Gas Optimizations for Axiom Contracts Security Review

Auditors

Desmond Ho, Lead Security Researcher

Riley Holterhus, Lead Security Researcher

Blockdev, Security Researcher

Lucas Goiriz, Junior Security Researcher

David Chaparro, Junior Security Researcher

Report prepared by: Lucas Goiriz

January 17, 2024

Contents

1 About Spearbit **2**

2 Introduction **2**

3 Risk classification **2**
3.1 Impact . . . 2
3.2 Likelihood . . . 2
3.3 Action required for severity levels . . . 2

4 Executive Summary **3**

5 Findings **4**
5.1 Gas Optimization . . . 4
5.1.1 Extend `unchecked` blocks . . . 4
5.1.2 Boolean expression optimizations . . . 4
5.1.3 Unnecessary function arguments in `_sendQuery()` . . . 5
5.1.4 Replace zeroing-out variables for the `delete` keyword . . . 6
5.1.5 Set `axiomCoreAddress` as `immutable` . . . 7
5.1.6 Caching storage variables to save `SLOAD`s . . . 7
5.1.7 Move variable declaration outside of `for` loop . . . 11
5.1.8 Simplify `excessivelySafeCall()` for efficiency . . . 12
5.2 Informational . . . 13
5.2.1 Missing Natspec comments . . . 13
5.2.2 Change wording for `overrideAxiomQueryFee` documentation . . . 13
5.2.3 `dataQuery` argument can be commented . . . 13
5.2.4 Redundant import . . . 14
5.2.5 Typos . . . 14
5.2.6 Replace manual hours to seconds conversion by `hours` keyword . . . 14
5.2.7 Document theoretical overflow . . . 15
5.2.8 Variable renaming . . . 15
5.2.9 Revert behavior in `_recordDeposit()` when `depositAmount > MAX_DEPOSIT_SIZE` has been lost . . . 15

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Axiom gives smart contracts trustless access to the entire history of Ethereum and arbitrary ZK-verified compute over it. Developers can send on-chain queries into Axiom, which are trustlessly fulfilled with ZK-verified results sent in a callback to the developer's smart contract. This allows developers to build rich on-chain applications without additional trust assumptions.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of axiom-v2-contracts-working according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 3 days in total, Axiom engaged with Spearbit to perform gas optimizations to the Axiom V2 protocol in axiom-v2-contracts-working. In this period of time a total of **16** issues were found.

Summary

Project Name	Axiom
Repository	axiom-v2-contracts-working
Commit	64c428...c813f0
Type of Project	Data availability, ZK
Audit Timeline	Dec 18 to Dec 20

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	0	0	0
Gas Optimizations	8	8	0
Informational	9	9	0
Total	**17**	**17**	**0**

5 Findings

5.1 Gas Optimization

5.1.1 Extend `unchecked` blocks

Severity: Gas Optimization

Context: AxiomV2Query.sol#L345-L350, MerkleMountainRange.sol#L44-L49, MerkleMountainRange.sol#L205-L220, MerkleMountainRange.sol#L105-L127

Description: Certain `unchecked` blocks can be safely extended to cover more areas beyond their initial coverage. This results in an improvement in code readability and in some scenarios, even gas savings.

Recommendation: Consider applying the following changes:

- AxiomV2Query.sol#L345-L350 (overall gas change: -96337):

```
- uint256 deposit;
  unchecked {
      // in this branch, we know that msg.value > increaseAmount
-     deposit = msg.value - increaseAmount;
+     _recordDeposit(msg.sender, msg.value - increaseAmount); // from the context, the meaning is not
↪  lost that the deposit is computed via msg.value - increaseAmount
  }
- _recordDeposit(msg.sender, deposit);
```

- MerkleMountainRange.sol#L44-L49:

```
  unchecked {
      out.peaks[i - paddingDepth] = self.peaks[i];
- }
- unchecked {
      ++i;
  }
```

- MerkleMountainRange.sol#L205-L220:

  The entire function body of `getCompleteLeaves` can be encapsulated in a single `unchecked` block.

- MerkleMountainRange.sol#L105-L127:

  The entire function body of `appendLeaf` can be encapsulated in a single `unchecked` block.

Axiom: Addressed in PR 180.

Spearbit: Verified that PR 180 implements the recommendations.

5.1.2 Boolean expression optimizations

Severity: Gas Optimization

Context: AxiomV2Query.sol#L756-L757, AxiomV2Query.sol#L763-L764, AxiomAccess.sol#L61

Description: Some boolean expressions can be simplified into shorter versions, resulting in gas savings as less boolean operation opcodes are invoked during the computation.

Recommendation: Consider applying the following simplifications:

- AxiomV2Query.sol#L756-L757 (overall gas change: -235280):

```
- !hasRole(PROVER_ROLE, msg.sender) && !hasRole(PROVER_ROLE, address(0))
-     && !perQueryProvers[proofData.querySchema][target][msg.sender]
+ !(hasRole(PROVER_ROLE, msg.sender) || hasRole(PROVER_ROLE, address(0))
+     || perQueryProvers[proofData.querySchema][target][msg.sender])
```

- AxiomV2Query.sol#L763-L764 (overall gas change: -96499):

```
- !aggregateVkeyHashes[aggregateVkeyHash]
-     && !perQueryAggregateVkeyHashes[proofData.querySchema][target][aggregateVkeyHash]
+ !(aggregateVkeyHashes[aggregateVkeyHash]
+     || perQueryAggregateVkeyHashes[proofData.querySchema][target][aggregateVkeyHash])
```

- AxiomAccess.sol#L61 (overall gas change: -235280):

```
- if (!hasRole(PROVER_ROLE, address(0)) && !hasRole(PROVER_ROLE, _msgSender())) {
+ if (!(hasRole(PROVER_ROLE, address(0)) || hasRole(PROVER_ROLE, _msgSender()))) {
```

Axiom: Addressed in PR 180.

Spearbit: Verified that PR 180 implements the recommendations.

5.1.3 Unnecessary function arguments in `_sendQuery()`

Severity: Gas Optimization

Context: AxiomV2Query.sol#L652-L653

Description: The `_sendQuery()` function contains `address caller` and `uint256 depositAmount` as arguments. However, on all if its callsites, namely AxiomV2Query.sol#L265-L272 and AxiomV2Query.sol#L295-L302, `msg.sender` is passed as `caller` and `msg.value` as `depositAmount`.

It is therefore worth considering dropping these arguments and re-factor the function in such a way that `msg.sender` and `msg.value` are directly used instead.

Recommendation: Consider dropping the `address caller` and `uint256 depositAmount` function arguments and re-factoring `_sendQuery()` so that `msg.sender` and `msg.value` are directly used:

```
  function _sendQuery(
      uint256 queryId,
      uint64 maxFeePerGas,
      uint32 callbackGasLimit,
+     uint256 overrideAxiomQueryFee
-     uint256 overrideAxiomQueryFee,
-     address caller,
-     uint256 depositAmount
  ) internal {
      if (queries[queryId].state != AXIOM_QUERY_STATE_INACTIVE) {
          revert QueryIsNotInactive();
      }

      if (maxFeePerGas < minMaxFeePerGas) {
          revert MaxFeePerGasIsTooLow();
      }

      uint256 _axiomQueryFee = axiomQueryFee;
      if (overrideAxiomQueryFee > _axiomQueryFee) {
          _axiomQueryFee = overrideAxiomQueryFee;
      }

      uint256 maxQueryPri = _getMaxQueryPri(maxFeePerGas, callbackGasLimit, _axiomQueryFee);
-     if (depositAmount != maxQueryPri) {
-         if (depositAmount > 0) {
-             _recordDeposit(caller, depositAmount);
+     if (msg.value != maxQueryPri) {
+         if (msg.value > 0) {
+             _recordDeposit(msg.sender, msg.value);
          }
```

```
          if (maxQueryPri > balances[caller]) {
              revert EscrowAmountExceedsBalance();
          }
          unchecked {
-             // in this branch, we know that maxQueryPri <= balances[caller]
-             balances[caller] -= maxQueryPri;
+             // in this branch, we know that maxQueryPri <= balances[msg.sender]
+             balances[msg.sender] -= maxQueryPri;
          }
      }

      queries[queryId] = AxiomQueryMetadata({
          state: AXIOM_QUERY_STATE_ACTIVE,
          deadlineBlockNumber: uint32(block.number) + queryDeadlineInterval,
          callbackGasLimit: callbackGasLimit,
          payee: address(0),
          payment: maxQueryPri
      });
      emit QueryFeeInfoRecorded(
-         queryId, caller, uint32(block.number) + queryDeadlineInterval, maxFeePerGas,
↪  callbackGasLimit, maxQueryPri
+         queryId, msg.sender, uint32(block.number) + queryDeadlineInterval, maxFeePerGas,
↪  callbackGasLimit, maxQueryPri
      );
  }
```

Also, update all callsites accordingly. This change reduces the bytecode size (`-0.054` kB). Furthermore, using `msg.sender` and `msg.value` results in minor gas savings.

Axiom: Addressed in PR 180.

Spearbit: Verified that PR 180 implements the recommendation.

5.1.4 Replace zeroing-out variables for the `delete` keyword

Severity: Gas Optimization

Context: AxiomV2Query.sol#L482-L488

Description: Deleting a variable es equivalent to setting it to the default value, which is usually the underlying `bytes32 0x0` value. Although zeroing-out manually is possible, it is recommended to use Solidity's built-in `delete` keyword for instead, especially when treating with complex data types (such as structs) as it is slightly more efficient.

Recommendation: Consider employing the `delete` keyword to enhance readability and efficiency.

```
- queries[queryId] = AxiomQueryMetadata({
-     state: AXIOM_QUERY_STATE_INACTIVE,
-     deadlineBlockNumber: 0,
-     callbackGasLimit: 0,
-     payee: address(0),
-     payment: 0
- });
+ delete query[queryId];
```

This change reduces deployment size from 24.548kB to 24.371kB, resulting in gas savings.

Axiom: Addressed in PR 180.

Spearbit: Verified that PR 180 implements the recommendation.

5.1.5 Set `axiomCoreAddress` as `immutable`

Severity: Gas Optimization

Context: AxiomV2HeaderVerifier.sol#L14

Description: The `axiomCoreAddress` is currently a public state variable that is only set on construction. Furthermore, when set, it emits an `UpdateAxiomCoreAddress(address)` event. Given this scenario, it would be better to declare it as `immutable` and redeploy the contract if needed. The benefits of this change would be:

- 2100 gas savings in `fulfillQuery()` because a cold key `SLOAD` is avoided (349280 → 347180).
- The `UpdateAxiomCoreAddress()` event becomes obsolete, and its removal slightly reduces the contract size.

Recommendation: Consider declaring `axiomCoreAddress` as `address public immutable axiomCoreAddress`.

Axiom: Addressed in PR 173.

Spearbit: Verified that PR 173 implements the recommendations.

5.1.6 Caching storage variables to save `SLOAD`s

Severity: Gas Optimization

Context: AxiomV2Query.sol#L745-L747, AxiomV2Query.sol#L683-L693, AxiomV2Query.sol#L674-L680, AxiomV2Query.sol#L310-L352, AxiomV2Query.sol#L431-L439, AxiomV2Query.sol#L380-L407, AxiomV2Query.sol#L470-L480, AxiomV2Query.sol#L511-L523

Description: When a storage variable is used several times within the same scope, it is more efficient caching it for the subsequent reads rather than directly reading from storage, as this implies unnecessary executions of the `SLOAD` instruction.

Recommendation: Consider caching the storage variables that are used several times within the same scope. See below for several instances within the codebase where this optimization may be applied:

- AxiomV2Query.sol#L745-L747 (overall gas change: -768233):

```
+ address _axiomHeaderVerifierAddress = axiomHeaderVerifierAddress;
  // verify against on-chain data
- IAxiomV2HeaderVerifier(axiomHeaderVerifierAddress).verifyQueryHeaders(blockhashMmrKeccak, mmrWitness);
+ IAxiomV2HeaderVerifier(_axiomHeaderVerifierAddress).verifyQueryHeaders(blockhashMmrKeccak,
↪  mmrWitness);

- if (proofData.sourceChainId != IAxiomV2HeaderVerifier(axiomHeaderVerifierAddress).getSourceChainId())
↪  {
+ if (proofData.sourceChainId !=
↪  IAxiomV2HeaderVerifier(_axiomHeaderVerifierAddress).getSourceChainId()) {
      revert SourceChainIdDoesNotMatch();
  }
```

- AxiomV2Query.sol#L683-L693:

```
+ uint32 _queryDeadlineInterval = queryDeadlineInterval;
  queries[queryId] = AxiomQueryMetadata({
      state: AXIOM_QUERY_STATE_ACTIVE,
-     deadlineBlockNumber: uint32(block.number) + queryDeadlineInterval,
+     deadlineBlockNumber: uint32(block.number) + _queryDeadlineInterval,
      callbackGasLimit: callbackGasLimit,
      payee: address(0),
      payment: maxQueryPri
  });
  emit QueryFeeInfoRecorded(
-     queryId, caller, uint32(block.number) + queryDeadlineInterval, maxFeePerGas, callbackGasLimit,
↪  maxQueryPri
+     queryId, caller, uint32(block.number) + _queryDeadlineInterval, maxFeePerGas, callbackGasLimit,
↪  maxQueryPri
  );
```

- AxiomV2Query.sol#L674-L680:

```
+ uint256 _callerBalance = balances[caller];
- if (maxQueryPri > balances[caller]) {
+ if (maxQueryPri > _callerBalance ) {
      revert EscrowAmountExceedsBalance();
  }
  unchecked {
-     // in this branch, we know that maxQueryPri <= balances[caller]
-     balances[caller] -= maxQueryPri;
+     // in this branch, we know that maxQueryPri <= balances[caller] (i.e. _callerBalance)
+     balances[caller] = _callerBalance - maxQueryPri;
  }
```

- AxiomV2Query.sol#L310-L352 (overall gas change: -621213):

```
  function increaseQueryGas(
      uint256 queryId,
      uint64 newMaxFeePerGas,
      uint32 newCallbackGasLimit,
      uint256 overrideAxiomQueryFee
  ) external payable onlyNotFrozen {
+     AxiomQueryMetadata storage queryMetadata = queries[queryId];
-     if (queries[queryId].state != AXIOM_QUERY_STATE_ACTIVE) {
+     if (queryMetadata.state != AXIOM_QUERY_STATE_ACTIVE) {
          revert CanOnlyIncreaseGasOnActiveQuery();
      }
      if (newMaxFeePerGas < minMaxFeePerGas) {
          revert MaxFeePerGasIsTooLow();
      }

-     uint256 oldAmount = queries[queryId].payment;
+     uint256 oldAmount = queryMetadata.payment;

      uint256 _axiomQueryFee = axiomQueryFee;
      if (overrideAxiomQueryFee > _axiomQueryFee) {
          _axiomQueryFee = overrideAxiomQueryFee;
      }
      uint256 newMaxQueryPri = _getMaxQueryPri(newMaxFeePerGas, newCallbackGasLimit, _axiomQueryFee);
      if (newMaxQueryPri <= oldAmount) {
          revert NewMaxQueryPriMustBeLargerThanPrevious();
      }
      uint256 increaseAmount;
      unchecked {
          // in this branch, we know that newMaxQueryPri > oldAmount
```

```
            increaseAmount = newMaxQueryPri - oldAmount;
        }
        if (msg.value < increaseAmount) {
            revert InsufficientFunds();
        }
-       queries[queryId].payment = newMaxQueryPri;
+       queryMetadata.payment = newMaxQueryPri;
        emit QueryGasIncreased(queryId, newMaxFeePerGas, newCallbackGasLimit, overrideAxiomQueryFee);

        if (msg.value > increaseAmount) {
            uint256 deposit;
            unchecked {
                // in this branch, we know that msg.value > increaseAmount
                deposit = msg.value - increaseAmount;
            }
            _recordDeposit(msg.sender, deposit);
        }
    }
```

- AxiomV2Query.sol#L431-L439 (overall gas change: -789150):

```
+ AxiomQueryMetadata storage queryMetadata = queries[queryId];
- if (queries[queryId].state != AXIOM_QUERY_STATE_INACTIVE) {
+ if (qqueryMetadata.state != AXIOM_QUERY_STATE_INACTIVE) {
        revert CannotFulfillFromOffchainIfNotInactive();
    }

    if (proofData.payee != msg.sender) {
        revert OnlyPayeeCanFulfillOffchainQuery();
    }

- queries[queryId].state = AXIOM_QUERY_STATE_FULFILLED;
+ queryMetadata.state = AXIOM_QUERY_STATE_FULFILLED;
```

- AxiomV2Query.sol#L380-L407:

```
+ AxiomQueryMetadata storage queryMetadata = queries[queryId];
- if (queries[queryId].state != AXIOM_QUERY_STATE_ACTIVE) {
+ if (queryMetadata.state != AXIOM_QUERY_STATE_ACTIVE) {
      revert CannotFulfillIfNotActive();
  }

- queries[queryId].payee = proofData.payee;
- queries[queryId].state = AXIOM_QUERY_STATE_FULFILLED;
+ queryMetadata.payee = proofData.payee;
+ queryMetadata.state = AXIOM_QUERY_STATE_FULFILLED;

  bool success;
  /// @dev re-entrancy protection:
  ///   we check and transition the query state before calling a client contract
  if (callback.target != address(0)) {
      bytes memory data = abi.encodeWithSelector(
          IAxiomV2Client.axiomV2Callback.selector,
          proofData.sourceChainId,
          queryWitness.caller,
          proofData.querySchema,
          queryId,
          computeResults,
          callback.extraData
      );

      /// @dev This checks that the callback is provided at least `callbackGasLimit` gas.
      ///      Factor of 64 / 63 accounts for the EIP-150 gas forwarding rule.
      ///      Additional 300 gas accounts for computation of the conditional branch.
-     if (gasleft() - 300 <= queries[queryId].callbackGasLimit * 64 / 63) {
+     if (gasleft() - 300 <= queryMetadata.callbackGasLimit * 64 / 63) {
          revert InsufficientGasForCallback();
      }
-     (success,) = callback.target.excessivelySafeCall(queries[queryId].callbackGasLimit, 0, 0, data);
+     (success,) = callback.target.excessivelySafeCall(queryMetadata.callbackGasLimit, 0, 0, data);
```

- AxiomV2Query.sol#L470-L480 (overall gas change: -358781):

```
+ AxiomQueryMetadata storage queryMetadata = queries[queryId];
- if (queries[queryId].state != AXIOM_QUERY_STATE_ACTIVE) {
+ if (queryMetadata.state != AXIOM_QUERY_STATE_ACTIVE) {
      revert CannotRefundIfNotActive();
  }
- if (block.number <= queries[queryId].deadlineBlockNumber) {
+ if (block.number <= queryMetadata.deadlineBlockNumber) {
      revert CannotRefundBeforeDeadline();
  }

  unchecked {
      // balances cannot overflow
-     balances[refundee] += queries[queryId].payment;
+     balances[refundee] += queryMetadata.payment;
  }
```

- AxiomV2Query.sol#L511-L523 (overall gas change: -812972):

```
+ AxiomQueryMetadata storage queryMetadata = queries[queryId];
- if (queries[queryId].state != AXIOM_QUERY_STATE_FULFILLED) {
+ if (queryMetadata.state != AXIOM_QUERY_STATE_FULFILLED) {
      revert QueryIsNotFulfilled();
  }
- uint256 payment = queries[queryId].payment;
+ uint256 payment = queryMetadata.payment;
  if (amountUsed > payment) {
      revert UnescrowAmountExceedsEscrowedAmount();
  }
- address payee = queries[queryId].payee;
+ address payee = queryMetadata.payee;
  if (msg.sender != payee) {
      revert OnlyPayeeCanUnescrow();
  }

- queries[queryId].state = AXIOM_QUERY_STATE_PAID;
+ queryMetadata.state = AXIOM_QUERY_STATE_PAID;
```

Axiom: Addressed in PR 180.

Spearbit: Verified that PR 180 implements the recommendation.

5.1.7 Move variable declaration outside of `for` loop

Severity: Gas Optimization

Context: AxiomV2Core.sol#L275

Description: The `commitment` variable is declared and constantly re-assigned within the `for` loop. Furthermore, it is re-declared at AxiomV2Core.sol#L289, causing the solidity compiler to throw a compiler warning due to variable shadowing and costing more gas than necessary.

Recommendation: Given that `commitment` is re-assigned on every loop iteration plus at AxiomV2Core.sol#L289, consider declaring it outside of the loop. Moving the declaration to AxiomV2Core.sol#L272 and then just performing the assignments leads to gas savings (overall gas change: -100583):

```
+ bytes32 commitment;
  // check all complete leaves
  for (uint256 i; i < roots.length - 1;) {
-     bytes32 commitment = keccak256(abi.encodePacked(prevHashes[i], roots[i], BLOCK_BATCH_SIZE));
+     commitment = keccak256(abi.encodePacked(prevHashes[i], roots[i], BLOCK_BATCH_SIZE));
      if (historicalRoots[startBlockNumber] != commitment) {
          revert AxiomBlockVerificationFailed();
      }
      startBlockNumber += BLOCK_BATCH_SIZE;
      unchecked {
          ++i;
      }
  }

  // append all complete leaves
  uint256 peaksChanged = pmmr.appendCompleteLeaves(BLOCK_BATCH_SIZE, roots[:roots.length - 1]);

  // check the last, possibly incomplete leaf
- bytes32 commitment =
+ commitment =
      keccak256(abi.encodePacked(prevHashes[roots.length - 1], roots[roots.length - 1], lastNumFinal));
```

Axiom: Addressed in PR 183.

Spearbit: Verified that PR 183 implements the recommendation.

5.1.8 Simplify `excessivelySafeCall()` for efficiency

Severity: Gas Optimization

Context: ExcessivelySafeCall.sol#L23-L55

Description: All callsites of `excessivelySafeCall()` pass `0` as `_value` parameter, and also do not make use of the call's return. Therefore, `excessivelySafeCall()` admits some simplifications that result in major gas savings.

Recommendation:

- Consider applying the following changes to `excessivelySafeCall()`:

```
- function excessivelySafeCall(address _target, uint256 _gas, uint256 _value, uint16 _maxCopy, bytes
↪  memory _calldata)
+ function excessivelySafeCall(address _target, uint256 _gas, bytes memory _calldata)
      internal
-     returns (bool, bytes memory)
+     returns (bool)
  {
      // set up for assembly call
-     uint256 _toCopy;
      bool _success;
-     bytes memory _returnData = new bytes(_maxCopy);
      // dispatch message to recipient
      // by assembly calling "handle" function
      // we call via assembly to avoid memcopying a very large returndata
      // returned by a malicious contract
      assembly {
          _success :=
              call(
                  _gas, // gas
                  _target, // recipient
-                 _value, // ether value
+                 0, // ether value
                  add(_calldata, 0x20), // inloc
                  mload(_calldata), // inlen
                  0, // outloc
                  0 // outlen
              )
-         // limit our copy to 256 bytes
-         _toCopy := returndatasize()
-         if gt(_toCopy, _maxCopy) { _toCopy := _maxCopy }
-         // Store the length of the copied bytes
-         mstore(_returnData, _toCopy)
-         // copy the bytes from returndata[0:_toCopy]
-         returndatacopy(add(_returnData, 0x20), 0, _toCopy)
      }
-     return (_success, _returnData);
+     return _success;
  }
```

And update the callsites accordingly. These modifications result in an overall gas change of -3012942.

Note: it would be beneficial to rename the modified `excessivelySafeCall()` to something else for the sake of clarity. Alternatively, since `excessivelySafeCall()` is only used within `AxiomV2Query`, consider removing the library altogether, and create an internal helper function with the above functionality.

- Use this updated version for the external call at AxiomV2Query.sol#L770:

```
(bool success,) = verifierAddress.call(proof);
```

Axiom: Implemented the first recommendation in PR 182.

Spearbit: Verified that PR 182 implements the recommendation.

5.2 Informational

5.2.1 Missing Natspec comments

Severity: Informational

Context: AxiomV2HeaderVerifier.sol#L261

Description: Most of the functions include proper Natspec comments. However, `supportsInterface()` lacks Natspec comments.

Recommendation: Add the missing Natspec comments to the aforementioned function.

Axiom: Addressed in PR 176.

Spearbit: Verified that PR 176 implements the recommendation.

5.2.2 Change wording for `overrideAxiomQueryFee` documentation

Severity: Informational

Context: AxiomV2Query.sol#L644, IAxiomV2Query.sol#L116, IAxiomV2Query.sol#L350, IAxiomV2Query.sol#L449, README.md#L138, README.md#L299

Description: Throughout the code/README documentation, the new `overrideAxiomQueryFee` argument is described in the following ways:

```
/// @param  overrideAxiomQueryFee If non-zero, a different query fee to send to Axiom.
- `overrideAxiomQueryFee` (`uint256`): If non-zero, a higher `axiomQueryFee` to use for this query.
- `overrideAxiomQueryFee` (`uint256`) -- if non-zero, higher `axiomQueryFee` than the default to be
↪  used for this query
```

This wording is slightly incorrect, and the actual behavior would be better described by "If larger than `axiomQueryFee`, the value to be used for the query fee".

Recommendation: Change the wording for the `overrideAxiomQueryFee` documentation.

Axiom: Addressed in PR 176.

Spearbit: Verified that PR 176 implements the recommendation.

5.2.3 `dataQuery` argument can be commented

Severity: Informational

Context: AxiomV2Query.sol#L231

Description: `dataQuery` argument in `sendQuery()` is taken just for onchain DA. Compiler throws a warning for this.

Recommendation: Comment the argument:

```
bytes calldata /*dataQuery*/
```

Axiom: Addressed in PR 175.

Spearbit: Verified that PR 175 implements the recommendation.

5.2.4 Redundant import

Severity: Informational

Context: AxiomV2HeaderVerifier.sol#L6

Description: `MAX_MMR_PEAKS` is imported but unused.

Recommendation: Remove `MAX_MMR_PEAKS`.

```
- import { MerkleMountainRange, MAX_MMR_PEAKS } from "../libraries/MerkleMountainRange.sol";
+ import { MerkleMountainRange } from "../libraries/MerkleMountainRange.sol";
```

Axiom: Addressed in PR 177.

Spearbit: Verified that PR 177 implements the recommendation.

5.2.5 Typos

Severity: Informational

Context: README.md#L42, README.md#L69

Description: Below is a list with some typos and their corresponding fixes.

Recommendation: See each individual case:

- README.md#L42:

```
- - `core/AxiomV2CoreMainnetVerifier.*.soll`: On-chain verifier for ZK circuits verifying chains of
↪  block headers on mainnet for `AxiomV2Core`. These are versioned.
+ - `core/AxiomV2CoreMainnetVerifier.*.sol`: On-chain verifier for ZK circuits verifying chains of
↪  block headers on mainnet for `AxiomV2Core`. These are versioned.
```

- README.md#L69:

```
- - The `blockhashPmmr` stores a padded Merkle mountain range which commits to a continguous chain of
↪  block hashes starting from genesis using:
+ - The `blockhashPmmr` stores a padded Merkle mountain range which commits to a contiguous chain of
↪  block hashes starting from genesis using:
```

Axiom: Addressed in PR 176.

Spearbit: Verified that PR 176 implements the recommendation.

5.2.6 Replace manual hours to seconds conversion by `hours` keyword

Severity: Informational

Context: AxiomTimelock.sol#L26

Description: Solidity version `0.8.19` (used in this project) implements time unit keywords which ease time conversions into seconds. Therefore, manual time conversions are unnecessary and bloat the codebase.

Recommendation: Replace time conversions by the appropriate keywords, such as in the case shown below:

```
- if (minDelay < 3 * 60 * 60) {
+ if (minDelay < 3 hours) {
```

Axiom: Addressed in PR 178.

Spearbit: Verified that PR 178 implements the recommendation.

5.2.7 Document theoretical overflow

Severity: Informational

Context: PaddedMerkleMountainRange.sol#L94-L96

Description: The `unchecked` logic within `updatePaddedLeaf()` can technically overflow with large enough `paddingSize` and `leafSize` arguments. For Axiom, this scenario would occur once `block.number` hits 2^{32} = 4294967296 (~4.29 billion), which would take around 1.6 millennia on mainnet. Thus, it will not overflow in practice.

Nonetheless, since `PaddedMerkleMountainRange` is a library that might be used in other ways by third parties, or even Axiom in the future, perhaps it would be good to add a `@dev Warning:` comment to `updatePaddedLeaf()` stating this theoretical overflow.

Recommendation: Consider adding a `@dev Warning:` comment to `updatePaddedLeaf()` mentioning the previously discussed situation.

Axiom: Addressed in PR 176.

Spearbit: Verified that PR 176 implements the recommendation.

5.2.8 Variable renaming

Severity: Informational

Context: AxiomV2Query.sol#L345

Description: The `deposit` variable shadows an existing declaration, and should be renamed.

Recommendation: Consider applying the following renaming:

- `deposit` → `depositAmount`

Axiom: Addressed in PR 179.

Spearbit: Verified that PR 179 implements the recommendations.

5.2.9 Revert behavior in `_recordDeposit()` when `depositAmount > MAX_DEPOSIT_SIZE` has been lost

Severity: Informational

Context: AxiomV2Query.sol#L668-L681

Description: The code prior to the PR of this security review reverted in `_recordDeposit()` when `depositAmount > MAX_DEPOSIT_SIZE`. In the current implementation, it is possible to get around the check when `depositAmount == maxQueryPri`. See the diff of the `_sendQuery` function below:

```
  function _sendQuery(
      uint256 queryId,
      uint64 maxFeePerGas,
      uint32 callbackGasLimit,
+     uint256 overrideAxiomQueryFee,
      address caller,
      uint256 depositAmount
  ) internal {
      if (queries[queryId].state != AXIOM_QUERY_STATE_INACTIVE) {
          revert QueryIsNotInactive();
      }

      if (maxFeePerGas < minMaxFeePerGas) {
          revert MaxFeePerGasIsTooLow();
      }

+     uint256 _axiomQueryFee = axiomQueryFee;
```

```
+     if (overrideAxiomQueryFee > _axiomQueryFee) {
+         _axiomQueryFee = overrideAxiomQueryFee;
+     }

-     uint256 maxQueryPri = _getMaxQueryPri(maxFeePerGas, callbackGasLimit);
+     uint256 maxQueryPri = _getMaxQueryPri(maxFeePerGas, callbackGasLimit, _axiomQueryFee);
+     if (depositAmount != maxQueryPri) {
          if (depositAmount > 0) {
              _recordDeposit(caller, depositAmount);
          }

          if (maxQueryPri > balances[caller]) {
              revert EscrowAmountExceedsBalance();
          }
+         unchecked {
+             // in this branch, we know that maxQueryPri <= balances[caller]
              balances[caller] -= maxQueryPri;
+         }
+     }

      queries[queryId] = AxiomQueryMetadata({
          state: AXIOM_QUERY_STATE_ACTIVE,
          deadlineBlockNumber: uint32(block.number) + queryDeadlineInterval,
          callbackGasLimit: callbackGasLimit,
          payee: address(0),
          payment: maxQueryPri
      });
      emit QueryFeeInfoRecorded(
          queryId, caller, uint32(block.number) + queryDeadlineInterval, maxFeePerGas,
↪   callbackGasLimit, maxQueryPri
      );
  }
```

Clearly, when `depositAmount == maxQueryPri`, the `depositAmount > MAX_DEPOSIT_SIZE` within `_recordDeposit()` is never reached as `_recordDeposit()` itself is never called.

Recommendation: Although it is a minor detail, if the exact old behavior is desired, then the `depositAmount > MAX_DEPOSIT_SIZE` should be added outside of the `if (depositAmount != maxQueryPri) { ... }` (either after or before it).

Axiom: Addressed in PR 184.

Spearbit: Verified.