Aurora

Rainbow Bridge ETH2 Client

Version: 2.2

June, 2023

Contents

Introduction **2**
- Disclaimer . . . 2
- Document Structure . . . 2
- Overview . . . 2

Security Assessment Summary **3**
- Findings Summary . . . 3

Detailed Findings **4**

Summary of Findings **5**
- Skipped Slots Cause Relayer to Stall . . . 6
- Relayer Will Submit Potentially Stale Blocks Until Submission Limit Is Reached . . . 7
- Unbounded State Growth of NEAR `eth2-client` . . . 8
- Lack of Validation on DAO Proposal Parameters . . . 9
- `eth2-client` Cannot Update Finality If Beacon Chain Finalisation Period is Skipped . . . 10
- `eth2-client` Cannot Update With Large Finality Gaps Due To Gas Limits . . . 11
- Lack of Polling Delay in Relayer Increases Computational Load . . . 13
- Hand Made Updates Do Not Confirm Number Of Signers . . . 14
- Misguided Mainnet Safety Checks . . . 16
- Lack of Client Syncing Status Checks . . . 17
- Relayer Does Not Account for Unfinalised Headers Submission Quota . . . 18
- Missing `LightClientUpdate` Checks in DAO `eth2-validator` . . . 19
- Missing `LightClientUpdate` Checks in `eth2-client` . . . 21
- Client Initialisation Process . . . 23
- The Package `contract_wrapper` Does Not Compile . . . 24
- Excessive RPC Timeout Configuration . . . 25
- Potential Subtraction Overflow in `send_hand_made_light_client_update()` . . . 26
- Centralisation Risks . . . 27
- Miscellaneous General Comments . . . 29

A Vulnerability Severity Classification **33**

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Aurora Rainbow Bridge Eth2 Client. The review focused on the security aspects of both NEAR contracts and off-chain verification Rust programs comprising the Ethereum to NEAR side of the rainbow bridge.

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 Aurora Rainbow Bridge Eth2 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 within the Aurora Rainbow Bridge Eth2 Client.

Overview

The Aurora Rainbow Bridge contains a bridge between the NEAR blockchain and Ethereum blockchain. The focus of this review was related to the recent Merge upgrade of Ethereum. The upgrade required redesigning the Ethereum light client on NEAR. An Ethereum Beacon Chain Client will now be used to provide new finalised updates to the NEAR blockchain.

The protocol consists of four main components. First is a smart contract implementation of a Beacon Chain light client on the NEAR blockchain. The second component is a relayer which is an off-chain rust program that communicates with a full Beacon Node via RPC.

A limitation of the light client implementation in a NEAR smart contract is that BLS signature verification cannot be done in a single block due to gas limits. To overcome this issue the development team have added two extra components. One is a Sputnik DAO smart contract where each new proposal is a potential light client update. The second component is a validator for Sputnik DAO which will verify the BLS signature off-chain and submit votes on-chain. Any proposal that pass the vote will be forwarded to the on-chain Beacon chain light client implementation.

Security Assessment Summary

This review was conducted on the files hosted on the repositories rainbow-bridge and eth2-on-near-client-validator. They were assessed at commits 50427ed and 738833b.

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the source code. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the NEAR VM (for example, verifying correct storage/memory layout). Additionally, the manual review process focused on all known Rust and NEAR Smart Contract anti-patterns and attack vectors. These include, but are not limited to, the following vectors: panics, front-running, integer overflow/underflow and correct visibility specifiers.

To support this review, the testing team used the following automated testing tools:

- cargo audit: `https://crates.io/crates/cargo-audit`
- cargo outdated: `https://crates.io/crates/cargo-outdated`
- clippy: `https://github.com/rust-lang/rust-clippy`

Output for these automated tools is available upon request.

Findings Summary

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

- High: 1 issue.
- Medium: 2 issues.
- Low: 10 issues.
- Informational: 6 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Aurora Rainbow Bridge Eth2 Client. 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 source code 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
RBE2-01	Skipped Slots Cause Relayer to Stall	High	Resolved
RBE2-02	Relayer Will Submit Potentially Stale Blocks Until Submission Limit Is Reached	Medium	Resolved
RBE2-03	Unbounded State Growth of NEAR `eth2-client`	Medium	Closed
RBE2-04	Lack of Validation on DAO Proposal Parameters	Low	Resolved
RBE2-05	`eth2-client` Cannot Update Finality If Beacon Chain Finalisation Period is Skipped	Low	Closed
RBE2-06	`eth2-client` Cannot Update With Large Finality Gaps Due To Gas Limits	Low	Resolved
RBE2-07	Lack of Polling Delay in Relayer Increases Computational Load	Low	Resolved
RBE2-08	Hand Made Updates Do Not Confirm Number Of Signers	Low	Resolved
RBE2-09	Misguided Mainnet Safety Checks	Low	Closed
RBE2-10	Lack of Client Syncing Status Checks	Low	Resolved
RBE2-11	Relayer Does Not Account for Unfinalised Headers Submission Quota	Low	Resolved
RBE2-12	Missing `LightClientUpdate` Checks in DAO `eth2-validator`	Low	Resolved
RBE2-13	Missing `LightClientUpdate` Checks in `eth2-client`	Low	Resolved
RBE2-14	Client Initialisation Process	Informational	Resolved
RBE2-15	The Package `contract_wrapper` Does Not Compile	Informational	Resolved
RBE2-16	Excessive RPC Timeout Configuration	Informational	Resolved
RBE2-17	Potential Subtraction Overflow in `send_hand_made_light_client_update()`	Informational	Resolved
RBE2-18	Centralisation Risks	Informational	Resolved
RBE2-19	Miscellaneous General Comments	Informational	Resolved

RBE2-01	Skipped Slots Cause Relayer to Stall		
Asset	`rainbow-bridge/eth2near/eth2near-block-relay-rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

The NEAR `eth2-client` may accept an attested execution header into `unfinalized_headers`, which the beacon chain later *re-orgs* into a skipped slot.

In such cases, the `eth2_to_near_relay` function `block_known_on_near(slot)` returns an `Error` while processing query data. This error is not handled, but is instead propagated by the calling function `get_last_slot()`, which is searching for the last beacon chain slot submitted to `unfinalized_headers` in order to submit a batch of headers up to the current unfinalised head. An inability to discover the last unfinalised beacon slot submitted will cause the `eth2_to_near_relay` to stall until the finalised slot on NEAR is larger than the last submitted slot.

The root cause of the error is due to the following chain of queries.

1. `get_last_slot()` attempts to find the last non-skipped unfinalised slot submitted to the NEAR `eth2-client`.
2. Within, `last_submitted_slot = self.eth_client_contract.get_last_submitted_slot()` returns the skipped slot.
3. Until the skipped slot becomes finalised on the beacon chain, `let slot = max(finalized_slot, last_submitted_slot);` also returns the skipped slot.
4. In either linear or binary search options, the conditional statement `self.block_known_on_near(slot)?` will be reached.
5. `self.block_known_on_near(slot)?` queries `get_beacon_block_body_for_block_id(&format!("{}", slot)` with the skipped slot, which returns an `Error`.
6. This `Error` case is not handled, and causes `block_known_on_near(slot)?` to fail.
7. This `Error` case is not handled in the calling function `get_last_slot()`, causing it to fail also.

Recommendations

This issue can be mitigated by handling the error case for skipped slots in `self.block_known_on_near(slot)`. Rather than propagating the error, allow `get_last_slot()` to search for the correct last submitted block.

Additionally, only submitting finalised execution headers on-chain mitigates this issue. Only submitting finalised blocks will prevent `last_submitted_slot` from becoming a skip slot.

Resolution

The issue if fixed by modifying the search logic to account for skipped slots. Updates can be seen in the following PRs #800 and #832.

RBE2-02	Relayer Will Submit Potentially Stale Blocks Until Submission Limit Is Reached		
Asset	`rainbow-bridge/eth2near/eth2near-block-relay-rs/src/eth2near_relay.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

`eth2_to_near_relay` submits the head of the beacon chain from the most recently attested (unfinalised) slot to the NEAR `eth2-client`, which gets registered in `unfinalized_headers`. Limits are set on the number of unfinalised submissions from each account to prevent unbounded state growth.

Each submission has the potential to be excluded from the finalised beacon chain in the case of a *re-org*. The relayer will therefore reach its unfinalised block submission quota through its routine operation. No longer will the account be able to submit new blocks or unregister as a submitter to receive its storage deposit refund. If only one relayer is connected to `eth2-client`, then the `Eth2Near` side of bridge operation will be stalled until a new relayer can be connected.

One cause of a beacon chain *re-org* is when the head of beacon chain is proposed late, after other validators have already attested to a skip slot. The next proposer will fork the late block (current head) and will create a new head which does not include the late block.

The code which determines which blocks to submit occur in the main `run()` loop. The following code selects the highest beacon slot by fetching the slot number of the current head through `get_last_slot_number()`. Each block between the last finalised block on NEAR and the beacon head will then be submitted on-chain.

```
let last_eth2_slot_on_eth_chain: u64 =
match self.beacon_rpc_client.get_last_slot_number() {
    Ok(slot) => slot.as_u64(),
    Err(err) => {
        warn!(target: "relay", "Fail to get last slot on Eth. Error: {}", err);
        continue;
    }
};
```

Recommendations

`eth2_to_near_relay` can be updated to only submit finalised execution headers to the NEAR `eth2-client`.

A *re-org* of a finalised block breaks the underlying assumptions of the Ethereum consensus. By applying the assumption that finalised blocks will not be *re-orged*, the number of blocks submitted by the relayer will be exactly the number of blocks between the current and previous finalised blocks. Hence, it will remain safely within its submission quota.

Resolution

A resolution to this issue has been implemented in commit 2198169. The mitigation implements a configurable variable `submit_only_finalized_blocks` which, if enabled, will cause `last_eth2_slot_on_eth_chain` to be the last finalised slot of the Beacon Chain rather than the current head.

RBE2-03	Unbounded State Growth of NEAR `eth2-client`		
Asset	`rainbow-bridge/contracts/near/eth2-client`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

`eth2-client` processes blocks in three stages:

1. Unfinalised execution headers are added to `unfinalized_headers` map.
2. Once they are finalised, headers are moved from `unfinalized_headers` to `finalized_execution_blocks` by tracing their ancestry up to the current finalised head of the beacon. The function only iterates over headers which are finalised, non-canonical headers remain as stale data.
3. Blocks are pruned from `finalized_execution_blocks` after a configurable period elapses (i.e. 7 days) and forgotten by the NEAR `eth2-client`

Detailed in RBE2-02, `eth2_to_near_relay` is highly likely to submit headers which become stale data in `unfinalized_headers` during its routine operation, until it becomes non-functional.

There is no method to remove stale data from `unfinalized_headers` in the NEAR `eth2-client`, so this process implies that the state of the contract can grow indefinitely. Although there are practical limits enforced by the relayer submission quota `max_submitted_blocks_by_account` parameter of `eth2-client`, a new relayer must be added when the former reaches its submission quota in order for the bridge to remain operational. Therefore, this process may continue with unbounded state growth, or halt the operation of the bridge.

Furthermore, because `unregister_submitter()` requires a relayer to have zero pending submissions in `unfinalized_headers`, a relayer who submits a single non-canonical block will never be able to unregister and recover their storage deposit.

Recommendations

One way to mitigate this issue would be to allow stale headers to be pruned from `unfinalized_headers` in the NEAR `eth2-client` contract if they have a block number lower than the current finalised block.

Resolution

The development team are aware of the issue and have provided the following comment.

We are aware of this, we didn't implement a clean method API due to multiple reasons:

- *Time limits before audit and this task had low priority.*
- *We can control our relayers to avoid submitting forked blocks.*
- *The cost of the storage is not too high, so we can just register a new relayer.*

RBE2-04	Lack of Validation on DAO Proposal Parameters		
Asset	`eth2-to-near-client-validator`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

In NEAR `eth2-client` operation mode with a `trusted_signer`, the `eth2_to_near_relay` should not submit a `LightClientUpdate` to the `eth2-client` directly. Rather, it creates `FunctionCall` proposal within SputnikDAO that will call `submit_beacon_chain_light_client_update()` via the DAO as a `trusted_signer` (if approved by the DAO `eth2-validators`). Each DAO `eth2-validator` must verify the `LightClientUpdate` off-chain, and vote to approve the `FunctionCall` proposal.

The DAO `eth2-validator` off-chain program verifies that the `FunctionCall` proposal calls `submit_beacon_chain_light_client_update()`. However, it does not verify that the `reciever_id` of the proposal, unpacked here, is indeed the NEAR `eth2-client` contract, nor does it check `actions[0].gas` or `actions[0].deposit` parameters.

The impact is rated as medium as the lack of checks would allow a proposal to transfer an uncapped amount of native tokens from the DAO contract to a user defined account. The proposal will be approved by the client validators so long as the function signature and `LightClientUpdate` are valid. Limited value is intended to be held in the contract in the range of 200 NEAR tokens and thus only rated medium severity. The likelihood is rated as low as the setup will include only one account which may create proposals, that account is controlled by the Aurora team.

Recommendations

DAO `eth2-validators` should reject proposals which:

1. do not target the NEAR `eth2-client` contract
2. have a non-zero deposit or
3. have an unreasonable gas field.

Resolution

Additional checks have been added in PR #4 resolving the issue. Each of the three items listed in Recommendations section have been implemented.

RBE2-05	`eth2-client` Cannot Update Finality If Beacon Chain Finalisation Period is Skipped		
Asset	`rainbow-bridge/contracts/near/eth2-client`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

If finalisation does not occur for an entire beacon chain *period*, upon recieving a finalised `LightClientUpdate`, the NEAR `eth2-client` state will have the condition that `update_period == finalized_period + 2`. This causes an assertion failure verifying that a sync comittee period has not been skipped while processing the update. There is no other method to update finality in the NEAR `eth2-client`, therefore it will halt indefinitely.

The assertion failure occurs in `Eth2Client::verify_finality_branch`, causing `submit_beacon_chain_light_client_update()` to fail:

```
assert!(
    update_period == finalized_period || update_period == finalized_period + 1,
    "The acceptable update periods are '{}' and '{}' but got {}",
    finalized_period,
    finalized_period + 1,
    update_period
);
```

This check is required by the light client implementation because each `BeaconState` contains only the `current_sync_committee` and the `next_sync_committee`. Thus, the client cannot verify the validity of future sync committees. The client needs to verify the chain of sync committees as they progress, and cannot do so if finalisation is not updated for an entire *period* without applying data from an intermediate `BeaconState` within the stalled *period*.

For these reasons, the specifications also include implementations for a forced update method, which may apply a "best valid" `LightClientUpdate`. The estimated update may not contain a finalised block but may increment the *period* and hence update the `current_sync_committee` and `next_sync_committee`.

This issue is has a very low likelihood due to the requirement of not finalising for an *epoch*. A beacon chain *period* is about one day. For finality to not occur, in an entire *period* the Ethereum network will be undergoing considerable stress. Under normal operating conditions finalisation occurs most *epochs* (approx 6 mins).

Recommendations

This issue may be mitigated by allowing manual intervention to update finality from a trusted source, with consideration for constraints detailed in RBE2-06.

Resolution

The development team have acknowledged this issue and intend to implement a fix. However, due to the very low likelihood combined with the significant portion of work required to implement a mitigation the patch has been given a low priority.

RBE2-06	`eth2-client` Cannot Update With Large Finality Gaps Due To Gas Limits		
Asset	`rainbow-bridge/contracts/near/eth2-client`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

There is an unbounded loop in the function `update_finality_header()` which may exceed NEAR gas limits and prevent the NEAR `eth2-client` from applying finality updates, halting it indefinitely.

The loop iterates through `unfinalized_headers` to move them into `finalized_beacon_blocks`.

```
loop {
    let num_of_removed_headers = *submitters_update
        .get(&cursor_header.submitter)
        .unwrap_or(&0);
    submitters_update.insert(cursor_header.submitter, num_of_removed_headers + 1);


    self.unfinalized_headers.remove(&cursor_header_hash);
    self.finalized_execution_blocks
        .insert(&cursor_header.block_number, &cursor_header_hash);


    if cursor_header.parent_hash == self.finalized_beacon_header.execution_block_hash {
        break;
    }


    cursor_header_hash = cursor_header.parent_hash;
    cursor_header = self
        .unfinalized_headers
        .get(&cursor_header.parent_hash)
        .unwrap_or_else(|| {
            panic!(
                "Header has unknown parent {:?}. Parent should be submitted first.",
                cursor_header.parent_hash
            )
        });
}
```

If finality does not occur in the beacon chain for moderate times, iterating through this loop to update finality may exceed NEAR gas limits. Empirical tests on a local testnet, with a 350 Tgas limit, determine that a gap of 356 slots with no blocks produced and one relayer connected will exceed gas limits applying an update.

Under network conditions where many validators are unavailable and unable to update finality, many slots will be skipped because the proposer is offline. Our estimate of a critical finalisation delay must be adjusted to account for the skipped slots that will not add to the size of the loop:

```
critial_pending_blocks = 365 blocks
slot_time = 12 seconds per slot
fraction_skipped_slots = 0.5
critial_finality_delay = (critical_pending_blocks / fraction_skipped_slots) * slot_time
```

$$time = (356 * 12/0.5) = 8544s = 2.4hr$$

`eth2_to_near_relay` also imposes a gas limit of 250 Tgas on this transaction sent via the DAO, which will be reached before the NEAR network limit is reached.

Recommendations

This issue may be mitigated similarly to RBE2-05, requiring manual intervention by a trusted source to force update the client under certain conditions.

An alternate solution is to modify the finality update processing logic into multiple transactions of smaller sizes.

Resolution

The development team have implemented a fix in PR #882 to handle the cases where finality is missing for multiple epochs. This fix arrived after the review period had finished and has not been reviewed.

RBE2-07	Lack of Polling Delay in Relayer Increases Computational Load		
Asset	`rainbow-bridge/eth2near/eth2near-block-relay-rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The main `run()` loop in the `eth2_to_near_relay` does not include a delay between iterations (source). This may cause the `eth2_to_near_relay` to utilize excessive computational resources in its operation.

Specifically, an excessively fast `eth2_to_near_relay` loop execution cycle may cause unacceptable CPU, RAM, or storage usage or exceed RPC-API rate limits.

Recommendations

Include a short delay (0.5 - 12 seconds) between polling, with consideration for the NEAR and Ethereum block production times. The loop execution may finish in multiple locations (L75, L84, and L144), so this should be included for each pathway.

Resolution

Configurable sleep delays have been included for each iteration of the `run()` loop except when transactions have been pushed on-chain in which case there are no delays. The updates have been made in PR #800.

RBE2-08	Hand Made Updates Do Not Confirm Number Of Signers		
Asset	`rainbow-bridge/eth2near/eth2near-block-relay-rs/src/hand_made_finality_light_client_update.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

If `eth2_to_near_relay` detects that the NEAR `eth2-client` is far behind the beacon client finalised head, it must send `HandMadeLightClientUpdate` to update finality instead of its normal execution methods. These updates aim to reduce the number of execution blocks that need to be iterated over in a single transaction, thereby staying within gas limits, an issue detailed in RBE2-06.

There are no checks by `eth2_to_near_relay` that at least two-thirds of the sync committee has signed `HandMadeLightClientUpdate`, therefore it may fail on-chain assertions in the `eth2-client` upon submission and stall the relayer.

The relayer function `HandMadeLightClientUpdate::get_finality_light_client_update()` crafts an update from `signature_slot == attested_slot + 1` without checking the sum of `sync_committee_bits`. This result is handled by the calling function, `send_hand_made_light_client_update()`, within a `match` statement that will send the update to `eth2-client` via `send_specific_light_cleint_update()`:

```
    match HandMadeFinalityLightClientUpdate::get_finality_light_client_update(
        &self.beacon_rpc_client,
        attested_slot,
        BeaconRPCClient::get_period_for_slot(last_finalized_slot_on_near)
            != BeaconRPCClient::get_period_for_slot(attested_slot),
    ) {
        Ok(light_client_update) => {
            let finality_update_slot = light_client_update
                .finality_update
                .header_update
                .beacon_header
                .slot;

            if finality_update_slot <= last_finalized_slot_on_near {
                info!(target: "relay", "Finality update slot for hand made light client update <= last finality update on near.
                      ↪ Increment gap for attested slot and skipping light client update.");
                self.current_gap_between_finalized_and_attested_slot += ONE_EPOCH_IN_SLOTS;
                return;
            }

            trace!(target: "relay", "Hand made light client update: {:?}", light_client_update);

            self.send_specific_light_cleint_update(light_client_update);
        }
        Err(err) => {
            debug!(target: "relay", "Error \"{}\" on getting hand made light client update for attested slot={}.", err,
                  ↪ attested_slot);
            self.current_gap_between_finalized_and_attested_slot += 1;
        }
    }
}
```

If the submission via `send_specific_light_client_update()` fails because the update lacks the requisite signature threshold, no result is returned and the error is not handled. Therefore, `Eth2NearRelay.current_gap_between_finalized_and_attested_slot` is not incremented. Because this variable determines the `attested_slot` from which to craft a `HandMadeLightClientUpdate`, the relayer will repeat the submission cycle with the same failing update and be stalled.

Recommendations

`eth2_to_near_relay` should check that the sum of `sync_committee_bits` on a `HandMadeLightClientUpdate` is greater than two-thirds of the sync committee, handling the error to avoid sending failing updates to the `eth2-client` or DAO contract.

Additionally, `send_specific_light_cleint_update()` should return a result indicating transaction success or failure. The calling logic in `send_hand_made_light_client_update()` should handle the error accordingly to account for this and other possible error cases. In other error cases, such as a timeout, it would not be desirable to increment `current_gap_between_finalized_and_attested_slot`.

Resolution

The implemented solution iterates through the chain searching for an attested header where the following block (i.e. the signature block) has more than the required number of signatures in the `SyncAggregate`. This solution ensures that the selected attested header has been signed by a sufficient number of the sync committee members before submission.

The solution is implemented in PR #800 and later patched in PR #833 to account for the case where there is numerous skip lost after an attested header.

RBE2-09	Misguided Mainnet Safety Checks		
Asset	`rainbow-bridge/contracts/near/eth2-client`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The safety checks on `network` argument of `init()` are intended to prevent a client deployed on the NEAR mainnet network from implementing an insecure configuration with critical security features disabled, while allowing such configurations on NEAR testnet deployments.

It is discouraged to use conditional safety checks as it is possible for misconfigurations to disable these safety checks. Once such example is if the production code was deployed with `network = "kiln"` and `validate_udpates = false`. It would still run on mainnet but would have critical safety features disabled.

```
if network == Network::Mainnet {
    assert!(
        args.validate_updates,
        "The updates validation can't be disabled for mainnet"
    );

    assert!(
        (cfg!(feature = "bls") && args.verify_bls_signatures)
            || args.trusted_signer.is_some(),
        "The client can't be executed in the trustless mode without BLS sigs verification on Mainnet"
    );
}
```

The severity of this issue is rated low, as it would require significant misconfiguration of the `init()` function. Furthermore, the current setup requires the DAO validator clients to anlayse each `LightClientUpdate` and vote on their validity before `submit_beacon_chain_light_client_update()` may be called.

Recommendations

This issue can be mitigated by removing possibilities for insecure client configurations from the production codebase. The production contract should be assumed to always run on NEAR mainnet. Consider increasing the depth of the tests to account for the mainnet requirements.

Additionally, the `network` argument would be better named `eth_network` to avoid confusion with the NEAR network.

Resolution

The development team have opted to instead use feature flags `#[cfg(feature = "mainnet")]`, which will be included in the `default` features. This solution still leaves the potential for misconfiguration if the `maintnet` feature is not enabled during compilation. To address this CI tools are used to ensure that the built `.wasm` file has been compiled with the `mainnet` feature enabled by validating the checksum of the bytecode.

RBE2-10	Lack of Client Syncing Status Checks		
Asset	`rainbow-bridge/eth2near/eth2near-block-relay-rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `eth2_to_near_relay` connects to user-specified Ethereum consensus and execution client endpoints, as well as a NEAR node, and submits Ethereum data to the NEAR `eth2-client` contract without validating that the sources are fully synced. This may result in a range of issues, including:

1. submitting stale data to the NEAR client
2. executing unexpected logic where the NEAR finalised head is ahead of the beacon source
3. panics while processing unexpected data

Recommendations

This issue can be mitigated by checking the sync status of Ethereum consensus and execution clients upon startup and periodically during operation to avoid relaying stale data to the NEAR `eth2-client`. The appropriate API endpoints are:

1. getSyncingStatus for the Ethereum consensus client
2. eth_syncing for the Ethereum execution client
3. SyncInfo for the NEAR client

Resolution

A function named `is_syncing()` was added for each of the external RPC clients beacon_rpc_client.rs, eth1_rpc_client.rs and near_rpc_client.rs to query respective endpoints.

These are organised into a function wait_for_syncronization() which is called near the start of the relayer `run()` loop. The function sleeps for a specified time if `is_syncing()` is true for any of the clients.

RBE2-11	Relayer Does Not Account for Unfinalised Headers Submission Quota		
Asset	`rainbow-bridge/eth2near/eth2near-block-relay-rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `eth2_to_near_relay` accepts a configuration parameter `max_submitted_headers` which determines the number of unfinalised headers in a batch. These headers are submitted on-chain to the NEAR `eth2-client` in a batch transaction via `NearContractWrapper::call_change_method_batch()`, composed of calls to `submit_execution_header()`.

The execution logic of the `eth2_to_near_relay` does not account for remaining submission quota allocated to the relayer by `max_submitted_blocks_by_account` and `submitters[relayer]` in the `eth2-client`. This can result in scenarios where the `eth2_to_near_relay` attempts to send batches of size `max_submitted_headers` that exceed its remaining quota. Exceeding the quota will cause the transaction to fail.

Furthermore, it will repeatedly attempt to send the same failing transaction until some pending `unfinalized_headers` are processed as finalised and its remaining quota rises above `max_submitted_headers`.

Recommendations

This issue can be mitigated by adding logic to the `eth2_to_near_relay` that queries the NEAR `eth2-client` to calculate `remaining_headers`, and setting the batch size via `min(max_submitted_headers, remaining_headers)`.

Resolution

A resolution was implemented in PR #828 then later made obsolete by PR #882 which removed the vulnerable code.

RBE2-12	Missing `LightClientUpdate` Checks in DAO `eth2-validator`		
Asset	`eth2-to-near-client-validator`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The DAO `eth2-validator` program does not implement, or mistakenly implements, several checks defined in the Ethereum consensus light client specifications. The same checks are also noted in RBE2-13 for another component of the bridge. This allows for potentially invalid light client updates to be submitted by the `trusted_signer` to the NEAR `eth2-client`.

These checks include:

1. Verify sync committee has sufficient participants
2. Verify slot ordering
3. Verify update does not skip a sync committee period

The `eth2-validator` program applies checks on update proposals in `validate_updates::validate_light_client_update()`.

The first noted requirement is incorrectly checked by the implementation. The check should be strictly less than `MIN_SYNC_COMMITTEE_PARTICIPANTS`:

```
if sync_committee_bits_sum <= MIN_SYNC_COMMITTEE_PARTICIPANTS {
    return Err("Invalid sync committee bits sum: {}")?;
}
```

The remaining specification requirements are missing from `eth2-validator` implementation, shown in python below:

```
assert current_slot >= update.signature_slot > update.attested_header.slot >= update.finalized_header.slot
store_period = compute_sync_committee_period_at_slot(store.finalized_header.slot)
update_signature_period = compute_sync_committee_period_at_slot(update.signature_slot)
if is_next_sync_committee_known(store):
    assert update_signature_period in (store_period, store_period + 1)
else:
    assert update_signature_period == store_period
```

Recommendations

Modify `eth2-validator` to make accurate checks on each `LightClientUpdate` specification requirement, to avoid sending invalid updates to `eth2-client` via the `trusted_signer`.

Resolution

The required checks have been added in PR #4. Included in the mitigation is an update to the `MIN_SYNC_COMMITTEE_PARTICIPANTS` check to use strictly less than `<`.

Additionally, two checks were added to ensure `update.signature_slot > update.attested_header.slot` and `update.attested_header.slot >= update.finalized_header.slot`.

The new check does not confirm that `current_slot >= update.signature_slot`. However, the impact of not checking `current_slot` is negligible as the sync committee aggregate must be signed by two-thirds of the sync committee to be a valid finalised update. Each non-malicious sync committee member will reject blocks which have a slot higher than their internal `current_slot`. Therefore, unless two-thirds of the sync committee is malicious it won't be feasible to have an update where the `current_slot` is in the future.

Finally, a check was added to ensure that the `signature_slot` belongs to either the current finalised period or the next finalised period.

RBE2-13	Missing `LightClientUpdate` Checks in `eth2-client`		
Asset	`rainbow-bridge/contracts/near/eth2-client`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `eth2-client` applies several checks before accepting a `LightClientUpdate`. It does not verify the BLS signature of the sync committee because this is not possible to do on-chain in NEAR at the time of writing. This is a known issue and is currently mitigated by verifying these signatures off-chain with a trusted source.

However, several other checks required by the Ethereum light client specifications are not implemented, or implemented incorrectly by the NEAR `eth2-client`. The same checks are also noted in RBE2-12 for another component of the bridge.

The missing checks can be seen in the python reference implementation:

```
assert current_slot >= update.signature_slot > update.attested_header.slot >= update.finalized_header.slot
store_period = compute_sync_committee_period_at_slot(store.finalized_header.slot)
update_signature_period = compute_sync_committee_period_at_slot(update.signature_slot)
if is_next_sync_committee_known(store):
    assert update_signature_period in (store_period, store_period + 1)
else:
    assert update_signature_period == store_period
```

The assertion at the top line ensures that update slots do not occur in the future and that they occur in the expected order. The assertions at the bottom, within the `if/else` blocks conditioned upon `is_next_sync_committee_known()`, verify the allowed period of the sync committee signature.

The `eth2-client` applies relevant checks in the function `verify_finality_branch()`, called within `validate_light_client_update()`.

```
assert!(
    active_header.slot > self.finalized_beacon_header.header.slot,
    "The active header slot number should be higher than the finalized slot"
);

let update_period = compute_sync_committee_period(active_header.slot);
assert!(
    update_period == finalized_period || update_period == finalized_period + 1,
    "The acceptable update periods are '{}' and '{}' but got {}",
    finalized_period,
    finalized_period + 1,
    update_period
);
```

`eth2-client` does not check the current slot and signature slot. Lack of validation of `signature_slot` allows an attacker to use either the `current_sync_committee` or `next_sync_committee` to sign messages, while lack of validation of `attested_header.slot` means attested headers may have a lower slot number than the finalised header. In both cases, the attested header needs to be signed by either the `current_sync_committee` or `next_sync_commmittee`. Thus, the likelihood of this issue is considered low.

Recommendations

This issue may be resolved by adding the missing checks.

Resolution

The issue is resolved in two PRs, first #804 implements the following checks.

```
require!(
    update.attested_beacon_header.slot
        >= update.finality_update.header_update.beacon_header.slot,
    "The attested header slot should be equal to or higher than the finalized header slot"
);

require!(
    update.signature_slot > update.attested_beacon_header.slot,
    "The signature slot should be higher than the attested header slot"
);
```

Similarly to RBE2-12, the checks do no validate `current_slot`. However, this is reasoned to be or negligible security risk.

Furthermore, PR #839 adds a check to ensure the `signature_slot` belongs to either the current finalised period or the next finalised period.

RBE2-14	Client Initialisation Process
Asset	`rainbow-bridge/contracts/near/eth2-client`
Status	**Resolved:** See Resolution
Rating	Informational

Description

While RBE2-09 describes an issue related to the inaccurate validation of `init()` arguments, this issue details problems with the general process and missing validation of several parameters.

The client contract allows any account to initialise it, and initialisation is not conducted atomically with deployment. While this design allows a relay to optionally initialise the client when connecting to it, anybody can frontrun the initialisation with bad parameters.

The client does not verify the merkle proof supporting the first sync committee that it receives, as implemented in the Ethereum specifications, allowing for an arbitrary `SyncCommittee` to be accepted.

Recommendations

This issue can be mitigated by restricting client initialisation to the trusted contract admin role with `assert_self()`, thereby preventing relays and other untrusted actors from initialising the client. While this solution hardens and simplifies the client intialisation process overall, it does trivially complicate the process for the contract admin, who must provide the initial client checkpoint without a relay. This solution also accepts that an initialised client may fall out-of-sync until the first relay connects to it.

Consider adding a merkle proof to the `init()` functions that verify the `current_sync_committee` and `next_sync_committee` exist in the `finalized_beacon_header`. It is possible to use a merkle proof of the `BeaconStateRoot` inside the `BeaconHeader` to prove the sync committees are accurate. There is a gas and complexity trade-off for adding these checks on-chain and thus is an optional recommendation.

Resolution

A patch has been made to the `init()` function in PR #795. `#[private]` macro has been added to the function which ensures that `init()` can only be called by the current account. It provides the same functionality as `assert_self()`.

RBE2-15	The Package `contract_wrapper` Does Not Compile
Asset	`rainbow-bridge/contracts/near2eth/contract_wrapper`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The package `contract_wrapper` cannot be compiled independently of other packages in the workspace, generating the following error:

```
error: failed to select a version for the requirement `eth2_hashing = "^0.3.0"`
```

The `Cargo.toml` of this package specifies `eth2_hashing = "0.3.0"`, for which cargo attempts to pull the `eth2_hashing` package on `crates.io`, which is published and maintained by Sigma Prime only up to version `0.2.0` at the time of writing.

Lighthouse implements an unpublished version `0.3.0` of this package, and there is also a local package authored by Aurora in `contracts/near/eth2_hashing` implementing version `0.3.0`, which has been significantly modified for optimized usage with NEAR contracts. The latter dependency is assumed to be required.

The `contract_wrapper` package compiles when the relay package is built due to a dependency patch specified in the relay `Cargo.toml` file.

Recommendations

Minimally, the `Cargo.toml` of `contract_wrapper` should be patched similarly to that of `near2eth-block-relay-rs`:

```
[patch.crates-io]
eth2_hashing = { git = "https://github.com/aurora-is-near/lighthouse.git", \
  rev = "b624c3f0d3c5bc9ea46faa14c9cb2d90ee1e1dec" }
```

For a more robust solution, the modified package dependency should be maintained under a separate name from the parent and managed as its own package.

Resolution

The required crates have been patched in commit c308e60.

RBE2-16	Excessive RPC Timeout Configuration
Asset	`rainbow-bridge/eth2near/eth2near-block-relay-rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `BeaconRPCClient` configures a uniform blocking timeout of 180 seconds for all calls made to the source (see here). This response timeout period is excessively long for most of the API queries. If any of the queries fail to receive a response, this configuration imposes a longer period before the `eth2_to_near_relay` loop restarts, potentially stalling the `eth2_to_near_relay` unduly when an RPC source is intermittently unavailable.

```
/// Creates `BeaconRPCClient` for the given BeaconAPI `endpoint_url`
pub fn new(endpoint_url: &str) -> Self {
    Self {
        endpoint_url: endpoint_url.to_string(),
        client: reqwest::blocking::Client::builder()
            .timeout(Duration::from_secs(180))
            .build()
            .unwrap(),
    }
}
```

Note that a long timeout period is desireable for queries fetching `BeaconState`.

Recommendations

This issue may be mitigated by configuring different timeout periods for each endpoint relative to the data load.

Resolution

The issue is resolved in PR #800. The `BeaconRPCClient` was updated to include two different timeout durations. One duration is for fetching the `BeaconState` which requires transferring several hundred Mega bytes. The other duration is for all other RPC calls.

These durations are configurable and allow shorter timeouts to be used for smaller RPC calls and longer timeouts for larger RPC calls.

RBE2-17	Potential Subtraction Overflow in `send_hand_made_light_client_update()`
Asset	`rainbow-bridge/eth2near/eth2near-block-relay-rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `eth2_to_near_relay` utilises unchecked subtraction to compute the gap between the last finalised slot on the beacon chain and on the near client to conditionally send hand made light client updates when the gap is too large for normal execution (source). Although the last finalised slot is on the beacon chain is expected to be larger or equal to the NEAR client, there are edge cases where the NEAR client may have a larger slot (i.e. if the beacon source is syncing). Such cases can cause the subtraction to overflow and the `eth2_to_near_relay` may submit an invalid update.

```
if last_finalized_slot_on_eth - last_finalized_slot_on_near
        >= self.max_blocks_for_finalization
    {
        info!(target: "relay", "Too big gap between slot of finalized block on Near and Eth. Sending hand made light client
            ↪  update");
        self.send_hand_made_light_client_update(
            last_finalized_slot_on_near,
            last_submitted_slot,
        );
        return;
    }
```

There is a check immediately following the logic above that would effectively prevent such edge cases:

```
if last_finalized_slot_on_eth <= last_finalized_slot_on_near {
    info!(target: "relay", "Last finalized slot on Eth equal to last finalized slot on NEAR. Skipping sending light client
        ↪  update.");
    return;
}
```

Recommendations

This issue can be mitigated by re-ordering the two `if` blocks referenced above such that a subtraction overflow in `send_light_client_updates()` is not possible.

Resolution

The resolution for this issue can be found in PR #787 Each of the values have been cast to `i64` (from `u64`) thereby preventing negative overflows. Noting that these values are slots fetched via RPC from a trusted sources and will not be large enough to negative overflow an `i64`.

RBE2-18	Centralisation Risks
Asset	`/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The Rainbow Bridge is to act as a decentralised bridge between Ethereum and NEAR. There are some potential centralisation risks associated with the current design that could lead to one or a small number of actors exploiting the bridge for personal gain. If a user can add arbitrary blocks to the on-chain light client they would be able to make fraudulent transactions on the Rainbow Bridge. The risks and their mitigations are outlined in this issue.

NEAR contracts are upgradeable if a full access key is associated with the smart contract account. Allowing the bytecode of the `eth2-client` contract to be modified would allow arbitrary blocks to be added to storage. To mitigate the upgradeability the `eth2-client` should not have a full access key associated with this account.

The `trusted_signer` can be updated via the function `update_trusted_signer()`. Access control to this function is `assert_self()`. Anyone with control of the access key to call this function may set the `trusted_signer` to any address. As a `trusted_signer` is it possible to submit malicious blocks which do not have valid BLS signatures. Ensure the `trusted_signer` is only set the SputnikDAO contract. Furthermore, consider modifying the access control of `update_trusted_signer()` to also be only callable via a timelocked trusted DAO.

The SputnikDAO access control allows groups of users to be specified for different voting functions and creating proposals. Ensure the groups are sufficiently large to avoid centralisation.

Similarly to `eth2-client` the SputnikDAO contract / `trusted_signer` should not have a full access key.

Recommendations

Ensure the centralisation risks are understood and avoided where possible.

Resolution

The centralization setup is to be solved in the future with the introduction of BLS signatures verification on-chain. On-chain signature verification will provide the same safety guarantees as any light client using the Ethereum network.

The development team have provided the following comments in relation to the current setup.

At the moment, `trusted_signer()` is indeed configured to SputnikDAO instance. That DAO has the following groups:

- *Validator group consists of 7 hot-wallets. Deployment of validators is distributed on different servers. We don't have any single entity controlling more than 1 key in the group. This group is allowed only to vote for proposals. Voting policy is 5/7.*

- *Relayer group consists of 1 relayer account. This group is allowed only to create specific proposals to the DAO.*
- *Council group consists of 7 cold-wallets. We don't have any single entity controlling more than 1 key in the group. This group is allowed to manage SputnikDAO contract, change rules, groups, etc. Voting policy is 4/8.*

RBE2-19	Miscellaneous General Comments
Asset	/*
Status	**Resolved:** See Resolution
Rating	Informational

Description

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

1. **Relay configuration input validation.**
   The `Eth2NearRelay::init()` function accepts a Config struct containing several user-specified arguments. Because relayers may be operated by non-expert users, greater degree of validation of these inputs is warranted. Ideally, inputs should be validated at the earliest interface, handled within specialized types, and return helpful error messages upon failure. Specific examples follow:
   (a) `beacon_endpoint`, `eth1_endpoint` and `near_endpoint` are parsed and stored as generic `String` types, without validation during initialisation that they adhere to URL format, that the source exists, and that it responds with expected data.
   (b) `signer_account_id`, `contract_account_id`, and `dao_contract_account_id` are handled as generic `String` types, without validation that they adhere to NEAR `AccountId` format until downstream processing in `NearContractWrapper`.
   (c) `network`, `contract_type`, and `near_network_id` are accepted as generic `String` types, but are better suited by `enum` types.
   (d) `light_client_updates_submission_frequency_in_epochs` parameter is accepted and stored as `i64` type, but is better suited by an unsigned integer value.
2. **DAO validator configuration input validation.**
   Similarly as above, the arguments accepted in `eth2-validator-crate::config::Config` are handled as generic `String` types and should utilize `Url`, `AccountId`, and `enum` with validation where appropriate.
3. **Non-idiomatic function and type imports.**
   (a) The usage of glob operators `*` for import statements throughout parts of the codebase makes it difficult to see what types are in scope and where they are defined. The Rust book specifically discourages this practice.
   (b) Several function and type imports deviate from conventions detailed in the Rust book, whereby the full path of type imports are specified in `use` statements, but function imports bring the entire parent module within scope, to call with `parent_module::function_name()`. This makes it clear that the function is not defined locally, and concisely hints toward its location.
4. **Usage of hardcoded unamed contrants.**
   `Eth2NearRelay::send_light_client_updates()` converts a frequency specified in epochs to slots multiplying by unnamed constant `32`. The named constant `ONE_EPOCH_IN_SLOTS` is defined and within scope, which should be used in this operation.
5. **Unclear or ambiguous function and variable names.**
   The naming of functions and variables should optimally convey its intent and effects with concision. Several examples could benefit from renaming, or splitting into separate functions with distinct purposes. Functions and variables names with typos are also reported here, separately from comment typos, as these are more important to highlight.

(a) `block_hash_safe()` could return a finalized or an unfinalized block.

(b) `gc_headers()` does not indicate its purpose.

(c) `validate_update::verify_finality_proof()` verifies both the finality branch and the next sync committee branch (if present); this would be best split into distinct functions.

(d) `eth2near_relay::send_specific_light_cleint_update()` has a typo.

(e) `BeaconRPCClient::get_sync_comittee_update_from_light_lient_update_json_str()` has a typo.

(f) `unfinalized_headers` can refer to unfinalized execution headers or beacon headers.

(g) `network` specifies the Ethereum network to sync to, but is mistaken to specify the NEAR network of deployment.

(h) `submitters` is a mapping of registered `sumbitter` 's to the number of unfinalized blocks they have submitted.

(i) `max_submitted_headers` refers to the headers batch size that the relay should attempt to submit, and is not coupled to `max_submitted_blocks_by_account` in the client.

(j) `light_client_updates_submission_frequency_in_epochs` actually specifies a period "N submissions per epoch" (the inverse of frequency).

(k) `sync_committee_signature` parameter of `get_sync_committee_bits()` is actually `SyncAggregate` type.

6. **Consistency of directory naming sytle.**

   `rainbow-bridge/contracts/near/eth2_hashing` uses underscores while the rest of the directories in the parent folder use hyphens.

7. **Consistency of NEAR capitalization.**

   Both "NEAR" and "Near" are used to refer to the NEAR network in error messages and comments throughout the codebase. Consistent usage would be preferable.

8. **Assertion of invariant can cause panic.**

   The method `ExecutionBlockProof::merkle_root_from_branch()` contains an assertion statement on the properties of its inputs, which will cause the `eth2_to_near_relay` to panic if it is triggered: `assert_eq!(branch.len(), depth, "proof length should equal depth");`. Although this assertion should not be triggered based on the inputs that the relayer passes, it is generally bad practice for reusable library code to cause panics. It would be more appropriate to return an error.

9. **Code simplifications.**

   (a) The method chain encoding `sync_committee_bits` into a fixed byte array is expressed as `sync_committee_signature.clone().sync_committee_bits.into_bytes().into_vec().as_slice().try_into()` (see here).
   This would be more concisely achieved with `sync_committee_signature.sync_committee_bits.as_ssz_bytes().try_into()`.

   (b) Unnecessary call to method `copied()` both here and here here, as `map()` works with a reference in both cases.

   (c) The type `Option::<SyncCommitteeUpdate>::None` (see here) can be simplified as `None` with the type inferred by the compiler.

10. **NEAR optimisations.**

    The `Eth2NearClient` contract could benefit from several recommended optimisations for NEAR contracts. More detailed examples of where these recommendations apply have been communicated with the development team and are only generally enumerated here:

    (a) Prefer `near_sdk::require()` to `assert!`: the standard assert macro implicitly introduces unnecessary string formatting.

   (b) Prefer `near_sdk::env::panic_str()` to `panic!`: the standard panic marco implicitly introduces unnecessary string formatting (also applies to `expect()`).
   (c) Remove `#[payable]` decorator from methods which do not accept NEAR tokens.
   (d) Optimise selection of `near_sdk::collections` data structures based on implementation requirements.

11. **Typos in Comments.**
    (a) Comments in `rainbow-bridge/contracts/near/eth2-utility/src/consensus.rs` on line [**198-199**] should be `HeaderInfo: 70B` and `counter: 4B` (note the colon).
    (b) "Headerss" (see here).
    (c) Missing colons in what should be "HeaderInfo: 70B" and "counter: 4B" (see here)

12. **Unwraps panic without helpful error messages in setup.**
    (a) eth2-on-near-client-validator
       - main.rs L28
       - main.rs L37
       - main.rs L54
    (b) `rainbow-bridge`
       - main.rs L55
       - main.rs L84
       - main.rs L85
       - main.rs L88
       - eth2near_relay.rs L59
       - init_contract.rs L21
       - init_contract.rs L26
       - init_contract.rs L39
       - init_contract.rs L45
       - init_contract.rs L49
       - init_contract.rs L52
       - init_contract.rs L57

13. **Rust Security Advisory Vulnerable Crates.**
    cargo audit is a tool used to check all dependant crates against the Rust Security Advisory. There are two vulnerable dependencies that are flagged by `cargo audit`.

    - `time 0.1.44`
    - `chrono 0.4.19`

    The testing team has concluded that the vulnerable code is not reachable from any of the in scope repositories. Consider updating these crates and those along the dependency tree to the most recent versions to remove the security advisory.

14. **Clippy Lints.**
    The tool clippy provides useful rust lints. Consider applying the following lints found by `clippy`.

    - unneeded return statements
       – get_last_slot()
       – linear_slot_search()
       – binary_slot_search()()
       – get_last_submitted_slot()

- reference to reference i.e. extra & for `beacon_state` which is already a pointer.
- reference to ‘Vec‘ rather than slice
- unused import

15. **Potential Panics From RPC Responses.**

    The beacon client is trusted however trust on external resources should be minimised to account for potential bugs and updates. This issue highlights potential panics that may occur if malformed data is received from an RPC call.

    These cases may result in index out of bounds errors if the result has insufficient length. Consider adding bounds check to each array or string before indexing.

    - Beacon RPC #145
    - Beacon RPC #156

    These cases may result in a panic when indexing the `Value` if the key is non-existent. Consider indexing a `Value` using the function ‘get()‘ and handling the error case.

    - Beacon RPC #155
    - Beacon RPC #179
    - Beacon RPC #217
    - Beacon RPC #251
    - Beacon RPC #265
    - Beacon RPC #273
    - Beacon RPC #281
    - Beacon RPC #291
    - Beacon RPC #308
    - Beacon RPC #348
    - Beacon RPC #351
    - Beacon RPC #387
    - Beacon RPC #392
    - Eth1 RPC #36

Recommendations

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

Resolution

The development team have implemented the suggestions where it was deemed appropriate and practical.

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

σ'