shieldify

Futaba

SECURITY REVIEW

Date: 3 December 2023

CONTENTS

1. About Shieldify

We are Shieldify Security – revolutionizing Web3 Security. Elevating standards with top-tier reports, a uniques subscription based model.

Book a security review and learn more about us at shieldify.org or @ShieldifySec

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Futaba

Futaba is a cross-chain protocol designed for efficient and secure data acquisition from other blockchain networks. It provides a seamless and minimal-trust solution for querying the states of different chains while allowing developers to customize their security models for cross-chain interactions. Futaba addresses the challenges posed by the growing diversity of blockchain networks and the loss of comparability among them by offering a method for retrieving data from other chains.

Learn more about Futaba's concept and the technicalities behind it here.

3.1. Observations

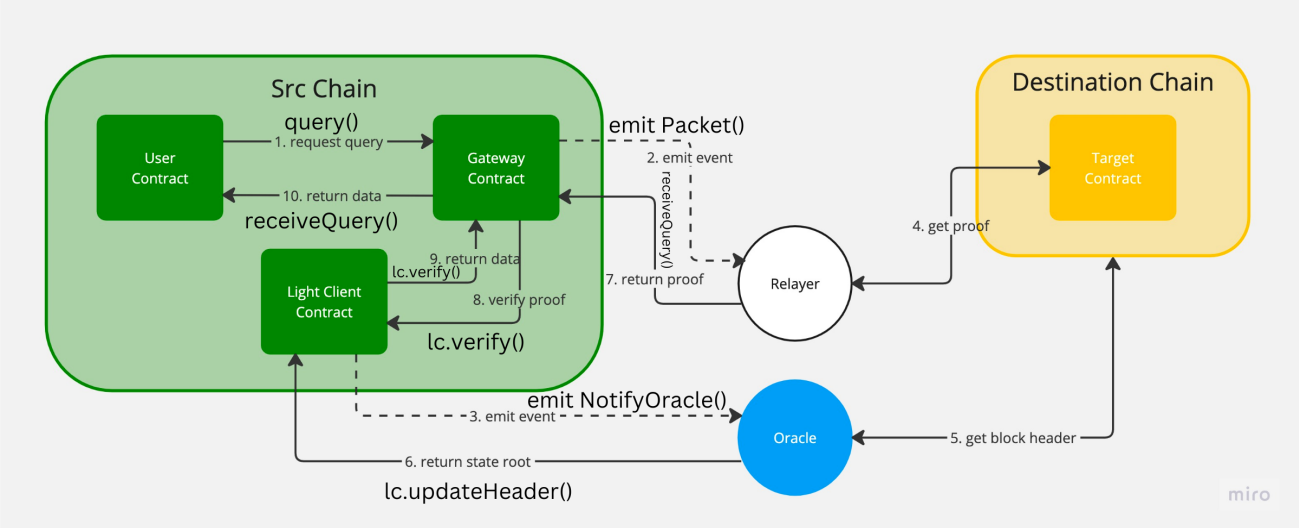

Figure 1: Futaba's Architecture

The protocol's main components are the `Gateway`, `Relayer`, `LightClient` and an optional `Oracle` (depending on the logic of the `LightClient` Contract). With them, Futaba's workflow can be divided into three main phases:

Phase 1. Request Query

In this initial phase, the `UserContract` initiates a query by paying the required fee to the `Gateway` contract. The system performs several checks to ensure the validity of addresses and encodes the query data for processing. The query data is then sent to the `LightClient` contract. Additionally, the data is formatted and passed to the Oracle for verification. An event is emitted from the `Gateway` to the `Relayer`, indicating the query request.

Phase 2. Off-chain Agents Work

This phase is split into two segments, one for Oracle work and another for `Relayer` work. The Oracle retrieves the state root of a specific block height on the destination chain via Chainlink's External Adapter. The obtained state root is sent to the Oracle Contract, and it's saved in the `LightClient` contract. Meanwhile, the `Relayer` receives the query request event, stores relevant transaction data, and initiates JSON-RPC requests for storage proof and account proof. After receiving the proofs, the `Relayer` checks if the target state root has been updated in the `LightClient` contract. If updated, the proof is sent to the `Gateway` contract, and the transaction status is updated.

Phase 3. Verify and Return Data to the User

In the final phase, the obtained proof is sent to the `LightClient` contract for verification. If the proof is valid, the data is returned to the `Gateway` contract and stored as a cache in a mapping format. The `Relayer` fee is paid, and the data is finally returned to the `UserContract`. This phase ensures that the data obtained in the query is valid and securely delivered to the user.

3.2 Privileged Roles & Actors

Gateway Contract

The Gateway is a singleton contract, equipped with endpoints for queries. It includes a `query` function, which serves as a query endpoint, and a `receiveQuery` function to collect results from Relayers. Additionally, it defines a `getCache` function for accessing already cached queries.

Relayer

Acts as an intermediary between source and destination chains receives and processes query requests from the source chain's Gateway Contract. It also obtains and sends proof for these requests. The current implementation uses Gelato but a custom relayer is planned in the future.

Light Client Contract

The Light Client is a contract responsible for proof verification and data extraction. It allows developers to define their own verification methods. As a start, verification using Patricia Merkle Tree (MPT) is used for account and storage proofs. However, the contract is designed to support future verification methods like zero-knowledge proofs or other custom implementations. It includes an interface for `requestQuery` and `verify` functions.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted three weeks and a total of 672 hours have been spent by the four smart contract security researchers from the Shieldify team.

Overall, the code is professionally written with foundational best practices for the mitigation of common vulnerabilities being implemented. Nevertheless, Shieldify's audit report contributed by identifying several Critical and High-severity issues, residing around the `verify()` function implementation, missing upgradeability mechanism of the Gateway contract, improper implementation of the `withdraw()` function, and incomplete verification of the storage roots, among other Medium and Low findings.

The test coverage can further be worked on, especially in the context of the `Gateway` contract. Futaba's excellent and fast communication was an important factor in the quality of the audit report and Shieldify would like to extend their gratitude for this.

5.1 Protocol Summary

Project Name	**Futaba**
Repository	Futaba-Contract
Type of Project	Cross-chain query infra, optimized for data acquisition from other chains
Audit Timeline	21 days
Review Commit Hash	e0e5c247bee19cf2a96926d96895b9e8c221fea3
Fixes Review Commit Hash	e155b3c86e7bb556e97eb3f9fc48de5c1f164947

5.2 Scope

The following smart contracts were in the scope of the security review:

`Futaba-Contract` GitHub repository:

Futaba-Contract/contracts/Gateway.sol	197
Futaba-Contract/contracts/ChainlinkLightClient.sol	194
Futaba-Contract/contracts/ChainlinkOracle.sol	122
Futaba-Contract/contracts/QueryType.sol	22
Futaba-Contract/contracts/interfaces/IChainlinkLightClient.sol	6
Futaba-Contract/contracts/interfaces/IGateway.sol	20
Futaba-Contract/contracts/interfaces/ILightClient.sol	11
Futaba-Contract/contracts/interfaces/IReceiver.sol	10
Futaba-Contract/contracts/lib/EthereumDecoder.sol	253
Futaba-Contract/contracts/lib/RLPReader.sol	267
Futaba-Contract/contracts/lib/RLPEncode.sol	173
Futaba-Contract/contracts/lib/TrieProofs.sol	162
Total	**1437**

`Futaba-Relayer` GitHub repository:

Futaba-Relayer/src/abi/gateway.json	520
Futaba-Relayer/src/abi/lightclient.json	464
Futaba-Relayer/src/api/api.ts	169
Futaba-Relayer/src/api/api.ts	1
Futaba-Relayer/src/client/DatabaseClient.ts	118
Futaba-Relayer/src/client/GelatoClient.ts	47
Futaba-Relayer/src/client/index.ts	2
Futaba-Relayer/src/config/chains.ts	42
Futaba-Relayer/src/config/index.ts	14
Futaba-Relayer/src/handler/errorHandler.ts	9
Futaba-Relayer/src/handler/eventHandler.ts	81
Futaba-Relayer/src/handler/index.ts	2
Futaba-Relayer/src/index.ts	9
Futaba-Relayer/src/logger/index.ts	45
Futaba-Relayer/src/relayer/eventTypes.ts	18
Futaba-Relayer/src/relayer/index.ts	4
Futaba-Relayer/src/relayer/listener.ts	70
Futaba-Relayer/src/relayer/parser.ts	24
Futaba-Relayer/src/relayer/relay.ts	36
Futaba-Relayer/src/types/index.ts	98
Futaba-Relayer/src/util/helper.ts	42
Futaba-Relayer/src/util/index.ts	2
Futaba-Relayer/src/util/proof.ts	103
Total	**1920**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Critical** and **High** issues: **3**
- **Medium** issues: **4**
- **Low** issues: **10**
- **Gas** and **Informational** issues: **22**

ID	Title	Severity
[C-01]	Logic Flaw in `verify()` Function in `ChainlinkLightClient.sol` Contract	Critical
[C-02]	Protocol Misses Upgradability Mechanism, Hindering the Proper Implementation of The Custom Relayer	Critical
[H-01]	The `Gateway.receiveQuery()` Function Will Revert if the Owner Calls `Gateway.withdraw()` Function Before it	High
[M-01]	`requestQuery()` and `verify()` Functions Lack Proper Access Control	Medium
[M-02]	Insufficient Input Data Validation For `Gateway.query()` Function	Medium
[M-03]	The `Gateway.query()` Function Does Not Return Excess Fees for Executing `Query` Request	Medium
[M-04]	Missing Check For `callback` and `lightClient` Variables to be Contract Addresses	Medium
[L-01]	Using `uint32` in `QueryRequest` to Define the Destination `chainId` Could Not Be Sufficient for Queries to Newly-Released EVM-Compatible Chains	Low
[L-02]	`ChainlinkLightClient.requestQuery()` Will Revert if The Owner Forget to Executed `setOracle()` Function	Low
[L-03]	Unnecessary Declaration of `AbiencoderV2`	Low
[L-04]	Improper RLP Encoding Validation in The `RLPReader` Library	Low
[L-05]	No Validation Of Node Operator `fee` Input Data	Low
[L-06]	`ChainlinkLightClient.sol` Whitelisting Logic Should be Removed	Low
[L-07]	Missing Zero Address Checks	Low
[L-08]	Missing `bytes32(0)` Check For `_jobid` in `ChainlinkOracle.sol` Constructor And `setJobId()`	Low
[L-09]	`withdraw()` Does Not Follow The `Checks-Effects-Interactions` (CEI) Pattern	Low
[L-10]	Ownership Role Transfer Function Implement `Single-Step Role` Transfer	Low
[I-01]	Missing Event Emissions	Informational
[I-02]	Unused Imports Affect Readability	Informational
[I-03]	Use Locked Solidity Version Pragma	Informational
[I-04]	Mixed Use of Custom Errors and Revert Strings	Informational
[I-05]	Missing Error Messages in `require()` Statements	Informational
[I-06]	Update External Dependency to the Latest Version	Informational

[I-07]	Missing/Incomplete NatSpec Comments	Informational
[I-08]	Use Named Imports Instead of Plain Imports	Informational
[I-09]	Create a Modifier Only if it will be Used in More than One Place	Informational
[I-10]	Choose the Proper Functions Visibility	Informational
[I-11]	Complete the `TODO` regarding the `NatSpec`	Informational
[I-12]	Remove Testing Logging	Informational
[I-13]	All Contracts Are Missing License	Informational
[I-14]	Unit Tests are Incomplete in Futaba Relayer	Informational
[I-15]	Remove `address(0)` Check for `ChainlinkOracle.fulfill()`	Informational
[I-16]	Typos	Informational
[G-01]	No Need to Initialize Variables with Default Values	Gas Optimization
[G-02]	Cache Array Length Outside of Loops	Gas Optimization
[G-03]	Splitting `require()` Statements that Use `&&` Saves Gas	Gas Optimization
[G-04]	Use Assembly to Check for `address(0)`	Gas Optimization
[G-05]	Remove Internal `_getQueryStatus()` Function For Save Gas	Gas Optimization
[G-06]	Using `uint256` Instead `int64` For `Gateway.nonce`	Gas Optimization

7. Findings

[C-01] Logic Flaw in `verify()` Function in `ChainlinkLightClient.sol` Contract

Severity

Critical Risk

Description

The goal of this function is to validate the receipt of the query. There are two scenarios depending on if the `account.storageRoot` is stored in the `approvedStorageRoots` mapping:

1. When it is stored, the account proof verification is skipped and only the storage proof is verified.
2. When it is not stored in the `approvedStorageRoots` mapping, only the account proof is verified.

However, the storage root corresponding to the query is checked if it exists in the `approvedStorageRoots` mapping, but actually, this mapping is never updated, which results in Case-1 becoming inaccessible. This leads to omitting the storage proof verification check every time the `Gateway.sol` contract calls `verify()` function on the `ChainlinkLightClient.sol` contract. Without this check wrong data might be passed to the `UserContract.sol` through the `Gateway.sol` contract.

Location of Affected Code

File: contracts/ChainlinkLightClient.sol#L191

```
function verify(
  bytes memory message
) public view returns (bool, bytes[] memory) {
  .
  .
  .
  else {
// Account proof verification
    EthereumDecoder.Account memory account = EthereumDecoder
      .toAccount(
          accountProof.proof.verify(
            approvedStateRoots[proof.dstChainId][proof.height],
            keccak256(abi.encodePacked(accountProof.account))
          )
      );

// If the account proof is successfully verified, the storage root that
   can be obtained from it is stored in the mapping.
    storageRoot = account.storageRoot;

// Storage proof verification
    bytes memory result;
    for (uint j; j < storageProofSize; j++) {
      StorageProof memory storageProof = storageProofs[j];
      bytes32 value = getStorageValue(storageProof);
      result = bytes.concat(result, value);
    }
    results[i] = result;
  }
}
```

Recommendation

- The `approvedStorageRoots` mapping should be updated in the case where `account.storageRoot` is not set in the mapping and account proof verification passed successfully.
- The NatSpec commented upfront that the check is misleading and should be corrected in such a way that storage root instead of state root is checked.

```
function verify(
  bytes memory message
- ) public view returns (bool, bytes[] memory) {
+ ) public returns (bool, bytes[] memory) {
  .
  .
  .
- // Check if there is a corresponding state root for each query
+ // Check if there is a corresponding storage root for each query
  .
  .
  .
  else {
// Account proof verification
    EthereumDecoder.Account memory account = EthereumDecoder
      .toAccount(
          accountProof.proof.verify(
            approvedStateRoots[proof.dstChainId][proof.height],
            keccak256(abi.encodePacked(accountProof.account))
          )
      );

// If the account proof is successfully verified, the storage root that
   can be obtained from it is stored in the mapping.
-  storageRoot = account.storageRoot;
+  approvedStorageRoots[proof.dstChainId][proof.height][accountProof.
   account] = account.storageRoot;

// Storage proof verification
    bytes memory result;
    for (uint j; j < storageProofSize; j++) {
      StorageProof memory storageProof = storageProofs[j];
      bytes32 value = getStorageValue(storageProof);
      result = bytes.concat(result, value);
    }
    results[i] = result;
  }
}
```

Team Response

Acknowledged and fixed as proposed.

[C-02] Protocol Misses Upgradability Mechanism, Hindering the Proper Implementation of The Custom Relayer

Severity

Critical Risk

Description

The current implementation of the `Gateway.sol` contract restricts the reception of queries exclusively to the Gelato relayer. This is not in alignment with the protocol's broader vision, as stated in the documentation, which includes the intention to develop an independent relayer in the future.

This limitation stems from the `onlyGelatoRelayERC2771` modifier imposed on the `receiveQuery()` function, which mandates interactions exclusively from the designated Gelato relayer. Additionally, the current contract lacks upgradeability, rendering it immutable and impervious to any form of modification.

Furthermore, the `estimateFee()` function's current implementation returns 0. Since there is no corresponding setter function for the fee, there will not be a possibility for it to be changed, upon implementing the custom relayer.

This modification aligns with the protocol's future goals and lays the groundwork for the potential development of an independent relayer.

Location of affected code

File: contracts/Gateway.sol#L206

```
function receiveQuery(
  QueryType.QueryResponse memory response
) external payable onlyGelatoRelayERC2771 {
```

Recommendation

Exploring upgradeability options for the `Gateway.sol` contract is a prudent step towards enabling future adjustments and enhancements. This strategy not only aligns with the protocol's long-term vision but also ensures the contract's adaptability and scalability in response to evolving requirements.

While weighing the advantages and disadvantages, it's crucial to approach the topic with diligence. We would recommend sticking to the UUPS upgradeability pattern. For further insights, you may refer to these insightful articles: Proxies - UUPSUpgradeable and Using uups proxy pattern upgrade smart contracts.

Team Response

Acknowledged, fixed by adding a custom relayer and UUPS upgradeability pattern.

[H-01] The `Gateway.receiveQuery()` Function Will Revert if the Owner Calls `Gateway.withdraw()` Function Before it

Severity

High Risk

Description

`Gateway.sol` contract has a `query()` function that executes a query as a query endpoint and a `receiveQuery()` function that receives the result from Relayer.

What the `query()` function does:

1. Encode the query data
2. Calculate the query ID
3. Execute the light client process as necessary - (Currently, sending a request to Chainlink's Node Operator)
4. Emit a request event
5. Pay fees to relayer (Calculate the transaction fee off-chain and send the token based on it)

What the `receiveQuery()` function does:

1. Verify if Gelato's relayer executed the function
2. Verify the proof and extract the acquisition data - (Done through the interface by the `LightClient.sol` contract)
3. Storing query results in mapping
4. Pay fees to the relayer
5. Return data to the user

About Relayers:

Relayers act as intermediaries between the source chain and the destination chain. Their main functions are as follows:

1. Monitor and receive query requests emitted from the `Gateway.sol` contract on the source chain
2. Obtain proof based on the query requests (Use `eth_getProof` to obtain an account proof and a storage proof)
3. Send the proof as a transaction to the `Gateway.sol` Contract on the source chain

Relayers use Gelato's `callWithSyncFeeERC2771` service.

This is a feature that replaces the relayer's role in sending transactions. Its main functions are as follows:

1. Execute transactions on behalf of the relayer
2. Estimate the fees to be paid to `Gelato`

The advantages of using this service are as follows:

1. Simplification of the fee structure (Only one line needs to be written in the contract)
2. No need to manage the relayer's private key (Developers only need to make relay requests to the Relayer, so they don't need to use keys)
3. Easy tracking (`Gelato` provides an API for tracking purposes)

The disadvantages are as follows:

1. Need to pay fees to `Gelato`

When paying fees using `Gelato`, funds must be stored in the `Gateway.sol` contract. This is because the fee payment needs to be made at the destination of the transaction (the `Gateway.sol` contract). Therefore, when requesting a query, it is necessary to send the fee amount from the user to the `Gateway.sol` contract.

The impact is that the user will have paid a fee for `Query` request, but the data will not be returned to `UserContract` since the `receiveQuery()` function will revert. The following scenario can happen:

1. Alice wants to make a query.
2. Alice calls the `query()` function from User `contract` and she pays a fee - `estimateFee()`.

- The calculation of fees: `nativeTokenAmount = nativeTokenAmount + msg.value;`.

3. At the same time, the Owner of the `Gateway.sol` contract decides to call the `withdraw()` function before the callback `receiveQuery()` function is executed by `GelatoRelay` contract.

- The `withdraw()` function will set the `nativeTokenAmount` variable to zero: `nativeTokenAmount = 0;`.

4. Futaba Relayer will handle the event receive query and call `receiveQuery()` function, but it will revert because the `nativeTokenAmount` variable is equal to zero.

- The calculation of GelatoRelay fees: `nativeTokenAmount = nativeTokenAmount - _getFee();`.

Location of Affected Code

File: contracts/Gateway.sol#L265

```
// Amount of native tokens in this contract
uint256 public nativeTokenAmount;


//  * @notice This contract is an endpoint for executing query
//  * @param queries query data
//  * @param lightClient The light client contract address
//  * @param callBack The callback contract address
//  * @param message Data used when executing callback
function query(
  QueryType.QueryRequest[] memory queries,
  address lightClient,
  address callBack,
  bytes calldata message
) external payable nonReentrant {
  .
  .
  .
  nativeTokenAmount = nativeTokenAmount + msg.value;
}
```

```
//  @notice This function is an endpoint for receiving query
//  @param response query response data
function receiveQuery(
  QueryType.QueryResponse memory response
) external payable onlyGelatoRelayERC2771 {
  .
  .
  .
//  refund relay fee
  nativeTokenAmount = nativeTokenAmount - _getFee();

  _transferRelayFee();
}


//  @notice Withdraw native token from the contract
function withdraw() external onlyOwner {
  address payable to = payable(msg.sender);
  (bool sent, bytes memory data) = to.call{value: nativeTokenAmount}("");
  require(sent, "Futaba: Failed to withdraw native token");
  uint256 amount = nativeTokenAmount;
  nativeTokenAmount = 0;
  emit Withdraw(to, amount);
}
```

Recommendation

The nature of this finding is deeply related to the functionality of the custom relayer. Since it is due to be built and Shieldify Security will do several additional audits of the codebase, a feasible solution would be proposed in a subsequent security review, when the code for the custom relayer is ready.

Team Response

Acknowledged. A custom relayer will be built before fixing this finding.

[M-01] `requestQuery()` and `verify()` Functions Lack Proper Access Control

Severity

Medium Risk

Description

According to the documentation and protocol's logic, **`requestQuery()`** and **`verify()`** functions are supposed to be callable only from the **`Gateway.sol`** contract. The lack of access controls for these extremely sensitive features could be a serious problem as wrong data can easily be stored.

Location of Affected Code

File: contracts/ChainlinkLightClient.sol

```
function requestQuery(QueryType.QueryRequest[] memory queries) external {
function verify(bytes memory message) public view returns (bool, bytes[]
    memory) {
```

Recommendation

Ensure secure access control by implementing a modifier that verifies whether the `msg.sender` is the authorized `Gateway.sol` contract.

```
+ error NotAuthorized();
+ error ZeroAddressNotAllowed();

+ address immutable public GATEWAY;

+ /**
+   * @notice Constructor that sets LightClient information
+   * @param _gateway The address of the Gateway contract
+ */
+ constructor(address _gateway) {
+    if(_gateway == address(0)) revert ZeroAddressNotAllowed();

+    GATEWAY = _gateway;
+ }

+ modifier onlyGateway() {
+    if(gateway != msg.sender) revert NotAuthorized();
+    _;
+ }

+ function requestQuery(QueryType.QueryRequest[] memory queries) external
     onlyGateway {
+ function verify(bytes memory message) public onlyGateway returns (bool,
     bytes[] memory) {
```

Team Response

Acknowledged and fixed by adding a modifier and the associated constructor.

[M-02] Insufficient Input Data Validation For `Gateway.query()` Function

Severity

Medium Risk

Description

The `QueryRequest` struct and `message` inputs, which are the parameters of the `Gateway.query()` function are not sufficiently validated. The missing checks are the following:

1. `QueryRequest.dstChainId` - This is the chain ID of the destination chain and can be the zero value
2. `QueryRequest.height` - This is the height of the block when making the query can be the zero value
3. `QueryRequest.slot` - This is the key used by the `Relayer` to acquire the storage proof, it can be set to an empty value
4. `message` - This is the data used when executing callback, it can be set to an empty value

Location of Affected Code

File: contracts/Gateway.sol#L155-L160

```
function query(
  QueryType.QueryRequest[] memory queries,
  address lightClient,
  address callBack,
  bytes calldata message
) external payable nonReentrant {
  .
  .
  .
  for (uint i = 0; i < queries.length; i++) {
    QueryType.QueryRequest memory q = queries[i];
    if (q.to == address(0)) {
      revert ZeroAddress();
    }
  }
}
```

Recommendation

It is recommended to add sensible constraints and validations for all user input mentioned above.

File: contracts/Gateway.sol#L155-L160

```
+ error InvalidInputMessage();
+ error InvalidInputZeroValue();
+ error InvalidInputEmptyBytes32();

function query(
  QueryType.QueryRequest[] memory queries,
  address lightClient,
  address callBack,
  bytes calldata message
) external payable nonReentrant {
+ if (message.length == 0) revert InvalidInputMessage();
  .
  .
  .
  for (uint i = 0; i < queries.length; i++) {
    QueryType.QueryRequest memory q = queries[i];
    if (q.to == address(0)) revert ZeroAddress();
+   if (q.dstChainId == 0) revert InvalidInputZeroValue();
+   if (q.height == 0) revert InvalidInputZeroValue();
+   if (q.slot == bytes32(0)) revert InvalidInputEmptyBytes32();
  }
}
```

Team Response

Acknowledged and fixed by adding validation on **message**, **dstChainId**, **height**, and **slot** and any custom errors associated with them.

[M-03] The `Gateway.query()` Function Does Not Return Excess Fees for Executing `Query` Request

Severity

Medium Risk

Description

In the **query()** function, the contract **Gateway.sol** requires the users to pay a **estimateFee()** for executing a query.

However, it does not handle the situation where the user pays more **msg.value** than the required **estimateFee()**. Currently, if **msg.value > estimateFee()**, the contract simply accepts the payment without refunding the difference to the user.

The impact is primarily financial, as the current implementation does not return any excess ether **msg.value** paid beyond the required **estimateFee()**. Users who pay more than the required **estimateFee()** will not get their payment back.

The following scenario can happen:

1. Alice wants to make an executing query.
2. Alice calls the **query()** function from **User contract** and she sends **0.2 ether**, but the **estimateFee()** is **0.1 ether** for example.

3. The `estimateFee()` is correctly charged at `0.1 ether` and Alice makes a query request successfully, but she is not refunded the excess of `0.1 ether`.
4. The excess of `0.1 ether` price fee will go to the balance of the contract `Gateway.sol`.

Location of Affected Code

File: contracts/Gateway.sol#L165

```
function query(
  QueryType.QueryRequest[] memory queries,
  address lightClient,
  address callBack,
  bytes calldata message
) external payable nonReentrant {
  if (callBack == address(0) || lightClient == address(0)) {
    revert ZeroAddress();
  }

  if (msg.value < estimateFee(lightClient, queries)) {
    revert InvalidFee();
  }
  .
  .
  .
}
```

Recommendation

To address this vulnerability, we will propose two solutions:

1. Change the check mark for `executing query` price fee to `if (msg.value != price)`, if `msg.value` is different from `estimateFee()` the transaction should be reverted:

In this way, you guarantee the user that he will not lose his funds if he accidentally sends a `msg.value` greater than `estimateFee()` because the transaction will revert.

Example:

```
- if (msg.value < estimateFee(lightClient, queries)) revert InvalidFee();
+ if (msg.value != estimateFee(lightClient, queries)) revert InvalidFee()
   ;
```

2. Return the difference to the user via a call function:

In this way, you will return the difference to the user every time he sends a `msg.value` greater than the `estimateFee()`.

Example:

```
+ error FailedPaymentToUser();

+ uint256 excessEstimateFee = msg.value - estimateFee();

+ (bool success, ) = address(tx.origin).call{ value: excessEstimateFee
   }("");
+ if (!success) revert FailedPaymentToUser();
```

Team Response

Acknowledged and fixed as suggested in first point and implementation of `estimateFee()` function.

[M-04] Missing Check For `callback` and `lightClient` Variables to be Contract Addresses

Severity

Medium Risk

Description

In `Gateway.sol`, the `query()` function accepts `callback` and `lightClient` as address variables. The former should be the address of the contract, making the query via the Gateway contract, whereas the latter is the address of the light client, responsible for verification of the queried data, received from the destination chain.

The function contains zero address checks but misses one, validating if the addresses are actually smart contracts implementing the required interfaces (`IReceiver` and `ILightClient`). The absence of such a check could allow a user/script/UI to accept an EOA address for both variables or contracts that do not have the required functions, making it impossible to receive and validate the query.

Since the `query()` function is the core value proposition of the entire protocol, the absence of this check can seriously impact the business logic and result in significant gas costs, especially if a batch query is executed.

Location of affected code

File: contracts/Gateway.sol#L155

```
function query(
  QueryType.QueryRequest[] memory queries,
  address lightClient,
  address callBack,
  bytes calldata message
) external payable nonReentrant {
```

Recommendation

Add a check to verify that `callback` and `lightClient` implement the required interfaces `IReceiver` and `lightClient` respectively. There are several approaches towards this, but the one proposed by us is for both the `IReceiver` and `ILightClient` interfaces to inherit the `IERC165` interface as well, which would require the child contracts to implement the `supportsInterface()` function. The `supportsInterface()` function should return `true` if the interfaces are actually supported.

The `Gateway.sol`'s `query()` function should check if `callback` and `lightClient` addresses support the required interfaces. This can be achieved via a separate function in the Gateway contract, called `checkSupportedInterface()`.

Another recommendation would be to include examples `ReceiverWithIReceiverI165` and `LightClientWithILightClient165` contracts as examples in the documentation, which developers can build on.

File: `ReceiverWithReceiverI165.sol`

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.19;

import "interfaces/IReceiverI165.sol";

contract ReceiverWithIReceiver165 is IReceiverI165 {
  function receiveQuery(
    bytes32 queryId,
    bytes[] memory results,
    QueryType.QueryRequest[] memory queries,
    bytes memory message) external {

    function supportsInterface(bytes4 interfaceId) external pure returns
       (bool) {
      return interfaceId == type(IReceiverI165).interfaceId;
    }
  }
}
```

File: `LightClientWithILightClient165.sol`

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.19;

import "interfaces/ILightClientI165.sol";

contract LightClient is ILightClientI165 {

//  @notice This function is intended to make Light Client do something
   when a query request is made (mock emit events to Oracle)
//  @param queries request query data
  function requestQuery(QueryType.QueryRequest[] memory queries) external
      {}


//  @notice This function is for validation upon receipt of query(mock
   verifies account proof and storage proof)
//  @param message response query data
  function verify(
    bytes memory message
  ) external returns (bool, bytes[] memory) {}
```

```
// @notice Estimated fees to be collected on the LightClient Contract
    side
// @param queries request query data
  function estimateFee(
    QueryType.QueryRequest[] memory queries
  ) external view returns (uint256) {}

  function supportsInterface(bytes4 interfaceId) external pure returns (
     bool) {
    return interfaceId == type(ILightClientI165).interfaceId;
  }
}
```

File: `Gateway.sol`

```
bytes4 constant private _ILight_Client_I165Id = 0x7e256406;
bytes4 constant private _IReceiver_I165Id = 0x36f50db1;

error CallbackOrLightClientDontSupportInterface();

function checkSupportedInterface(address callBackAddress, address
   lightClient) internal view returns (bool) {
  return callBackAddress.supportsERC165InterfaceUnchecked(
     _IReceiver_I165Id) && LightClient.supportsERC165InterfaceUnchecked(
     _ILight_Client_I165Id);
}

function query(
  QueryType.QueryRequest[] memory queries,
  address lightClient,
  address callBack,
  bytes calldata message
) external payable nonReentrant {
  if (!checkSupportedInterface(callback, lightClient)) {
    revert CallbackOrLightClientDontSupportInterface();
  }
}
```

Team Response

Acknowledged and fixed as proposed.

[L-01] Using `uint32` in `QueryRequest` to Define the Destination `chainId` Could Not Be Sufficient for Queries to Newly-Released EVM-Compatible Chains

Severity

Low Risk

Description

`uint32` **`dstChainId`** accepts a max. value of 4 294 967 295. Although the vast majority of the chains, released to date have **`chainIds`** in the vicinity of up to a 5-digit number, there are rare cases in which the **`chainId`** is larger than the max value a `uint32` can hold. This could be further exacerbated in the future, as there is no official proposal on how to define a **`chainId`**.

Specific examples of a chain with a large **`chainId`** are Molereum, Palm, Zeniq, OneLedger's Frankenstein testnet and others, which can be found at the bottom of the list here. This could pose a hindrance when using Futaba on newly launched protocol's mainnets or testnets (the latter exacerbating the testing process.)

Location of affected code:

File: contracts/QueryType.sol#L9-L28

```
struct QueryRequest {
  uint32 dstChainId;
  address to;
// block height
  uint256 height;
// storage slot
  bytes32 slot;
}

struct OracleQuery {
  uint32 dstChainId;
  uint256 height;
}

struct OracleResponse {
  uint32 dstChainId;
  uint256 height;
// state root
  bytes32 root;
}
```

Recommendation

Change **`dstChainId`** to `uint256`.

```
struct QueryRequest {
- uint32 dstChainId;
+ uint256 dstChainId;
  address to;
// block height
  uint256 height;
// storage slot
  bytes32 slot;
}

struct OracleQuery {
- uint32 dstChainId;
+ uint256 dstChainId;
  uint256 height;
}

struct OracleResponse {
- uint32 dstChainId;
+ uint256 dstChainId;
  uint256 height;
// state root
  bytes32 root;
}
```

Team Response

Acknowledged and fixed by changing the `dstChainId` part to `uint256`.

[L-02] `ChainlinkLightClient.requestQuery()` Will Revert if The Owner Forget to Executed `setOracle()` Function

Severity

Low Risk

Description

The impact is `requestQuery()` function will always be reverted.

In case the owner of the `ChainlinkLightClient.sol` contract forgets to call `setOracle()` function to set the address of variable `oracle`, the storage variable will be initialized by default to `address(0)`. Therefore when someone calls `requestQuery()` function will always revert.

Location of Affected Code

File: contracts/ChainlinkLightClient.sol#L278-L280

```
function requestQuery(QueryType.QueryRequest[] memory queries) external {
  .
  .
  .
  bytes32 requestId = IExternalAdapter(oracle).notifyOracle(requests);

  emit NotifyOracle(requestId, oracle, abi.encode(requests));
}

function setOracle(address _oracle) public onlyOwner {
  oracle = _oracle;
}
```

Recommendation

It is recommended to pre-set the `oracle` in the constructor as follows:

```
+ error ZeroAddressNotAllowed();

+ constructor(address _oracle) {
+   if(_oracle == address(0)) revert ZeroAddressNotAllowed();

+   setOracle(_oracle);
+ }
```

Team Response

Acknowledged and fixed by adding address validation to `setOracle()` function and setting `oracle` address in the constructor.

[L-03] Unnecessary Declaration of `AbiencoderV2`

Severity

Low Risk

Description

Defining ABIEncoderV2 in Solidity version 0.8+ is redundant because it is automatically enabled by default in ^0.8.

Location of affected code

File: contracts/lib/EthereumDecoder.sol#L4

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.9;

pragma experimental ABIEncoderV2;
```

Recommendation

Consider removing `AbiEncoderV2`.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.9;

- pragma experimental ABIEncoderV2;
```

Team Response

Acknowledged and fixed by removing `AbiEncoderV2` pragma.

[L-04] Improper RLP Encoding Validation in The `RLPReader` Library

Severity

Low Risk

Description

The `EthereumDecoder.sol` contract relies on the external `RLPReader` dependency to decode `RLP-encoded` nodes within the Ethereum state trie. When it processes a byte string as an `RLPItem`, the library does not verify whether the encoded payload's length in the `RLPItem` matches the length of the raw bytes it contains.

Should the `RLPItem` have an encoded byte length that is excessively long or short, subsequent operations on it may access memory beyond the boundaries of the underlying buffer? In a broader context, given that the `Merkle` trie verifier expects all input to be in the valid `RLP-encoded` format, it's crucial to validate that potentially malicious data is correctly encoded.

Location of affected code

File: contracts/lib/RLPReader.sol#L55-L64

```
function toRlpItem(
  bytes memory item
) internal pure returns (RLPItem memory) {
  uint256 memPtr;
  assembly {
    memPtr := add(item, 0x20)
  }

  return RLPItem(item.length, memPtr);
}
```

Recommendation

Introduce a validation step within RLPReader's `.toRLPItem()` to ensure that the length of the provided argument precisely matches the expected length derived from the encoded prefix and payload. Similarly, add a check in `RLPReader.numItems()`, checking that the sum of the encoded lengths of sub-objects matches the total length of the RLP list.

Team Response

Acknowledged and fixed by adding validation to `.toRLPItem()` in the `RLPReader.sol` contract.

[L-05] No Validation Of Node Operator `fee` Input Data

Severity

Low Risk

Description

Missing checks on the `_fee` input argument for the constructor and `setFee()` function in `ChainlinkOracle.sol` contract.

The missing checks are the following:

1. Missing zero check in the constructor
2. Missing min `_fee` check in the `setFee()` function
3. Missing max `_fee` check in the `setFee()` function

Location of Affected Code

File: contracts/ChainlinkOracle.sol

```
constructor(
  address _tokenAddress,
  bytes32 _jobid,
  address _operator,
  uint256 _fee,
  address _lightClient
) ConfirmedOwner(msg.sender) {
  jobId = _jobid;
  setChainlinkToken(_tokenAddress);
  setChainlinkOracle(_operator);
  fee = _fee;
  lightClient = _lightClient;
}

function setFee(uint256 _fee) public onlyOwner {
  uint256 oldFee = fee;
  fee = _fee;

  emit SetFee(_fee, oldFee, block.timestamp);
}
```

Recommendation

Consider adding the following checks:

```
+ uint256 constant private MIN_NODE_OPERATOR_FEE = ?;
+ uint256 constant private MAX_NODE_OPERATOR_FEE = ?;

+ error NodeOperatorFeeCannotBeZero();
+ error MinNodeOperatorFee();
+ error MaxNodeOperatorFee();

constructor(
  address _tokenAddress,
- bytes32 _jobid,
+ bytes32 _jobId,
  address _operator,
  uint256 _fee,
  address _lightClient
) ConfirmedOwner(msg.sender) {
+ if (_fee == 0) revert NodeOperatorFeeCannotBeZero();

- jobId = _jobid;
+ jobId = _jobId;
  setChainlinkToken(_tokenAddress);
  setChainlinkOracle(_operator);
  fee = _fee;
  lightClient = _lightClient;
}
```

```
function setFee(uint256 _fee) public onlyOwner {
+ if (_fee < MIN_NODE_OPERATOR_FEE) revert MinNodeOperatorFee();
+ if (_fee > MAX_NODE_OPERATOR_FEE) revert MaxNodeOperatorFee();

  uint256 oldFee = fee;
  fee = _fee;

  emit SetFee(_fee, oldFee, block.timestamp);
}
```

Team Response

Fixed by setting `MIN_NODE_OPERATOR_FEE` to 0.001 ether and `MAX_NODE_OPERATOR_FEE` to 1 ether and adding a custom error regarding the upper and lower limits of fee associated with it.

[L-06] `ChainlinkLightClient.sol` Whitelisting Logic Should be Removed

Severity

Low Risk

Description

Currently, there is whitelisting logic in the `ChainlinkLightClient.sol` contract. The function `requestQuery()` can be called from whitelisted addresses only. This logic would not be used in production and it is an overhead for the unit testing since it requires the owner to whitelist every address that wants to make requests through this contract. In practice, it is useless due to the fact that the `requestQuery()` method should only be callable by the `Gateway.sol` contract as stated in a Medium finding above.

Location of Affected Code

File: contracts/ChainlinkLightClient.sol#L35

```
mapping(address => bool) public whitelist;
require(isWhitelisted(tx.origin), "Futaba: Not whitelisted");
function addToWhitelist(address[] calldata addresses) external onlyOwner
    {
function removeFromWhitelist(address[] calldata toRemoveAddresses)
    external onlyOwner {
```

Recommendation

Consider completely removing the whitelisting logic in `ChainlinkLightClient.sol` contract.

File: contracts/ChainlinkLightClient.sol

```
- // wallet => isWhitelisted
- mapping(address => bool) public whitelist;

function requestQuery(QueryType.QueryRequest[] memory queries) external {
- require(isWhitelisted(tx.origin), "Futaba: Not whitelisted");
  .
  .
  .
}

- /**
-  * @notice Add to whitelist
-  * @param addresses Addresses to add
- */
- function addToWhitelist(address[] calldata addresses) external
   onlyOwner {
-   for (uint i = 0; i < addresses.length; i++) {
-     whitelist[addresses[i]] = true;
-   }

-   emit AddWhitelist(addresses);
- }
```

```
- /**
-  * @notice Remove from whitelist
-  * @param toRemoveAddresses Addresses to remove
-  */
- function removeFromWhitelist(address[] calldata toRemoveAddresses)
   external onlyOwner {
-   for (uint i = 0; i < toRemoveAddresses.length; i++) {
-     delete whitelist[toRemoveAddresses[i]];
-   }

-   emit RemoveWhitelist(toRemoveAddresses);
- }
```

Team Response

Acknowledged and fixed by removing implementations related to whitelist.

[L-07] Missing Zero Address Checks

Severity

Low Risk

Description

The functions `setOracle()`, `setClient()`, `setLinkToken()`, `setOracle()` and `constructor(_tokenAddress, address _operator, address _lightClient)` of `ChainlinkOracle.sol` do not perform verification to ensure that no addresses provided as parameters are the zero addresses. Consequently, there is a risk of accidentally setting an eligible holder's address to the zero address, leading to unintended behavior or potential vulnerabilities in the future.

Location of Affected Code

File: contracts/ChainlinkLightClient.sol#L278

```
function setOracle(address _oracle) public onlyOwner {
```

File: contracts/ChainlinkOracle.sol

```
constructor(
  address _tokenAddress,
  bytes32 _jobid,
  address _operator,
  uint256 _fee,
  address _lightClient
) ConfirmedOwner(msg.sender) {

function setClient(address _client) public onlyOwner {

function setLinkToken(address _tokenAddress) public onlyOwner {

function setOracle(address _oracle) public onlyOwner {
```

Recommendation

Consider adding a check to ensure that the provided address is not the zero address.

File: contracts/ChainlinkOracle.sol

```
+ error InvalidInputZeroAddress();

constructor(
  address _tokenAddress,
  bytes32 _jobid,
  address _operator,
  uint256 _fee,
  address _lightClient
) ConfirmedOwner(msg.sender) {
+ if (_tokenAddress == address(0)) revert InvalidInputZeroAddress();
+ if (_operator == address(0)) revert InvalidInputZeroAddress();
+ if (_lightClient == address(0)) revert InvalidInputZeroAddress();
}
```

```
function setClient(address _client) public onlyOwner {
+ if (_client == address(0)) revert InvalidInputZeroAddress();
}

function setLinkToken(address _tokenAddress) public onlyOwner {
+ if (_tokenAddress == address(0)) revert InvalidInputZeroAddress();
}

function setOracle(address _oracle) public onlyOwner {
+ if (_oracle == address(0)) revert InvalidInputZeroAddress();
}
```

Team Response

Acknowledged and fixed by adding a custom error of `ZERO ADDRESS` to each function.

[L-08] Missing `bytes32(0)` Check For `_jobid` in `ChainlinkOracle.sol` Constructor And `setJobId()`

Severity

Low Risk

Description

The current implementation allows for `_jobId` to be empty bytes set in the constructor and `setJobId()` function, which might lead to a wrong `jobId` verification.

Location of Affected Code

File: contracts/ChainlinkOracle.sol

```
constructor(
  address _tokenAddress,
  bytes32 _jobid,
  address _operator,
  uint256 _fee,
  address _lightClient
) ConfirmedOwner(msg.sender) {

function setJobId(bytes32 _jobId) public onlyOwner {
```

Recommendation

Consider implementing a check for empty bytes.

```
+ error InvalidInputEmptyBytes32();

constructor(
  address _tokenAddress,
  bytes32 _jobid,
  address _operator,
  uint256 _fee,
  address _lightClient
) ConfirmedOwner(msg.sender) {
+ if (_jobId == bytes32(0)) revert InvalidInputEmptyBytes32();
}

function setJobId(bytes32 _jobId) public onlyOwner {
+ if (_jobId == bytes32(0)) revert InvalidInputEmptyBytes32();
}
```

Team Response

Acknowledged and fixed by adding a non-zero verification when setting `_jobId`.

[L-09] `withdraw()` Does Not Follow The `Checks-Effects-Interactions` (CEI) Pattern

Severity

Low Risk

Description

The `withdraw()` function does not follow `Checks-Effects-Interactions` (CEI) pattern. The `nativeTokenAmount` reset is performed after the ether transfers. It is recommended to always first change the state before doing external calls - while the code is not vulnerable right now due to the fact that only the owner of the protocol can withdraw and additionally, there is no potential financial loss since the whole amount of tokens is withdrawn at once, it is still a best practice to be followed mainly because it is possible that the code changes with time to partial `nativeTokenAmount` withdraw.

Location of Affected Code

File: contracts/Gateway.sol#L345

```
// @notice Withdraw native token from the contract
function withdraw() external onlyOwner {
  address payable to = payable(msg.sender);
  (bool sent, bytes memory data) = to.call{value: nativeTokenAmount}("");
  require(sent, "Futaba: Failed to withdraw native token");
  uint256 amount = nativeTokenAmount;
  nativeTokenAmount = 0;
  emit Withdraw(to, amount);
}
```

Recommendation

The CEI(Checks-Effects-Interactions) pattern should be followed in the `Gateway.withdraw()` function.

```
function withdraw() external onlyOwner {
- address payable to = payable(msg.sender);

+ uint256 withdrawAmount = nativeTokenAmount;
+ nativeTokenAmount = 0;

- (bool sent, bytes memory data) = to.call{value: withdrawAmount}("");
+ (bool success,) = payable(msg.sender).call{value: withdrawAmount}("");

- require(sent, "Futaba: Failed to withdraw native token");
+ require(success, "Futaba: Failed to withdraw native token");

- uint256 amount = nativeTokenAmount;
- nativeTokenAmount = 0;

- emit Withdraw(to, amount);
+ emit Withdraw(msg.sender, withdrawAmount);
}
```

Team Response

Acknowledged and fixed by changing `nativeTokenAmount` reset before sending tokens.

[L-10] Ownership Role Transfer Function Implement `Single-Step Role Transfer`

Severity

Low Risk

Description

The current ownership transfer process for all the contracts inheriting from `Ownable` involves the current owner calling the `transferOwnership()` function. If the nominated EOA account is not a valid account, it is entirely possible that the owner may accidentally transfer ownership to an uncontrolled account, losing access to all functions with the `onlyOwner` modifier.

Location of Affected Code

File: contracts/Gateway.sol#L23

File: contracts/ChainlinkLightClient.sol#L19

Recommendation

It is recommended to implement a two-step process where the owner nominates an account and the nominated account needs to call an `acceptOwnership()` function for the transfer of the ownership to fully succeed. This ensures the nominated EOA account is a valid and active account. This can be easily achieved by using OpenZeppelin's `Ownable2Step` contract instead of `Ownable`.

File: contracts/Gateway.sol

```
- import "@openzeppelin/contracts/access/Ownable.sol";
+ import {Ownable2Step} from "@openzeppelin/contracts/access/Ownable2Step
    .sol";

- contract Gateway is IGateway, Ownable, ReentrancyGuard,
    GelatoRelayContextERC2771 {
+ contract Gateway is IGateway, Ownable2Step, ReentrancyGuard,
    GelatoRelayContextERC2771 {
```

File: contracts/ChainlinkLightClient.sol

```
- import "@openzeppelin/contracts/access/Ownable.sol";
+ import {Ownable2Step} from "@openzeppelin/contracts/access/Ownable2Step
    .sol";

- contract ChainlinkLightClient is ILightClient, IChainlinkLightClient,
    Ownable {
+ contract ChainlinkLightClient is ILightClient, IChainlinkLightClient,
    Ownable2Step {
```

Team Response

Acknowledged and fixed by changing from `Ownable` to `Ownable2Step`.

[I-01] Missing Event Emissions

Severity

Informational

Description

It has been observed that important functionalities are missing an emitting event - `setOracle()` and `updateHeader()` functions in the `ChainlinkLightClient.sol`. For the `updateHeader()` method there is an event when the state root is updated, consider adding another one in the if statement to note that the state is up-to-date.

Events are a method of informing the transaction initiator about the actions taken by the called function. An event logs its emitted parameters in a specific log history, which can be accessed outside of the contract using some filter parameters.

Location of Affected Code

File: contracts/ChainlinkLightClient.sol

```
function updateHeader(QueryType.OracleResponse[] memory responses)
    external override onlyOracle {
function setOracle(address _oracle) public onlyOwner {
```

Recommendation

For best security practices, consider declaring events as often as possible at the end of a function. Events can be used to detect the end of the operation.

Team Response

Acknowledged and fixed by adding an event even if the state has already been saved in `updateHeader()` and adding an event when `Oracle` is registered in `setOracle()`.

[I-02] Unused Imports Affect Readability

Severity

Informational

Description

There are a few unused imports on the codebase. These imports should be cleaned up from the code if they have no purpose.

Location of Affected Code

File: contracts/Gateway.sol

```
import "@openzeppelin/contracts/utils/Strings.sol";
import "hardhat/console.sol";

import {Address} from "@openzeppelin/contracts/utils/Address.sol";
using Address for address payable;
```

File: contracts/ChainlinkOracle.sol

```
import "./interfaces/ILightClient.sol";
import "hardhat/console.sol";
```

Recommendation

Remove the unused imports.

File: contracts/Gateway.sol

```
- import "@openzeppelin/contracts/utils/Strings.sol";
- import "hardhat/console.sol";

- import {Address} from "@openzeppelin/contracts/utils/Address.sol";
- using Address for address payable;
```

File: contracts/ChainlinkOracle.sol

```
- import "./interfaces/ILightClient.sol";
- import "hardhat/console.sol";
```

Team Response

Acknowledged and fixed by removing unused imports.

[I-03] Use Locked Solidity Version Pragma

Severity

Informational

Description

Currently, version `^0.8.9` is used in the codebase. Contracts should be deployed with the same compiler version that they have been tested with. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, either an outdated compiler version that might introduce bugs or a compiler version too recent that has not been extensively tested yet.

Location of Affected Code

All smart contracts.

Recommendation

Consider locking the version pragma to the same Solidity version used during development and testing (`0.8.19`).

```
- pragma solidity ^0.8.9;
+ pragma solidity 0.8.19;
```

Team Response

Acknowledged and fixed by correcting the locked version.

[I-04] Mixed Use of Custom Errors and Revert Strings

Severity

Informational

Description

In some parts of the code, `custom errors` are declared and later used Custom Errors, while in other parts, classic revert strings are used in Require Statements.

Instead of using error strings, custom errors can be used, which would reduce deployment and run-time costs.

Location of Affected Code

Most contracts.

Recommendation

Consider using only custom errors as they are more gas-efficient.

Team Response

Acknowledged and fixed by changing to custom errors.

[I-05] Missing Error Messages in `require()` Statements

Severity

Informational

Description

When encountering transaction failures or unexpected behavior, the utilization of informative error messages is beneficial for troubleshooting exceptional conditions. Otherwise, inadequate error messages can lead to confusion and unnecessary delays during exploits or emergency situations.

Location of Affected Code

File: contracts/lib/RLPReader.sol

```
33: require(hasNext(self));
74: require(isList(self));
114: require(isList(item));
195: require(item.len == 1);
215: require(item.len == 21);
221: require(item.len > 0 && item.len <= 33);
241: require(item.len == 33);
253: require(item.len > 0);
270: require(item.len > 0);
```

Recommendation

Consider adding a descriptive reason in an error string.

Team Response

Acknowledged and fixed by changing to custom errors too.

[I-06] Update External Dependency to the Latest Version

Severity

Informational

Description

Update the versions `@openzeppelin/contracts` and `@chainlink/contracts` to be the latest in `package.json`.

Location of Affected Code

According to package.json, `@openzeppelin/contracts` is currently set to `^4.8.2` and `@chainlink/contracts` is `^0.6.1`.

Recommendation

We also recommend double-checking the versions of other dependencies as a precaution, as they may include important bug fixes.

Team Response

Acknowledged and fixed by updating the version (If there are no problems after confirming the operation including testing).

[I-07] Missing/Incomplete NatSpec Comments

Severity

Informational

Description

(`@notice`, `@dev`, `@param` and `@return`) are missing in some functions. Given that NatSpec is an important part of code documentation, this affects code comprehension, audibility, and usability.

This might lead to confusion for other auditors/developers that are interacting with the code.

Location of Affected Code

In some contacts.

Recommendation

Consider adding in full NatSpec comments for all functions where missing to have complete code documentation for future use.

Team Response

Acknowledged and fixed by adding the missing NatSpec.

[I-08] Use Named Imports Instead of Plain Imports

Severity

Informational

Description

It's possible to name the imports to improve code readability.

E.g. import `import "@openzeppelin/contracts/security/ReentrancyGuard.sol";`; can be rewritten as `import {ReentrancyGuard} from "@openzeppelin/contracts/security/ReentrancyGuard.sol";`

Location of Affected Code

Most contracts.

Recommendation

Consider using named imports.

```
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";->import {ReentrancyGuard} from "@openzeppelin/contracts/security/ReentrancyGuard.sol";

import "./interfaces/IGateway.sol"; -> import {IGateway} from "./interfaces/IGateway.sol";

import "./interfaces/ILightClient.sol";->import {ILightClient} from "./interfacesILightClient sol";

import "./interfaces/IReceiver.sol";->import {IReceiver} from "./interfaces/IReceiver.sol";

import "./QueryType.sol";->import {QueryType} from "./QueryType.sol";

import "./interfaces/IChainlinkLightClient.sol";  ->  import {IChainlinkLightClient} from "./interfaces/IChainlinkLightClient.sol";

import "./interfaces/IExternalAdapter.sol";   ->   import {IExternalAdapter} from "./interfaces/IExternalAdapter.sol";

import "./lib/TrieProofs.sol";->import {TrieProofs} from "./lib/TrieProofs.sol";

import "./lib/RLPReader.sol";->import {RLPReader} from "./lib/RLPReader.sol";

import "./lib/EthereumDecoder.sol";->import {EthereumDecoder} from "./lib/EthereumDecoder.sol";

import "@chainlink/contracts/src/v0.8/ConfirmedOwner.sol"; -> import {ConfirmedOwner} from "@chainlink/contracts/src/v0.8/ConfirmedOwner.sol";
```

Team Response

Acknowledged and fixed by modifying the import statements.

[I-09] Create a Modifier Only if it will be Used in More than One Place

Severity

Informational

Description

There is no need to create a separate modifier unless it will be used in more than one place. If this is not the case, simply add the modifier code to the function instead.

Location of Affected Code

File: contracts/ChainlinkOracle.sol#L205-L211

```
modifier onlyLightClient() {
  require(
    msg.sender == lightClient,
    "Futaba: only light client can call this function"
  );
  _;
}
```

File: contracts/ChainlinkLightClient.sol#L336-L339

```
modifier onlyOracle() {
  require(msg.sender == oracle, "Futaba: onlyOracle - not oracle");
  _;
}
```

Recommendation

Add the modifier logic into the functions directly.

Team Response

Acknowledged and fixed by removing modifier and adding a process to perform verification at the beginning of the function.

[I-10] Choose the Proper Functions Visibility

Severity

Informational

Description

It is best practice to mark functions that are not called internally as `external` instead of `public` and `private` instead of `internal` when they should only be called inside the given contract.

Location of Affected Code

File: contracts/ChainlinkLightClient.sol

```
139: function verify(
277: function setOracle(address _oracle) public onlyOwner {
281: function getOracle() public view returns (address) {
```

File: contracts/ChainlinkOracle.sol

```
134: function fulfill(
147: function setClient(address _client) public onlyOwner {
153: function getClient() public view returns (address) {
157: function setLinkToken(address _tokenAddress) public onlyOwner {
164: function getLinkToken() public view returns (address) {
168: function setOracle(address _oracle) public onlyOwner {
175: function getOracle() public view returns (address) {
179: function setJobId(bytes32 _jobId) public onlyOwner {
186: function getJobId() public view returns (bytes32) {
190: function setFee(uint256 _fee) public onlyOwner {
197: function getFee() public view returns (uint256) {
318: function getStorageValue(StorageProof memory storageProof) internal
   pure returns (bytes32) {
335: function checkRoot(Proof[] memory proofs) internal view {
```

Recommendation

- Consider using `external` modifier for clarity's sake if the function is not called inside the contract.
- Consider using `private` modifier for clarity's sake if the function can only be called inside the given contract.

Team Response

Acknowledged and fixed by changing the modifier of the function in question from `public` to `external` and from `internal` to `private`.

[I-11] Complete the `TODO` regarding the `NatSpec`

Severity

Informational

Description

NatSpec is an important part of code documentation, this affects code comprehension, audibility, and usability.

This might lead to confusion for other auditors/developers that are interacting with the code.

Location of Affected Code

File: contracts/Gateway.sol#L20

```
// #TODO: Add @notice & @param description for each: FUNCTION + EVENT +
   ERROR declaration
```

File: contracts/ChainlinkLightClient.sol

```
function updateHeader(
function setOracle(address _oracle) public onlyOwner {
function getOracle() public view returns (address) {
```

File: contracts/lib/EthereumDecoder.sol

File: contracts/lib/TrieProofs.sol

Recommendation

Consider adding full NatSpec comments for all functions where missing to have complete code documentation for future use.

Team Response

Acknowledged and fixed by adding the missing NatSpec.

[I-12] Remove Testing Logging

Severity

Informational

Description

The smart contract imports the Hardhat console library using import `hardhat/console.sol;`. While the Hardhat console is useful for debugging during the development and testing phases, it is generally not recommended to be included in the production code. The inclusion of debugging tools in production can lead to additional gas costs.

Location of Affected Code

File: contracts/ChainlinkOracle.sol#L10

```
import "hardhat/console.sol";
```

File: scripts/getProof.ts

Recommendation

To maintain a clean, secure, and efficient production environment, it is recommended to exclude the Hardhat console import from the final version of the contract.

File: contracts/ChainlinkOracle.sol#L10

```
- import "hardhat/console.sol";
```

File: scripts/getProof.ts

```
- console.log("balance:", balance);
- // console.log("accountProof:", accountProof);
- // console.log("storageProofs:", storageProofs);
- // console.log("proof:", proof);
```

Team Response

Acknowledged and fixed by removing console import statement and `console.log()`.

[I-13] All Contracts Are Missing License

Severity

Informational

Description

It seems like there is an issue with the licenses for the Smart Contracts in the audit scope. It is crucial to have proper licensing in place to ensure that the code is used and distributed legally.

Location of Affected Code

All smart contracts.

Recommendation

To address this, you should consider adding an appropriate license to the smart contracts. Remember to consult with a legal professional if you have specific legal concerns or if you're unsure which license is best for your project.

Team Response

Acknowledged and fixed by adding a specific license (to be discussed, as it has not yet been decided).

[I-14] Unit Tests are Incomplete in Futaba Relayer

Severity

Informational

Description

The absence of unit tests for essential components like the `Database Client`, `Listener`, and `Relay` is a concern in the `Futaba Relayer`. To ensure the system's integrity and performance, it is imperative that comprehensive testing is implemented.

Location of Affected Code

File: client/DatabaseClient.test.ts

File: handler/eventHandler.test.ts

File: relayer/listener.test.ts

File: relayer/relay.test.ts

Recommendation

Consider implementing thorough unit tests for the `Database Client`, `Listener`, and `Relay` components to ensure the system's stability and security.

Team Response

Acknowledged and fixed by adding Relayer tests.

[I-15] Remove `address(0)` Check for `ChainlinkOracle.fulfill()`

Severity

Informational

Description

Remove this check if you decide to implement a zero address check in the `constructor` and `setClient()` functions as proposed above in `L-07` finding.

Location of Affected Code

File: contracts/ChainlinkOracle.sol#L142

```
require(lightClient != address(0), "Futaba: invalid ligth client");
```

Recommendation

It is recommended to remove zero address check for `lightClient.sol` contract.

```
- require(lightClient != address(0), "Futaba: invalid ligth client");
```

Team Response

Acknowledged and fixed by removing the require statement regarding the `lightClient` address in `fulfill()` function.

[I-16] Typos

Severity

Informational

Description

There are a few typos in the contract source code. This could result in unforeseeable issues in the future development cycles.

Location of Affected Code

`ligth` -> `light`

`Depolyed` -> `Deployed`

`transaxtion` -> `transaction`

`exsit` -> `exist`

`transaxtion` -> `transaction`

`orcale` -> `oracle`

Recommendation

It is recommended to fix typos in smart contracts so that the code is clearer and more understandable.

Team Response

Acknowledged and fixed by finding and correcting the typos section to see if there are others as well.

[G-01] No Need to Initialize Variables with Default Values

Severity

Gas Optimization

Description

If a variable is not set/initialized, the default value is assumed (`0`, `false`, `0x0` …depending on the `data type`). Saves `8` `gas` per instance.

Location of Affected Code

File: contracts/ChainlinkLightClient.sol

```
247: for (uint i = 0; i < addresses.length; i++) {
261: for (uint i = 0; i < toRemoveAddresses.length; i++) {
321: for (uint i = 0; i < proofs.length; i++) {
```

File: contracts/Gateway.sol

```
169: for (uint i = 0; i < queries.length; i++) {
242: for (uint i = 0; i < results.length; i++) {
305: uint256 highestHeight = 0;
```

File: contracts/lib/EthereumDecoder.sol

```
153: for (uint256 i = 0; i < log.topics.length; i++) {
169: for (uint256 i = 0; i < receipt.logs.length; i++) {
192: for (uint256 i = 0; i < list.length; i++) {
215: for (uint256 i = 0; i < list.length; i++) {
```

File: contracts/lib/RLPEncode.sol

```
157: for (uint j = 0; j < res.length; j++) {
```

File: contracts/lib/RLPReader.sol

```
121: for (uint256 i = 0; i < items; i++) {
286: uint256 count = 0;
394: for (uint i = 0; i < idx; i++) {
```

File: contracts/lib/TrieProofs.sol

```
34: uint256 pathOffset = 0; // Offset of the proof
43: for (uint256 i = 0; i < proof.length; i++) {
176: uint nibblesLength = 0;
220: uint256 i = 0;
```

Recommendation

Do not initialize variables with their default values.

```
-  for (uint256 i = 0; ...
+  for (uint256 i; ...
```

Team Response

Acknowledged and fixed by initialization of indexes in for-loop statements.

[G-02] Cache Array Length Outside of Loops

Severity

Gas Optimization

Description

In the absence of caching, the Solidity compiler will consistently retrieve the array's length in every iteration. Specifically, for storage arrays, this entails an additional **`sload`** operation (resulting in 100 extra gas for each iteration, excluding the first one), while for memory arrays, it leads to an additional **`mload`** operation (resulting in 3 extra gas for each iteration, excluding the first one).

Location of Affected Code

File: contracts/ChainlinkLightClient.sol

```
208: for (uint i; i < responses.length; i++) {
247: for (uint i = 0; i < addresses.length; i++) {
261: for (uint i = 0; i < toRemoveAddresses.length; i++) {
321: for (uint i = 0; i < proofs.length; i++) {
```

File: contracts/Gateway.sol

```
169: for (uint i = 0; i < queries.length; i++) {
242: for (uint i = 0; i < results.length; i++) {
```

File: contracts/lib/EthereumDecoder.sol

```
153: for (uint256 i = 0; i < log.topics.length; i++) {
169: for (uint256 i = 0; i < receipt.logs.length; i++) {
192: for (uint256 i = 0; i < list.length; i++) {
215: for (uint256 i = 0; i < list.length; i++) {
```

File: contracts/lib/RLPEncode.sol

```
157: for (uint j = 0; j < res.length; j++) {
204: for (i = 0; i < _list.length; i++) {
214: for (i = 0; i < _list.length; i++) {
```

File: contracts/lib/TrieProofs.sol

```
43: for (uint256 i = 0; i < proof.length; i++) {
221: for (i = 0; i + xsOffset < xs.length && i < ys.length; i++) {
```

Recommendation

To optimize gas costs, it is recommended to instantiate a variable in every function that has a for loop, before the loop itself.

```
+  uint responsesLength = responses.length;
+  for (uint256 i; i < responsesLength; i++) {
-  for (uint256 i; i < responses.length; i++) {
...
}
```

Team Response

Acknowledged and fixed as proposed.

[G-03] Splitting `require()` Statements that Use `&&` Saves Gas

Severity

Gas Optimization

Description

Instead of using the `&&` operator in a single `require` statement to check multiple conditions, using multiple require statements with 1 condition per require statement will save 8 GAS per &&. The gas difference would only be realized if the `revert` condition is met.

Location of Affected Code

File: contracts/lib/RLPReader.sol#L221

```
require(item.len > 0 && item.len <= 33);
```

Recommendation

Instead of using the `&&` operator in a single require statement to check multiple conditions, use multiple `require` statements with 1 condition per `require` statement.

Team Response

Acknowledged and fixed.

[G-04] Use Assembly to Check for `address(0)`

Severity

Gas Optimization

Description

Use assembly to check for `address(0)` to make the gas fees lower.

Location of Affected Code

Most smart contracts.

Recommendation

It is recommended to create a helper function that checks if the address is `address(0)` and use it in all the functions that are doing the `if(address(0))` check to reduce gas costs.

```
+ function _assemblyOwnerNotZero(address _addr) private pure {
+   assembly {
+     if iszero(_addr) {
+       mstore(0 x00 , "Zero address")
+       revert(0 x00 , 0 x20 )
+     }
+   }
+ }
```

Team Response

Acknowledged and fixed by adding helper function for `address(0)`.

[G-05] Remove Internal `_getQueryStatus()` Function For Save Gas

Gas Optimization

Description

Remove the `_getQueryStatus()` function as there is no need for another internal function to read the `queryStore` mapping store.

Location of Affected Code

File contracts/Gateway.sol#L353-L357

```
function _getQueryStatus(
  bytes32 queryId
) internal view returns (QueryStatus) {
  return queryStore[queryId].status;
}
```

Recommendation

It is recommended to remove the internal `_getQueryStatus()` function.

```
function getQueryStatus(bytes32 queryId) external view returns (
    QueryStatus) {
- return _getQueryStatus(queryId);
+ return queryStore[queryId].status;
}

- function _getQueryStatus(bytes32 queryId) internal view returns (
    QueryStatus) {
-   return queryStore[queryId].status;
- }
```

Team Response

Acknowledged and fixed by deleting `_getQueryStatus()` function.

[G-06] Using `uint256` Instead `int64` For `Gateway.nonce`

Severity

Gas Optimization

Description

Using smaller `uint` variables outside of structs is not cost-efficient, as the EVM performs some additional operations to pad the data. There could also be the hypothetical case of the `query()` function reverting, because of overflow of the `nonce` variable, which is incremented outside of an `unchecked` box. The probability of this happening is almost zero to none, as the max value of `uint64` is `18,446,744,073,709,551,615` and it is big enough for plenty of queries to be made. This will be mitigated to the maximum if `nonce` is changed to `uint256` instead.

Location of Affected Code

File: contracts/Gateway.sol#L30

```
uint64 public nonce;
```

Recommendation

Change `nonce` to `uint256`.

```
- uint64 public nonce;
+ uint256 public nonce;
```

Team Response

Acknowledged and fixed by changing `nonce` type from `uint64` to `uint256`.

shieldify

Thank you!