         value: Default::default(),
             method_num,
1453         params,
             gas_limit: fvm_shared::BLOCK_GAS_LIMIT,
1455         gas_fee_cap: Default::default(),
             gas_premium: Default::default(),
1457     }
     }
```

A limitation of this attack is that the parameter of the callback is of type `Vec<u8>`, which represents the content of the blob that was read. This means that a targeted function should also take a single `Vec<u8>` as parameter, otherwise a decoding error may occur.

Recommendations

A solution is to send these callbacks from a different, unprivileged address.

Resolution

The development team has addressed this issue by having the `BLOB_READER_ACTOR_ADDR` send the callbacks in PR #500 and PR #515.

RECL-02	Blob Read Request ID Collision For BLS & Delegated Addresses		
Asset	`fendermint/actors/blob_reader/src/state.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

Every blob read request that is opened gets a `request_id`. This is the hash of the parameters of the request, where each parameter is padded or shortened to exactly 32 bytes.

blob_reader/src/state.rs

```
42  let blob_hash_bytes = pad_to_32_bytes(blob_hash.0.as_ref());
    let offset_bytes = pad_to_32_bytes(&offset.to_be_bytes());
44  let len_bytes = pad_to_32_bytes(&len.to_be_bytes());
    let callback_addr_bytes = pad_to_32_bytes(&callback_addr.to_bytes());
46  let callback_method_bytes = pad_to_32_bytes(&callback_method.to_be_bytes());
    let combined_bytes: Vec = [
48      &blob_hash_bytes[..],
        &offset_bytes[..],
50      &len_bytes[..],
        &callback_addr_bytes[..],
52      &callback_method_bytes[..],
    ]
54  .concat();
    let mut hasher = Sha256::new();
56  hasher.update(&combined_bytes);
    let request_id: [u8; 32] = hasher.finalize().into();
```

blob_reader/src/state.rs

```
127 pub(crate) fn pad_to_32_bytes(input: &[u8]) -> [u8; 32] {
        let mut padded = [0u8; 32];
129     let start = 32_usize.saturating_sub(input.len());
        padded[start..].copy_from_slice(&input[..input.len().min(32)]);
131     padded
    }
```

The `callback_addr` parameter has type `Address` which can have 5 different underlying formats. However, two of these formats are longer than 32 bytes: BLS which is 48 bytes and a delegated address which can be up to 66 bytes. This makes it trivial to create colliding request ids for requests with BLS or delegated addresses: simply leave the upper 32 bytes the same while modifying the lower bytes.

An attacker can use this to overwrite and thus effectively cancel any read request for a BLS or delegated address.

Recommendations

A possible solution is to pad the `callback_addr` to a larger number of bytes such that it can fit all formats.

Alternatively, consider moving to sequential ids instead of a hash-based system. This removes the worry of collisions and overwrites and allows multiple request with the same parameters to co-exist.

Another solution is to hash any objects which are longer than 32 bytes such that the hash output is now 32 bytes.

Resolution

The development team has addressed this issue in PR #501 by implementing sequential ids as `u64` values that get wrapped into a `Hash`.

RECL-03	No Access Control On `update_object_metadata()`		
Asset	`fendermint/actors/bucket/src/actor.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

The `update_object_metadata()` function in the `bucket` actor can be used to update the metadata for an object in the bucket. However there is no access control for this function, meaning that anyone can arbitrarily change, add or delete metadata for any object.

bucket/src/actor.rs

```
156 fn update_object_metadata(
      rt: &impl Runtime,
158   params: UpdateObjectMetadataParams,
    ) -> Result<(), ActorError> {
160   rt.validate_immediate_caller_accept_any()?; //@audit no access control
      // ...
162 }
```

Recommendations

Add access control to the call `update_object_metadata()`.

Resolution

The development team has addressed this issue in PR #524.

RECL-04	Unspent Tokens Are Sent To Origin		
Asset	`fendermint/actor/blobs/src/actor.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

When adding a blob using `add_blob()` a user must pay for the blob with credits. If the user does not have enough credits they can purchase them inline by attaching tokens to the call to `add_blob()`. Any tokens that exceed the cost of the blob are sent back to the user.

However, the excess tokens are sent to the transaction origin `rt.message().origin()` instead of the message caller `rt.message().caller()` who attached the tokens.

blobs/src/actor.rs

```
318 // Send the tokens not required for the buying back
    if !unspent_tokens.is_zero() {
320     extract_send_result(rt.send_simple(&origin, METHOD_SEND, None, unspent_tokens))?;
    }
```

It is noted that the *bucket* actor may also call `add_blob()`, hence the immediate sender may not always be the end user.

The impact is the wrong user is refunded if the sender is not the origin. For example in the case of the *multisig* actor the origin is a single signer rather than the *multisig*.

Recommendations

A solution is to add an extra parameter to the function call which is a refund address. All tokens may then be sent to the dedicated refund address.

Resolution

The development team has addressed this issue in PR #489 and PR #75.

RECL-05	`whenActive` Performs A Silent Return		
Asset	`ValidatorGater.sol, ValidatorRewarder.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

The `whenActive` modifier in `ValidatorGater` performs a silent return when `active` is false instead of reverting. This has implications for all functions marked with `whenActive` in case `_active` is false:

ValidatorGater.sol

```
48 modifier whenActive() {
     if (!_active) {
50     return; // Skip execution if not active
     }
52   _; // Continue with function execution if active
   }
```

- Most notably `interceptPowerDelta()` will return successfully regardless of the actual value of `newPower`. As such any power range set in `ValidatorGater` is circumvented when validators join, leave, or adjust their stake in `SubnetActorManagerFacet`.
- `setSubnet()`, `approve()` and `revoke()` will all return successfully without the operation actually being performed. However seeing as these functions are intended to be called by EOA's instead of contracts the impact is smaller.

The same issue is present in `ValidatorRewarder`:

- Notably the `notifyValidClaim()` function is also marked as `whenActive`. This function is meant to be called by the system when a validator claims their reward and sends the tokens to the validator. This means that when `_active` is false and a validators claim is processed in `processConsensusClaim()` it will be marked as claimed, but the validator will not receive any rewards. Resulting in the validator losing these rewards permanently.

Recommendations

Alter the modifier `whenActive` such that it reverts upon failure.

Resolution

The development team has addressed this issue in PR #57 by changing the modifier such that it reverts on failure.

RECL-06	Expired Approval Is Considered Valid In `push()`		
Asset	`fendermint/actors/timehub/src/actor.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

The `push()` function is used to add items to a `Timehub`. It can only be called by either the owner of the timehub or users that have a credit approval from the owner. However, the `push()` function does not check if an approval is expired. This means that users with an expired approval from the owner can still successfully call `push()` on the `Timehub`

timehub/src/actor.rs

```
fn push(rt: &impl Runtime, params: PushParams) -> Result {

    // ...

    if origin != owner {
        let approved = get_credit_approval(rt, owner, origin)?.is_some(); //@audit checks that approval exists but does not check
            ↪ its expiry
        if !approved {
            return Err(actor_error!(
                forbidden;
                format!("Unauthorized: missing credit approval from Timehub owner {} to origin {} for Timehub {}", owner, origin,
                    ↪ actor_address)));
        }
    }
```

Recommendations

The issue may be mitigated by validating the expiry of approvals in `push()`.

Resolution

The development team has addressed this issue by validating the expiry of approvals in PR #519.

RECL-07	Approval Expiry Can Overflow		
Asset	`fendermint/actors/blobs/src/state.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

When creating an approval in `approve_credit()` a user can set the `ttl` of the approval to decide when it should expire. However if the `ttl` is set too high it will overflow on line [255]. As a result `expiry` will be negative and the approval will have a negative expiry.

blobs/src/state.rs

```
255 let expiry = ttl.map(|t| t + current_epoch);
```

Seeing as there is no direct impact to an approval having a negative `expiry`, the testing team rates the impact as low.

Recommendations

To resolve the issue add extra validation to the `ttl` parameter. Additionally, consider enabling overflow checks in `Cargo.toml`.

Resolution

The development team has addressed this issue by using `saturating_add()` in PR #504.

RECL-08	`debit_accounts()` May Run Out Of Gas		
Asset	`fendermint/actors/blobs/src/actor.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

The `debit_accounts()` function is called by the system on a regular basis to charge all accounts for their blob usage. This function iterates through all the accounts and debits them based on their usage. However, this will lead to out-of-gas errors if there are too many accounts resulting in a DoS of the debiting functionality.

The development team is aware of this issue and has marked it as todo. However, since it currently still represents an attack vector the testing team has decided to include it in the report.

blobs/src/actor.rs

```
263 /// TODO: Take a start key and page limit to avoid out-of-gas errors.
fn debit_accounts(rt: &impl Runtime) -> Result<(), ActorError> {
265     // ...
}
```

Recommendations

A possible solution here is to debit a limited number of accounts each block and rotate them out. Such that every account is still debited after some time regardless of the total number of accounts.

Resolution

The development team has addresssed this issue in PR #522 and PR #562.

RECL-09	`rt.message().origin()` Is Used For Authentication		
Asset	`fendermint/actors/*`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

Throughout the actors `rt.message().origin()` is often used as the address for authentication. In general using the transaction origin for authentication is discouraged since it exposes users to 'phishing'-style attacks and may cause issues for transaction relays. Some example where `rt.message().origin()` is used:

- `approve_credit()` in the `blobs` actor
- `add_blob()` in the `blobs` actor
- `delete_blob()` in the `blobs` actor
- `overwrite_blob()` in the `blobs` actor
- The `bucket` actor indirectly uses the origin authentication from the `blobs` actor
- `push()` in the `timehub` actor

The origin sender may be different to the account in question when dealing with the `multisig` account, where the origin is a singer of the `multisig`.

Recommendations

Consider using `rt.message().caller()` instead of `rt.message().origin()`.

Resolution

The development team has addressed these issues in the following PRs:

- PR #524
- PR #575
- PR #582
- PR #594
- PR #597
- PR #598

- PR #600
- PR #611
- PR #615
- PR #617

RECL-10	No Validation For `RecallConfig`		
Asset	`fendermint/actors/recall_config/*`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

When setting the system configuration in `set_config()` no validation of the given config occurs. It is recommended to do some basic checks such that obviously incorrect configurations can not be set. Some possible checks that could be done:

- `blob_min_ttl` is not negative. Otherwise users can set a negative `ttl` when adding a blob, resulting in a negative `credit_required` which may cause other issues.
- `blob_default_ttl` is larger than `blob_min_ttl`.
- `token_credit_rate` is positive.
- `blob_credit_debit_interval` is positive and non-zero.

Recommendations

Ensure the above comments are understood and consider changes if desired.

Resolution

The development team has addressed this issue in PR #507.

RECL-11	Overflow In `capacity_available()`		
Asset	`fendermint/actors/blobs/src/state.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

The `capacity_available()` function calculates how much blob capacity is available. However, it can overflow if `blob_capacity_total` is smaller than `self.capacity_used`. This is possible since overflow checks are not enabled in `Cargo.toml` for the `release` profile.

blobs/src/state.rs

```
1270 /// Return available capacity as a difference between `blob_capacity_total` and `capacity_used`.
     fn capacity_available(&self, blob_capacity_total: u64) -> u64 {
1272     blob_capacity_total - self.capacity_used
     }
```

Most notably, this impacts the `add_blob()` function. If `config.blob_capacity` is lowered to under `self.capacity_used`, an overflow occurs and `available_capacity` will be very large, resulting in the check on line [**718**] being bypassed. Seeing as an admin must set `config.blob_capacity` the testing team rates the likelihood of this issue as low.

blobs/src/state.rs

```
717  let available_capacity = self.capacity_available(config.blob_capacity);
     if size > available_capacity {
719      return Err(ActorError::forbidden(format!(
             "subnet has insufficient storage capacity (available: {}; required: {})",
721          available_capacity, size
         )));
723  }
```

Similarly, the check on line [**161**] in `buy_credits()` will be bypassed if `config.blob_capacity` is set too low.

blobs/src/state.rs

```
160  // Don't sell credits if we're at storage capacity
     if self.capacity_available(config.blob_capacity).is_zero() {
162      return Err(ActorError::forbidden(
             "credits not available (subnet has reached storage capacity)".into(),
164      ));
     }
```

Recommendations

Consider enabling overflow checks in `Cargo.toml`. Additionally consider adding a check in `capacity_available()` to ensure that `blob_capacity_total` is larger than `self.capacity_used`.

Resolution

The development team has addressed this issue in PR #502.

RECL-12	User Does Not Receive Blob Read Request ID		
Asset	`fendermint/actors/blob_reader/*`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

When a user opens a blob read request they do not receive the request ID corresponding to their request. Additionally, during the callback only the blob bytes are given, and no request ID is passed along.

As a result, if a contract has multiple outstanding blob read requests it can not identify which callback corresponds to which request.

blob_reader/src/actor.rs

```
35 pub fn open_read_request(
       &mut self,
37     blob_hash: Hash,
       offset: u32,
39     len: u32,
       callback_addr: Address,
41     callback_method: u64,
   ) -> Result<(), ActorError> {
43
       //...
45
       Ok(()) //@audit request ID is not returned here
47 }
```

Recommendations

Consider returning the request ID both when opening the request and during the callback.

Resolution

The development team has addressed this issue by returning the request ID in PR #515.

RECL-13	Missing Constructor For `ValidatorGater` & `ValidatorRewarder`
Asset	`ValidatorGater.sol, ValidatorRewarder.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Both `ValidatorGater` and `ValidatorRewarder` have no constructor.

The issue arises since each contract is intended to be deployed behind a proxy. Therefore, the implementation contract may not be initialised upon construction.

Recommendations

It is recommended to add a constructor, which contains `_disableInitializers()`, to `ValidatorGater` and `ValidatorRewarder`.

Resolution

The development team has addressed the issue in PR #78.

RECL-14	`recipient.transfer()` Is Used
Asset	`Faucet.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `drip()` function in `Faucet.sol` uses `recipient.transfer(amount)` to send tokens to the recipient. In general, it is recommended to use `recipient.call{value: amount}("")` instead. This is because `transfer()` only sends a 2300 gas stipend, which may cause contracts with a complex `receive()` function to run out of gas and be unable to receive funds from the faucet.

In case this contract is deployed on Filecoin's EVM, the stipend is automatically adjusted and larger but still limited.

Recommendations

Ensure the above comments are understood and consider changes if desired.

Resolution

The development team has addressed this issue in PR #57.

RECL-15	Transfers Will Emit An `Approval` Event
Asset	`Recall.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The call to `_approve` on line [**119**] will emit an approval event. This means that every time `_spendAllowance` is called (i.e. for every `transferFrom` or `interchainTransferFrom` call) an `Approval()` event is emitted. This may cause issues in offchain indexing.

Recall.sol

```
113 function _spendAllowance(address sender, address spender, uint256 amount)
        internal
115     override(ERC20Upgradeable, InterchainTokenStandard)
    {
117     uint256 _allowance = allowance(sender, spender);
        if (_allowance != type(uint256).max) {
119         _approve(sender, spender, _allowance - amount); //@audit will emit an approval event
        }
121 }
```

Recommendations

Consider if this is desired behaviour, otherwise use `_approve(sender, spender, _allowance - amount, false)` instead.

Resolution

This issue was fixed in PR #64.

RECL-16	Pausing & Unpausing Have The Same Role
Asset	`Recall.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The role required to pause transfers for the Recall token is the same role required for unpausing. It is best practice to have two separate roles, a pauser and an unpauser, such that they can be set to different addresses. This allows for an easier to access and quicker response for pausing the protocol, while unpausing can require a more heavily guarded and timelocked multisig.

Recall.sol

```
73  /// @dev Pauses all token transfers
    function pause() external onlyRole(ADMIN_ROLE) {
75    _pause();
    }
77
    /// @dev Unpauses all token transfers
79  function unpause() external onlyRole(ADMIN_ROLE) {
      _unpause();
81  }
```

Recommendations

Ensure the above comments are understood and consider changes if desired.

Resolution

This issue was fixed in PR #66.

RECL-17	Miscellaneous General Comments
Asset	All contracts
Status	**Closed:** See Resolution
Rating	Informational

Description

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

1. **Uninitialised `UUPSUpgradeable`**
   Related Asset(s): ValidatorGater.sol
   `UUPSUpgradeable` is not initialised in `ValidatorGater`. Seeing as `__UUPSUpgradeable_init()` is a no-op this currently has no impact. Consider adding it for best practice.

2. **Transient Storage Is Not Supported On Filecoin**
   Related Asset(s): *.sol
   All contracts use Solidity version 0.8.26 which has support for `tstore` and `tload`. However, since these opcodes are not yet supported by filecoin using them will revert execution. Seeing as these instructions are currently not used in the contracts there is no direct impact. However, care must be taken for any potential future usage.

3. **Non Verbose Revert**
   Related Asset(s): Recall.sol
   Currently `_spendAllowance()` will implicitly revert if `_allowance < amount`. Consider adding an explicit check with a more verbose revert. This makes it easier for users to figure out why a call failed.

Recall.sol

```
113 function _spendAllowance(address sender, address spender, uint256 amount)
        internal
115     override(ERC20Upgradeable, InterchainTokenStandard)
    {
117     uint256 _allowance = allowance(sender, spender);
        if (_allowance != type(uint256).max) {
119         _approve(sender, spender, _allowance - amount); //@audit reverts if `_allowance` < `amount`
        }
121 }
```

4. **Gas Optimisations**
   Related Asset(s): Recall.sol
   The variables `ADMIN_ROLE` and `MINTER_ROLE` can be made constant. This will result in significant gas savings.

5. **`ReentrancyGuardUpgradeable` Is Unused In `Recall.sol`**
   Related Asset(s): Recall.sol
   The Recall token contract inherits and initialises `ReentrancyGuardUpgradeable`, but no functions are marked as `nonReentrant`. Consider removing `ReentrancyGuardUpgradeable` or mark some of the functions as `nonReentrant`. Given that there are no external calls in `Recall.sol` the reentrancy risk is low.

6. **Redundant Logic**
   Related Asset(s): recall/executor/src/lib.rs
   Lines 540 to 543 are redundant and could be removed to simplify the logic in `preflight_message()`.

recall/executor/src/lib.rs

```
let (gas_cost, sponsored_gas_cost) = if gas_allowance.sponsored_amount.is_zero() {
  // Consume from own allowance
  (total_gas_cost, TokenAmount::zero())
} else {
  // Prioritize sponsor allowance when consuming
  if gas_allowance.sponsored_amount > total_gas_cost {
    // Consume from sponsored allowance
    (TokenAmount::zero(), total_gas_cost)
  } else {
    // Consume entire sponsored allowance
    (
    &total_gas_cost - &gas_allowance.sponsored_amount,
    gas_allowance.sponsored_amount,
    )
  }
};
```

7. **Unused Libraries**

   Related Asset(s): ValidatorRewarder.sol (fe4d3b4)

   The libraries in `ValidatorRewarder` are unused. As such, they can be removed to reduce the contract size and simplify the code.

ValidatorRewarder.sol

```
contract ValidatorRewarder is IValidatorRewarder, UUPSUpgradeable, OwnableUpgradeable {
  using SubnetIDHelper for SubnetID; //@audit unused
  using SafeERC20 for Recall;        //@audit unused

  // ...
}
```

Recommendations

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

Resolution

This issue was fixed in PR #63 and #82. The comment about simplifying logic was discussed in issue #518.

RECL-18	`hash_rm()` Returns Succesfully on Error		
Asset	`recall/syscalls/src/lib.rs`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `hash_rm()` function is used in the blob actor to delete a blob from the Iroh client. However, if an error occurs in `hash_rm()`, the function will return successfully instead of passing an error back to the function caller. This occurs on both line [42] and line [51]. As a result, a blob may be registered as deleted and users no longer have to pay for it, even though it is still on the disk.

recall/syscalls/src/lib.rs

```
31 pub fn hash_rm(context: Context<'_, impl RecallOps>, hash_offset: u32) -> Result<()> {
       let hash_bytes = context.memory.try_slice(hash_offset, 32)?;
33     let hash = Hash::from_bytes(hash_source(hash_bytes)?);
       let iroh = IROH_INSTANCE.clone();
35
       // Don't block the chain with this.
37     spawn(async move {
           let iroh_client = match iroh.lock().await.client().await {
39             Ok(client) => client,
               Err(e) => {
41                 tracing::error!(hash = ?hash, error = e.to_string(), "failed to initialize Iroh client");
                   return; //@audit successful return here
43             }
           };
45         // Deleting the tag will trigger deletion of the blob if it was the last reference.
           // TODO: this needs to be tagged with a "user id"
47         let tag = iroh::blobs::Tag(format!("stored-{hash}").into());
           match iroh_client.tags().delete(tag.clone()).await {
49             Ok(_) => tracing::debug!(tag = ?tag, hash = ?hash, "removed content from Iroh"),
               Err(e) => {
51                 tracing::warn!(tag = ?tag, hash = ?hash, error = e.to_string(), "deleting tag from Iroh failed"); //@audit
                       ↪  successful return here
               }
53         }
       });
55     Ok(())
   }
```

Recommendations

Ensure the above comments are understood and consider changes if desired.

Resolution

The Recall development team has determined this to be a non-issue since not returning errors based on the behaviour of `Iroh`, prevents consensus desynchronisation. An alternate solution has been suggested, where the local state issue could still be resolved by reclaiming orphaned blobs through periodic log scans or a separate cleanup process. The

team at Sigma Prime agrees that the fix proposed by the development team, is a more appropriate resolution for the issue in question.

RECL-19	Trivial Hash Collisions For `RuntimeHasherWrapper`
Asset	`recall/ipld/src/hash_algorithm.rs`
Status	**Closed:** See Resolution
Rating	Informational

Description

The `finish()` function for `Hasher` is meant to return the hash of the written data. However, the current implementation for `RuntimeHasherWrapper` will always return `0` instead of an actual hash. As such, if `RuntimeHasherWrapper` were to be used as a hash function it would cause issues such as trivial hash collisions.

Seeing as this function is currently not used, there is no direct impact and the testing team rates this issue as informational. However, it is recommended to either implement a real hash function here or remove the function to avoid any future issues.

recall/ipld/src/hash_algorithm.rs

```
16 impl Hasher for RuntimeHasherWrapper {
       fn finish(&self) -> u64 {
18         // u64 hash not used in hamt
           0
20     }
```

Recommendations

Ensure the above comments are understood and consider changes if desired.

Resolution

The Recall development team has determined this as a non issue and as such has closed it. The `Hasher` implementation only intercepts key bytes and is used only together with `FvmHashSha256`.

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.

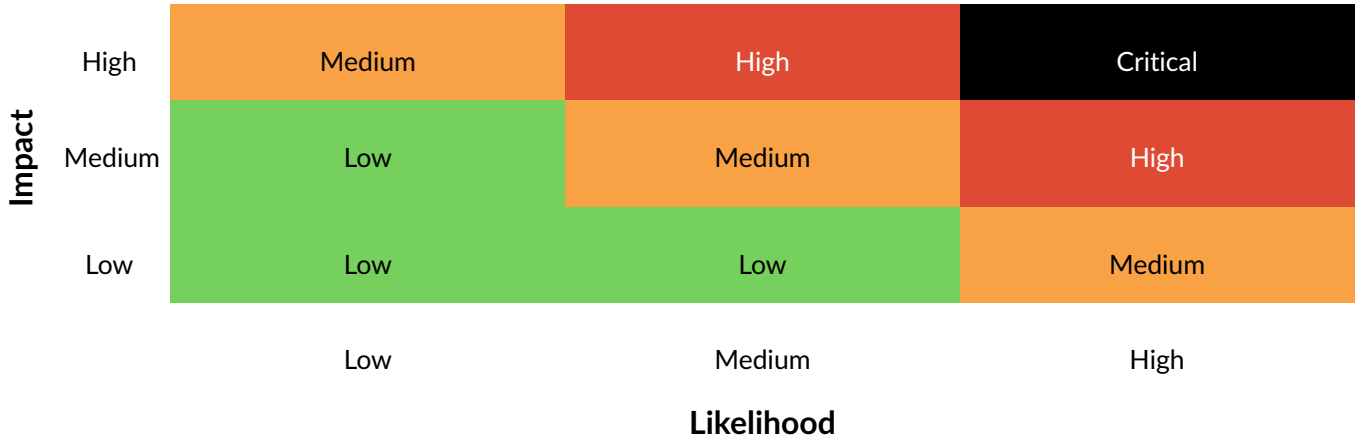

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].