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

November 26, 2023

Contents

1 About Spearbit **3**

2 Introduction **3**

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

4 Executive Summary **4**

5 Findings **5**
5.1 Medium Risk . . . 5
5.1.1 Lack of upper bounds in `queryDeadlineInterval` may lead to locked funds and/or DoS . . . 5
5.2 Low Risk . . . 6
5.2.1 [Axiom - 01] Paying to caller of `fulfillOffchainQuery` is vulnerable to mempool sniping . . . 6
5.2.2 [Axiom - 02] `queryId` should commit to `targetChainId` . . . 6
5.2.3 [Axiom - 03] Padded Merkle mountain range cannot be updated after `updateOld` . . . 6
5.2.4 Missing non-zero checks allow event emission spamming . . . 7
5.2.5 Lack of zero address check in `withdraw` and `deposit` can burn ether . . . 7
5.2.6 No enforcement of callback gas limit for on-chain queries . . . 7
5.2.7 Allow `refundQuery` to a custom `to` address . . . 8
5.2.8 Add code length check for `callback.target` . . . 9
5.2.9 `increaseQueryGas` can be called even if the contracts are in a `frozen` state . . . 9
5.2.10 Missing sanity zero-address checks may lead to undesired behavior . . . 10
5.2.11 Missing input validation on fee parameters . . . 10
5.2.12 Use `ExcessivelySafeCall` for external calls where return value isn't checked . . . 11
5.2.13 Excess ETH sent for `increaseQueryGas()` isn't accounted for . . . 11
5.3 Gas Optimization . . . 11
5.3.1 Boundary equality case can be moved to do `proofMmrPeaks` extension . . . 11
5.3.2 Minimise external calls . . . 12
5.3.3 Intentional `appendCompleteLeaves()` no-op can be removed . . . 12
5.3.4 `bool` costs more gas than `uint` . . . 13
5.3.5 Arithmetic operations can be optimized for gas savings . . . 13
5.3.6 Usage of custom errors and revert strings inconsistency . . . 14
5.3.7 Multiple `peakLength` typecasting to `uint32` . . . 15
5.3.8 `start` initialization can be optimized . . . 15
5.3.9 Iterators can be optimized . . . 15
5.3.10 `CHAIN_ID` can be set to `internal` due to the existence of a custom getter . . . 16
5.3.11 User input validation should have preference over other actions . . . 16
5.4 Informational . . . 22
5.4.1 Naive provers may spend more funds to compute than the amount awarded to them . . . 22
5.4.2 Remove redundant code to compute `queryId` . . . 22
5.4.3 Passing ETH to callback target is unsupported . . . 23
5.4.4 Callback call success is not monitored . . . 23
5.4.5 `frozen` state in `withdraw` introduces a centralization risk . . . 23
5.4.6 Missing indexed event parameters . . . 24
5.4.7 Use named imports to improve clarity and efficiency . . . 24
5.4.8 Usage of `axiomProverAddress` can be simplified . . . 25
5.4.9 Open up access to `addAllowedProver` and `removeAllowedProver` . . . 25
5.4.10 PMMR `size` initialization can be more accurate . . . 25
5.4.11 Idempotent checks may be performed to avoid emitting events with no changes . . . 26
5.4.12 Favor Yul `switch-case` statements over multiple `if` statements . . . 27
5.4.13 Goerli testnet will stop working in 2024 . . . 28
5.4.14 Variable naming improvements . . . 28

5.4.15 Incorrect `peaksLength` value set for MMRs 28
5.4.16 `virtual` is redundant in interfaces and it can be removed 29
5.4.17 Unused logic and statements 29
5.4.18 Absent/incomplete natspec affects readability and maintenance 29

6 Additional Comments **33**

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 14 days in total, Axiom engaged with Spearbit to review the axiom-v2-contracts-working protocol. In this period of time a total of **43** issues were found.

Summary

Project Name	Axiom
Repository	axiom-v2-contracts-working
Commit	41fab5...22d8
Type of Project	Data availability, ZK
Audit Timeline	Oct 16 to Oct 27
Two week fix period	Oct 27 - Nov 10

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	13	12	1
Gas Optimizations	11	5	6
Informational	18	15	3
Total	**43**	**33**	**10**

5 Findings

5.1 Medium Risk

5.1.1 Lack of upper bounds in `queryDeadlineInterval` may lead to locked funds and/or DoS

Severity: Medium Risk

Context: AxiomV2Query.sol#L98, AxiomV2Query.sol#L150

Description: The lack of upper bounds in `queryDeadlineInterval` may lead to two different harmful scenarios.

1. Ether included within a query is expected to be returned to the query creator in case the query is not fulfilled after a certain amount of blocks, through a call to `refundQuery`. This mechanism is created to protect users from scenarios in which provers can't/won't fulfill their queries. However, the amount of time someone might need to wait until getting their funds back is in the interval [`0`,`type(uint32).max - block.number`), which is stored in the `queries` mapping.

   Taking into account the following facts:

   - The current `block.number` (at the time of writing) in mainnet is `18_383_880`.
   - In mainnet, every 12s approximately, a block is produced.
   - `type(uint32).max` is the maximum value to be stored, i.e. `4_294_967_295`.
   - The time to reach `type(uint32).max` goes beyond human scales, i.e. approximately 1626 years from the current `block.number`.

   Even in the case of setting the deadline to 1 or 2 years, the user would have frozen funds and no other option but waiting for the deadline or query fulfillment. This shouldn't be so harmful in case the query can be fufilled quickly, or in case of a zero-day bug appearance and the query can't be filled, or in the case all the provers are down, the user may have to wait that amount of time.

2. In the case an initial huge (valid) value for `queryDeadlineInterval` is set, which makes `uint32(block.number) + queryDeadlineInterval > type(uint32).max` (i.e. overflow, as `deadlineBlockNumber`'s type is `uint32`), all calls to `_sendQuery` methods will revert. This may pass unalerted initially as it is valid for `queryDeadlineInterval`. The reason for this are that the solidity compiler version lies above `0.8.0` (i.e. automatic overflow/underflow checks) and the following lines within `_sendQuery` functions:

```
queries[queryId] = AxiomQueryMetadata({
    state: AxiomQueryState.Active,
    deadlineBlockNumber: uint32(block.number) + queryDeadlineInterval, //@audit can overflow and revert
    payee: address(0),
    payment: maxQueryPri
});
```

This scenario results in a bad user experience, as they would be wasting gas for transactions until the Axiom multisig with `TIMELOCK_ROLE` changes `queryDeadlineInterval` to a smaller value that doesn't make `uint32(block.number) + queryDeadlineInterval > type(uint32).max`.

Recommendation: Consider adding an upper bound for the different chains. For example, from 1 to 4 weeks in blocks:

- 1 week upperbound for mainnet considering ~12s: ~50,400 blocks.
- 1 week upperbound for arbitrum considering ~0.26s: ~2,326,923 blocks.

Axiom: Added validation on mainnet and testnets in PR 88.

Spearbit: Fixed. The `queryDeadlineInterval` now has an upper bound of `50_400` (1 week of 12 second blocks), and a helper function has been created to support different upper bounds as more chains are supported.

5.2 Low Risk

5.2.1 [Axiom - 01] Paying to caller of `fulfillOffchainQuery` is vulnerable to mempool sniping

Severity: Low Risk

Context: AxiomV2Query.sol

Description: We currently tag proofs with `payee` for on-chain queries and enforce that on-chain query payments are made to `payee`. This prevents mempool copying of proofs.

For off-chain queries which are fulfilled in `fulfillOffchainQuery`, we pass `caller` to the callback, which allows the callback recipient to make payment to that address. However, because `caller` is not tagged in the proof, this is vulnerable to mempool sniping of proofs.

Axiom: We plan to fix this by enforcing that `caller == payee`, which enables a downstream contract to make payment without causing MEV issues. The fix can be found in PR 116.

Spearbit: Fixed. Will revert if `caller != payee`.

5.2.2 [Axiom - 02] `queryId` should commit to `targetChainId`

Severity: Low Risk

Context: AxiomV2Query.sol

Description: Right now, any query with the same computation and referencing the same data may have the same `queryId`.

In future cross-chain / rollup use cases, we envision deploying the protocol on different target chains, meaning it will be helpful for off-chain backends to have `queryId` be a unique key across different target chains.

We can achieve this by committing to `targetChainId` within `queryId`.

Axiom: Fixed in PR 117. There is a subsequent PR (PR 130) to keep a consistent convention that all `chainId` types are `uint64`.

Spearbit: Fixed.

5.2.3 [Axiom - 03] Padded Merkle mountain range cannot be updated after `updateOld`

Severity: Low Risk

Context: AxiomV2Core.sol#L230

Description: The incorrect check in AxiomV2Core.sol#L230 of `appendHistoricalPMMR` may cause `blockhashPmmr` updates to be blocked.

If `blockhashPmmr.size` is not a multiple of `BLOCK_BATCH_SIZE`, the check above will prevent usage of `appendHistoricalPMMR`. This will not occur in ordinary usage, where `appendHistoricalPMMR` is used after `updateHistorical` to bootstrap the padded Merkle mountain range from genesis. However, it can cause an issue in the following scenario:

- The `AxiomV2Core` contract is in a state where `blockhashPmmr.size` is not a multiple of `BLOCK_BATCH_SIZE`.
- `updateRecent` is not called for more than 1024 blocks.
- To get AxiomV2Core back in sync, we call `updateRecent` and then `updateOld` several times.

At this point, it will not be possible to continue appending to `blockhashPmmr` due to the check above in `appendHistoricalPMMR`.

Axiom: We plan to fix this by changing the check:

```
- || startBlockNumber != pmmr.size // startBlockNumber must be the size of PMMR
+ || startBlockNumber != pmmr.size - (pmmr.size % BLOCK_BATCH_SIZE) // startBlockNumber must be the
↪  size of the complete leaves of PMMR
```

And updating the appending logic afterwards accordingly to account for the fact that `blockhashPmmr.size` may not start as a multiple of `BLOCK_BATCH_SIZE`.

Addressed in PR 111.

Spearbit: Fixed. 2 key changes were made. First, the start block number is checked to be the size of the start block number. Second, the `roots` have been extended by 1 such that the last entry contains the zero padded merkle root of a possibly incomplete batch (`size < 1024`) of blocks.

5.2.4 Missing non-zero checks allow event emission spamming

Severity: Low Risk

Context: AxiomV2Query.sol#L387, AxiomV2Query.sol#L421

Description: Functions `withdraw()` and `deposit()` are meant to facilitate deposits and withdrawals. However, they do not check if non-zero ether is deposited or withdrawn. Given their permissionless nature, this allows anyone to grief the system with zero ether deposits causing emission of events which may hinder indexing/monitoring systems.

Recommendation: Add non-zero checks for `deposit` and `withdraw` before event emission.

Axiom: This issue has been addressed in PR 93.

Spearbit: Fixed.

5.2.5 Lack of zero address check in `withdraw` and `deposit` can burn ether

Severity: Low Risk

Context: AxiomV2Query.sol#L387, AxiomV2Query.sol#L421

Description: `AxiomV2Query` contract has functions that send value to an address specified by the user. If the user accidentally uses the default value in the destination address, it will be `address(0)` and this will effectively burn the funds.

Recommendation: To prevent unintentional loss of funds, it is recommended to add a require statement to check that the destination address is not `address(0)`.

Axiom: This issue has been addressed in PR 85.

Spearbit: Fixed.

5.2.6 No enforcement of callback gas limit for on-chain queries

Severity: Low Risk

Context: AxiomV2Query.sol#L287-L297

Description: For on-chain queries, `sendQuery()` and `sendQueryWithIpfsData()` take in a `uint32 callbackGasLimit` input, which is the `gasLimit the payee wishes the callback to be called with.` However, there isn't an enforcement of the callback gas limit passed to the callback. The gas remaining for the callback could be less than the requested `callbackGasLimit` amount (eg. prover sends enough gas for proof verification but not the callback), and the query would still be fulfilled.

A prover has incentive to pass in as little gas as required since he can receive the max payment of `maxQueryPri` regardless of outcome.

Recommendation: Consider checking the remaining gas is at least the requested amount. Note that doing `callback.target.call{gas: callbackGasLimit}(...)` won't work because the return value isn't checked.

With consideration of the gas forwarding rule, the check could be as follows:

```
if (gasleft() <= queries[queryId].callbackGasLimit * 64 / 63) revert InsufficientGasForCallback()
```

where `AxiomQueryMetadata` is extended to store the callback gas limit as well.

Axiom: I'm a bit concerned that the gas usage to complete the computation of the condition in

```
if (gasleft() <= queries[queryId].callbackGasLimit * 64 / 63)
```

may cause the true amount of gas available to be forwarded to dip below `queries[queryId].callbackGasLimit` by a small amount. Having trouble quantifying this due to some Foundry issues, e.g. Foundry issue 6164.

One solution would be to allocate a small gas buffer (I think less than 500 should be OK since the `SLOAD` is warm) to complete the conditional computation, but would appreciate any suggestions.

This issue has been addressed in PR 92.

Spearbit: Agree with having a small buffer to be on the safe side.

A buffer of 300 gas has been given, and the requested `callbackGasLimit` is passed to the target.

```
if (gasleft() - 300 <= queries[queryId].callbackGasLimit * 64 / 63) {
  revert InsufficientGasForCallback();
}
(success,) = callback.target.excessivelySafeCall(queries[queryId].callbackGasLimit, 0, 0, data);
```

Axiom: Added a cap on the external call gas to avoid the opposite problem in PR 120.

Spearbit: Fixed.

5.2.7 Allow `refundQuery` to a custom `to` address

Severity: Low Risk

Context: AxiomV2Query.sol#L351-L385

Description: In the current implementation, the `refundee` field (which is a user input parameter) of the `queryWitness` struct is not enforced to be `msg.sender` at `sendQuery` functions. In fact, it can even be `address(0)`, and under that scenario remaining assets from the query can be locked by error.

Recommendation: It would be interesting to have 2 roles: a creator/owner of the query (the actual `refundee`) and a `to` address as input of `refundQuery` functions. Then, it would be necessary to only allow calls to `refundQuery` on behalf of `msg.sender`. An example implementation is shown below:

```
function refundQuery(AxiomV2QueryWitness calldata queryWitness, address to) external onlyNotFrozen {
    if (msg.sender != queryWitness.refundee) {
        revert CannotRefundIfNotRefundee();
    }
    if (to == address(0)){
        revert CannotBeZeroAddress();
    }

    // ...

    balances[to] += queries[queryId].payment;
    // ...
```

Axiom: Good points re validation of `refundee` / allowing a `to` address in the `refundQuery` call. We decided to address this in PR 94 by:

- Setting `refundee` to `msg.sender` if it is `address(0)`.
- Not adding a `to` address in the `refundQuery` function to reduce complexity.

The idea here is that if no `refundee` is given, the only way to permission a refund will be to require access to `msg.sender`, in which case offering `to` has minimal additional value.

Spearbit: Fixed.

5.2.8 Add code length check for `callback.target`

Severity: Low Risk

Context: AxiomV2Core.sol#L321, AxiomV2Core.sol#L326, AxiomV2Query.sol#L287, AxiomV2Query.sol#L337, AxiomV2Query.sol#L505

Description: The current implementation performs the `callback.target != address(0)` check to decide on whether performing the callback or not. Nonetheless, `callback.target` may not be `address(0)` but an address without code (for example, an EOA), which passes the check and gets the callback executed without purpose. If the address' code length was verified instead, neither `address(0)` nor addresses without code would be used for the callback. An example of this check is shown below:

```
+ uint256 size;
+ { // Wrapped this to avoid stack too deep
+     address callbackTarget = callback.target;
+     assembly {
+         size := extcodesize(callbackTarget)
+     }
+ }
+ if (size > 0) {
- if (callback.target != address(0)) {
  // ...
```

In other cases such as AxiomV2Core.sol#L321, AxiomV2Core.sol#L326 and AxiomV2Query.sol#L505 checks aren't present at all.

Recommendation: Consider checking for a code length greater than 0 instead of `address(0)` to account for the scenarios in which the address specified for the callback has no code.

Axiom: There are two types of external calls here:

- *Verifier contracts*: These should be updated infrequently (only for major protocol upgrades or security vulnerability fixes). We validate updates with zero address checks to guard against inadvertent deployment issues, but it is ultimately up to users to decide whether to trust the contracts. In the interest of reducing contract size and complexity, we don't think it adds much to security to also validate code length.
- *User-specified callbacks*: We use `callback.target == address(0)` to indicate that the callback should not be called. In all other cases, it's on the user to ensure that `target` has the appropriate callback implemented. Although we can certainly add more validation, ultimately the user must check that their `target` is appropriate, so we don't think this validation benefits security too much.

Spearbit: Acknowledged.

5.2.9 `increaseQueryGas` can be called even if the contracts are in a `frozen` state

Severity: Low Risk

Context: AxiomV2Query.sol#L229-L244

Description: The `increaseQueryGas` function in the `AxiomV2Query` contract allows users to increase the gas allocation for their queries by paying an additional fee. However, this function can currently be called even when the contracts are in a `frozen` state. When the system is frozen (usually due to an unforeseen vulnerability), it is not reasonable to allow users to increase their query gas since this operation is expected to be ineffective.

Recommendation: To maintain consistency and protect users, the `increaseQueryGas` function should include the `onlyNotFrozen` modifier, ensuring that it cannot be called when the system is in a `frozen` state. This will prevent users from potentially wasting funds on increasing query gas during a period when it is most likely to be ineffective, aligning the system's functionality with user expectations and system integrity.

Axiom: Yes, agree with this, we should make this have the `onlyNotFrozen` modifier. For context, we intend the frozen state to only be triggered in the event of an unforeseen ZK circuit or smart contract vulnerability. This issue has been addressed in PR 84.

Spearbit: Fixed.

5.2.10 Missing sanity zero-address checks may lead to undesired behavior

Severity: Low Risk

Context: AxiomV2Query.sol#L114-L131, AxiomV2Core.sol#L257-L270, AxiomV2Client.sol#L15, AxiomTimelock.sol#L20

Description: Certain logic should implement zero-address checks to avoid undesired behavior.

- `verifierAddress`, `axiomHeaderVerifierAddress` and `axiomProverAddress` implement these checks at the `initializer`. Nonetheless, these checks are absent at their updater functions, which leaves an open door to setting by mistake default value `address(0)` by the `TIMELOCK_ROLE` Axiom multisig. This scenario would create a temporal DoS until the value is changed back to a valid one.
- For immutable variables such as axiomV2QueryAddress this is recommended to be included too.
- For the timelock controller, a zero `minDelay` effectively negates the purpose of having a delay, which is contrary to the expected outcome. With a value of 0 or a very low number, concerns arise about administrative actions being executed without providing users sufficient time to make an informed decision about whether to continue using the system. Examples of critical actions that could be affected include `upgradeTo()` and `updatingFees()`, among others.

Recommendation: Consider adding zero-address checks on the aforementioned variables within their updater functions, and revert with the already defined appropriate error messages.

Axiom: Added zero-address checks for updater functions in PR 83. Added zero-address check for `axiomV2QueryAddress` in PR 134. Minimum delay added in PR 119.

Spearbit: Fixed.

5.2.11 Missing input validation on fee parameters

Severity: Low Risk

Context: AxiomV2Query.sol#L156, AxiomV2Query.sol#L163

Description: `updateProofVerificationGas()` and `updateAxiomQueryFee()` update two "fee" parameters without enforcing any bound checks on them. Therefore, the updated values can be as extreme as `0` or `type(uint256).max`. This may affect user experience if malicious/compromised `TIMELOCK_ROLE` user executes a fee change that front-runs some query, resulting in the user spending much more ether to fulfill said query, or in extreme cases (if the user has not enough funds to backup the query) a DoS.

Recommendation: Setting reasonable upper and lower bounds on these fee parameters would improve trustlessness and it would be harder to create a DoS situation under the scenario of a /compromised `TIMELOCK_ROLE` user front-running a query with a fee change. In fact, the documentation specifies some initial values (400000 and 0.003 ether for `proofVerificationGas` and `axiomQueryFee` respectively) that may be used as a reference when establishing the upper and lower bounds.

Axiom: This issue has been addressed in PR 90.

One thing to note is the frontrunning attack would require the signers of the `TIMELOCK_ROLE` to be compromised and for a new address to be added to the `TIMELOCK_ROLE` by the compromised signers. This would be publicly visible for the interval of the timelock, which would trigger a security concern / likely freezing of the contract. As a result, I think the attack described is quite unlikely.

Spearbit: Fixed.

5.2.12 Use `ExcessivelySafeCall` for external calls where return value isn't checked

Severity: Low Risk

Context: AxiomV2Query.sol#L287, AxiomV2Query.sol#L337

Description: `fulfillQuery()` and `fulfillOffchainQuery()` makes external calls to user specified `callback` address through the low-level `.call`. If it returns a huge amount of data, the current call can run out of gas when copying it to memory.

Since data copying isn't required here, we can skip it through `ExcessivelySafeCall` library. You can specify the number of bytes from return data to be copied to memory, which in this case can be `0`.

Recommendation: Use `ExcessivelySafeCall` instead of `.call`.

Axiom: Fixed with PR 107.

Spearbit: Fixed.

5.2.13 Excess ETH sent for `increaseQueryGas()` isn't accounted for

Severity: Low Risk

Context: AxiomV2Query.sol#L239-L242

Description: Any excess ETH sent, where `msg.value > newMaxQueryPri - oldAmount`, isn't accounted for, resulting in ETH permanently stuck in the contract.

Recommendation: Either ensure strict equality (check `msg.value == newMaxQueryPri - oldAmount`) or refund the excess ETH to the caller.

Axiom: Fixed in PR 91 by adding excess ETH to the user's balance. To avoid any reentrancy concern from refunding, I've chosen to attribute any excess balance to the user in `AxiomV2Query`.

Spearbit: Fixed.

5.3 Gas Optimization

5.3.1 Boundary equality case can be moved to do `proofMmrPeaks` extension

Severity: Gas Optimization

Context: AxiomV2HeaderVerifier.sol#L193-L194

Description: In the case where `proofMmrSize - mmrWitness.snapshotPmmrSize == IAxiomV2State(axiomCoreAddress).blockhashPmmrSize() - proofMmrSize`, instead of extending `pmmrSnapshots` to `proofMmrPeaks` which requires decommitting the PMMR, it would probably be cheaper to do the other case of extending `proofMmrPeaks` to `blockhashPmmr`.

Recommendation:

```
|| proofMmrSize - mmrWitness.snapshotPmmrSize
-    <= IAxiomV2State(axiomCoreAddress).blockhashPmmrSize() - proofMmrSize
+   < IAxiomV2State(axiomCoreAddress).blockhashPmmrSize() - proofMmrSize
```

Axiom: Acknowlegded. We've opted to keep this as-is pending future gas optimization.

Spearbit: Acknowledged.

5.3.2 Minimise external calls

Severity: Gas Optimization

Context: AxiomV2HeaderVerifier.sol#L192-L194, AxiomV2HeaderVerifier.sol#L264-L268

Description: In the first reference, there is a possible repeated call to `blockhashPmmrSize()`. The result can be cached and invoked before the if and else-if cases since they utilise its value.

In the second reference, there are `mmrWitness.proofMmrPeaks.length - BLOCK_BATCH_DEPTH` external calls to fetch the PMMR peaks for comparison against the proofMMr peaks. It would be a lot cheaper if there was a method to batch these calls into a single one (eg. take in a start index and length).

Recommendation: Cache the call for `IAxiomV2State(axiomCoreAddress).blockhashPmmrSize()` for the first reference.

```
uint32 corePmmrSize = IAxiomV2State(axiomCoreAddress).blockhashPmmrSize();
if (
   mmrWitness.snapshotPmmrSize >= block.number - 256
   && (
      proofMmrSize > corePmmrSize
      || proofMmrSize - mmrWitness.snapshotPmmrSize
         <= corePmmrSize - proofMmrSize
      )
) {
...
} else if (proofMmrSize >= block.number - 256) {
   if (proofMmrSize > corePmmrSize) {
      revert NoMoreRecentBlockhashMMR();
   }
   ...
}
```

And for the second reference, introduce a getter method that enables fetching multiple peaks.

Axiom: This issue has been addressed in PR 101.

Spearbit: Fixed.

5.3.3 Intentional `appendCompleteLeaves()` no-op can be removed

Severity: Gas Optimization

Context: AxiomV2HeaderVerifier.sol#L130-L148

Description: In one specific case in `verifyQueryHeaders()`, it is possible that the `proofMmrSize` is exactly one batch smaller than `mmrWitness.snapshotPmmrSize`. In this scenario, the code will call `appendCompleteLeaves()` on the PMMR with an empty list, which is a no-op. It could save some gas if `appendCompleteLeaves()` is not invoked in the first place.

Recommendation: Consider filtering this edge case out by restricting the `if` statement wrapping the `appendCompleteLeaves()` call. This can be accomplished by keeping the `proofMmrSize` updated as follows:

```
if (proofMmrSize % BLOCK_BATCH_SIZE > 0) {
    // complete the first 10 peaks of `proofMmrPeaks` to a full Merkle root and update `proofPmmr`
    bytes32 completedLeaf =
        proofBatchMmr.getComplementMerkleRoot(BLOCK_BATCH_DEPTH, mmrWitness.mmrComplementOrPeaks);
    proofPmmr.updatePaddedLeaf(BLOCK_BATCH_SIZE, completedLeaf, BLOCK_BATCH_SIZE);
+   proofMmrSize += BLOCK_BATCH_SIZE - (proofMmrSize % BLOCK_BATCH_SIZE);
}

if (mmrWitness.snapshotPmmrSize - (mmrWitness.snapshotPmmrSize % BLOCK_BATCH_SIZE) > proofMmrSize) {
    // append additional complete leaves
    proofPmmr.appendCompleteLeaves(BLOCK_BATCH_SIZE, mmrWitness.mmrComplementOrPeaks[11:]);
}
```

Note that this code would no longer enforce that `mmrWitness.mmrComplementOrPeaks[11:]` is indeed empty. However, this is similar to other parts of the code which do not inspect any unused parts of `mmrWitness.mmrComplementOrPeaks`.

Axiom: Acknowledged. We think this fix can work, but it causes `proofMmrSize` to change within the execution of `verifyQueryHeaders`, which we think may be confusing. We've opted to keep this as-is pending future gas optimization.

Spearbit: Acknowledged.

5.3.4 `bool` costs more gas than `uint`

Severity: Gas Optimization

Context: AxiomAccess.sol#L11

Description: In Solidity, using `bool` for storage variables can be more expensive in terms of gas than using `uint256` or other full-word types. This is due to the Solidity compiler's defense mechanisms against contract upgrades and pointer aliasing, each write operation to a `bool` storage variable results in an additional `SLOAD` operation. This extra operation is required to first read the slot's contents, replace the bits taken up by the boolean, and then write back the result.

The overall gas change was: `Overall gas change: -2288250 (-0.000%)`

Recommendation: To optimize the gas efficiency, consider using `uint256` for storage variables instead of `bool` as noticed by Open Zeppelin.

You can declare a couple of constant to make this more intuitive, `FROZEN == 1` and `NOT_FROZEN == 2` or just use the actual variable as such and compare the value.

Axiom: We decided not to make this optimization for clarity, as freezing / unfreezing should only occur in the case of an unforeseen ZK circuit or smart contract vulnerability.

Spearbit: Acknowledged.

5.3.5 Arithmetic operations can be optimized for gas savings

Severity: Gas Optimization

Context: AxiomV2Query.sol#L93, AxiomV2HeaderVerifier.sol#L92, AxiomV2HeaderVerifier.sol#L229, AxiomV2HeaderVerifier.sol#L239, AxiomV2HeaderVerifier.sol#L254, AxiomV2HeaderVerifier.sol#L264, AxiomV2HeaderVerifier.sol#L275, AxiomV2HeaderVerifier.sol#L315, MerkleTree.sol#L16, MerkleTree.sol#L21, MerkleMountainRange.sol#L35, MerkleMountainRange.sol#L47, MerkleMountainRange.sol#L58, MerkleMountainRange.sol#L73, MerkleMountainRange.sol#L104, MerkleMountainRange.sol#L146, MerkleMountainRange.sol#L181, PaddedMerkleMountainRange.sol#L63, AxiomV2Core.sol#L120, AxiomV2Core.sol#L204, AxiomV2Core.sol#L207, AxiomV2Core.sol#L235, AxiomV2Core.sol#L308, AxiomV2Configuration.sol#L54-L55, MerkleTree.sol#L15-L16, MerkleTree.sol#L19, AxiomV2Query.sol#L544-L566, AxiomV2Prover.sol#L93-L97, MerkleMountainRange.sol#L108-L114

Description: There are some arithmetic operations that can save gas:

- `Prefix` operator costs less gas than `postfix` operator: AxiomV2Query.sol#L93, MerkleTree.sol#L16, MerkleTree.sol#L21 are cases where `i++` can be `++i` safely.
- Increments can be `unchecked` increments in `for` and `while` loops: AxiomV2Query.sol#L93, AxiomV2HeaderVerifier.sol#L92, AxiomV2HeaderVerifier.sol#L229, AxiomV2HeaderVerifier.sol#L239, AxiomV2HeaderVerifier.sol#L254, AxiomV2HeaderVerifier.sol#L264, AxiomV2HeaderVerifier.sol#L275, AxiomV2HeaderVerifier.sol#L315, MerkleTree.sol#L16, MerkleTree.sol#L21, MerkleMountainRange.sol#L35, MerkleMountainRange.sol#L47, MerkleMountainRange.sol#L58, MerkleMountainRange.sol#L73, MerkleMountainRange.sol#L104, MerkleMountainRange.sol#L146, MerkleMountainRange.sol#L181, PaddedMerkleMountainRange.sol#L63, AxiomV2Core.sol#L120, AxiomV2Core.sol#L204, AxiomV2Core.sol#L207, AxiomV2Core.sol#L235, AxiomV2Core.sol#L308.
- Bit shifts cost less gas than multiplying/dividing by 2:
  - AxiomV2Configuration.sol#L54-L55: `<< 6` can be used instead of `* 64` .
  - AxiomV2Query.sol#L544-L566 and AxiomV2Prover.sol#L93-L97: `<< 5` can be used instead of `* 32`.
  - MerkleTree.sol#L15-L16: `>> 1` can be used instead of `/ 2`.
  - MerkleTree.sol#L19: `>> 2` can be used instead of `/ 4`.
  - MerkleMountainRange.sol#L108-L114: `j << 1` can be used instead of `2 * j`.

Recommendation: Consider implementing the previous changes for gas optimization.

Axiom: Implemented the first two categories in PR 108.

The bit-shift optimizations did not seem to have an impact when using Foundry gas metering to measure.

Spearbit: Fixed.

5.3.6 Usage of custom errors and revert strings inconsistency

Severity: Gas Optimization

Context: AxiomV2Prover.sol#L38, AxiomV2Prover.sol#L39, AxiomV2Prover.sol#L40, AxiomV2Prover.sol#L41, AxiomV2Prover.sol#L42, AxiomResultStore.sol#L26, AxiomResultStore.sol#L27, AxiomResultStore.sol#L28, AxiomResultStore.sol#L29, AxiomV2HeaderVerifier.sol#L40, AxiomV2HeaderVerifier.sol#L41, AxiomV2HeaderVerifier.sol#L42, AxiomV2HeaderVerifier.sol#L43, AxiomV2HeaderVerifier.sol#L84, AxiomTimelock.sol#L25, PaddedMerkleMountainRange.sol#L81, PaddedMerkleMountainRange.sol#L108, AxiomV2Core.sol#L60, AxiomV2Core.sol#L61, AxiomV2Core.sol#L62, AxiomV2Core.sol#L66, AxiomV2Core.sol#L67, AxiomV2Core.sol#L68, ExampleV2Client.sol#L24, AxiomV2Client.sol#L26, AxiomV2Client.sol#L41, AxiomV2Prover.sol#L105, MerkleTree.sol#L63

Description: Starting from Solidity version 0.8.4, the introduction of custom errors has provided a more gas-efficient way to handle errors compared to the traditional `require` and `revert` with string messages. When a `require` statement fails or a `revert` is called with a string argument, the EVM needs to store the string in memory, which consumes a significant amount of gas (due to the usage of `MSTORE` opcodes). In contrast, custom errors, defined using the `error` keyword, allow developers to define specific error types with associated parameters, which can significantly reduce the gas cost when an error occurs.

In some parts of the code, custom errors are declared and later used, while in other parts, classic `revert` strings are used in `require` or `revert` statements. These could be replaced by custom errors, which would reduce deployment and runtime costs while contributing to codebase consistency. It also avoids wasting gas with long `revert` strings (each extra memory word of bytes past the initial 32 incurs in an additional MSTORE, which costs 3 units of gas).

Recommendation: Consider using custom errors exclusively to improve gas efficiency and increase codebase consistency.

Axiom: The issue has been addressed in PR 82.

Spearbit: Fixed.

5.3.7 Multiple `peakLength` typecasting to `uint32`

Severity: Gas Optimization

Context: AxiomV2HeaderVerifier.sol#L90

Description: `peaksLength` can be defined as `uint32` instead of `uint256` to avoid subsequent castings later on in the function.

Recommendation: In addition, consider renaming the variable to `proofMmmrPeaksLength` for clarity.

```
- uint256 peaksLength = mmrWitness.proofMmrPeaks.length;
+ uint32 proofMmmrPeaksLength = uint32(mmrWitness.proofMmrPeaks.length);
```

And then replacing all instances of `uint32(mmrWitness.proofMmrPeaks.length)` by `proofMmmrPeaksLength`.

Axiom: This issue has been addressed in PR 96.

Spearbit: Fixed.

5.3.8 `start` initialization can be optimized

Severity: Gas Optimization

Context: AxiomV2Core.sol#L215

Description: `start` is defined, calculated and casted to `uint32` every iteration. This can be made more optimal.

Recommendation: Define `start` outside the loop, then shift the increment to after the event emission so that only an addition by `BLOCK_BATCH_SIZE` is required.

```
uint32 start = startBlockNumber;
for (uint256 i; i < HISTORICAL_NUM_ROOTS; ++i) {
...
   emit HistoricalRootUpdated(start, prevHash, roots[i], BLOCK_BATCH_SIZE);
   start += BLOCK_BATCH_SIZE;
}
```

Axiom: We decided not to make this optimization as we expect `updateHistorical` to only be called in the initial bootstrap phase for `AxiomV2Core`.

Spearbit: Acknowledged.

5.3.9 Iterators can be optimized

Severity: Gas Optimization

Context: AxiomV2Core.sol#L206-L209, AxiomV2HeaderVerifier.sol#L264-L268

Description: In the first reference, a couple of subtractions are required for index calculation when iterating `endHashProofs` from the back. These subtractions can be avoided if iteration is performed from the start.

In the second reference, an addition operation can be removed by offsetting the iterator by `BLOCK_BATCH_DEPTH`.

Recommendation:

```
bytes32 proofCheck = endHashProofs[i][0];
for (uint256 j = 1; j <= BLOCK_BATCH_DEPTH; ++j) {
   proofCheck = Hash.keccak(proofCheck, endHashProofs[i][j]);
}
```

```
for (uint32 idx = BLOCK_BATCH_DEPTH; idx < mmrWitness.proofMmrPeaks.length; ++idx) {
  if (
     IAxiomV2State(axiomCoreAddress).blockhashPmmrPeaks(idx - BLOCK_BATCH_DEPTH)
        != proofMmr.peaks[idx]
  ) {
     revert ClaimedMMRDoesNotMatchRecent();
   }
}
```

Axiom: We decided not to implement this to reduce complexity of changes.

Spearbit: Acknowledged.

5.3.10 `CHAIN_ID` can be set to `internal` due to the existence of a custom getter

Severity: Gas Optimization

Context: AxiomV2HeaderVerifier.sol#L23

Description: `CHAIN_ID` is declared as a `public` variable, which leads to the solidity compiler to create for it a default getter function called `CHAIN_ID()`. Nevertheless, within `AxiomV2HeaderVerifier` there is already a manual implementation of a more readable getter function called `getSourceChainId()`.

Having two different getter functions defined for the same variable contributes to an unnecessary increase in contract code size (and consequently deployment gas costs), and the added redundancy may add obscurity to the codebase.

Recommendation: Consider having a single getter for `CHAIN_ID`, which yields a slightly shorter contract code size and thus slightly cheaper deployment costs. There are two options:

- Remove the `getSourceChainId()` function and use the default generated getter `CHAIN_ID()`.
- Declare `CHAIN_ID` as `internal`, which results in the absence of the `CHAIN_ID()` getter.

The second recommendation would be preferred, as `getSourceChainId()` is more explicit and readable.

Axiom: Fixed in PR 81 by declaring `CHAIN_ID` as `internal`.

Spearbit: Fixed.

5.3.11 User input validation should have preference over other actions

Severity: Gas Optimization

Context: AxiomResultStore.sol#L23-L29, AxiomV2HeaderVerifier.sol#L37-L43, AxiomV2Prover.sol#L35-L42, AxiomV2Query.sol#L58-L82, AxiomV2Query.sol#L246-L278, AxiomV2Query.sol#L301-L329, AxiomV2Query.sol#L351-L385, AxiomV2Query.sol#L457-L471

Description: User input validation on a function should precede any other action within the function scope. Otherwise, under the scenario in which execution arrives to a revert due to a wrong user input, gas is wasted on all the computation that has occurred prior to the check that yielded the revert.

Recommendation: Consider applying the following code refactorings to save some gas under the aforementioned scenarios:

- AxiomResultStore.sol#L23-L29:

```
  function initialize(address axiomQueryAddress, address timelock, address guardian, address unfreeze)
      public
      initializer
  {
-     __UUPSUpgradeable_init();
-     __AxiomAccess_init_unchained();

      require(axiomQueryAddress != address(0), "AxiomResultStore: axiomQueryAddress cannot be zero
↪  address");
      require(timelock != address(0), "AxiomResultStore: timelock cannot be zero address");
      require(guardian != address(0), "AxiomResultStore: guardian cannot be zero address");
      require(unfreeze != address(0), "AxiomResultStore: unfreeze cannot be zero address");

+     __UUPSUpgradeable_init();
+     __AxiomAccess_init_unchained();

  // ...
```

- AxiomV2HeaderVerifier.sol#L37-L43:

```
  function initialize(address _axiomCoreAddress, address timelock, address guardian, address unfreeze)
      public
      initializer
  {
-     __UUPSUpgradeable_init();
-     __AxiomAccess_init_unchained();

      require(_axiomCoreAddress != address(0), "AxiomV2HeaderVerifier: Axiom core address is zero");
      require(timelock != address(0), "AxiomV2HeaderVerifier: timelock address is zero");
      require(guardian != address(0), "AxiomV2HeaderVerifier: guardian address is zero");
      require(unfreeze != address(0), "AxiomV2HeaderVerifier: unfreeze address is zero");

+     __UUPSUpgradeable_init();
+     __AxiomAccess_init_unchained();
      axiomCoreAddress = _axiomCoreAddress;
      emit UpdateAxiomCoreAddress(_axiomCoreAddress);

  // ...
```

- AxiomV2Prover.sol#L35-L42:

```
  function initialize(
      address _axiomQueryAddress,
      address prover,
      address timelock,
      address guardian,
      address unfreeze
  ) public initializer {
-     __UUPSUpgradeable_init();
-     __AxiomAccess_init_unchained();

      require(_axiomQueryAddress != address(0), "AxiomV2Prover: _axiomQueryAddress cannot be zero
↪  address");
      require(prover != address(0), "AxiomV2Prover: prover cannot be zero address");
      require(timelock != address(0), "AxiomV2Prover: timelock cannot be zero address");
      require(guardian != address(0), "AxiomV2Prover: guardian cannot be zero address");
      require(unfreeze != address(0), "AxiomV2Prover: unfreeze cannot be zero address");

+     __UUPSUpgradeable_init();
+     __AxiomAccess_init_unchained();
      axiomQueryAddress = _axiomQueryAddress;
      emit UpdateAxiomQueryAddress(_axiomQueryAddress);

  // ...
```

- AxiomV2Query.sol#L58-L82:

```
  function initialize(AxiomV2QueryInit calldata init) public initializer {
-     __UUPSUpgradeable_init();
-     __AxiomAccess_init_unchained();
     if (init.axiomHeaderVerifierAddress == address(0)) {
         revert AxiomHeaderVerifierAddressIsZero();
     }
     if (init.verifierAddress == address(0)) {
         revert VerifierAddressIsZero();
     }
     if (init.axiomProverAddress == address(0)) {
         revert AxiomProverAddressIsZero();
     }
     if (init.axiomResultStoreAddress == address(0)) {
         revert AxiomResultStoreAddressIsZero();
     }
     if (init.timelock == address(0)) {
         revert TimelockAddressIsZero();
     }
     if (init.guardian == address(0)) {
         revert GuardianAddressIsZero();
     }
     if (init.unfreeze == address(0)) {
         revert UnfreezeAddressIsZero();
     }
+     __UUPSUpgradeable_init();
+     __AxiomAccess_init_unchained();
  // ...
```

- AxiomV2Query.sol#L246-L278:

```
  function fulfillQuery(
      IAxiomV2HeaderVerifier.MmrWitness calldata mmrWitness,
      bytes32[] calldata computeResults,
      bytes calldata proof,
      AxiomV2Callback calldata callback,
```

```
      AxiomV2QueryWitness calldata queryWitness
  ) external onlyRole(PROVER_ROLE) onlyNotFrozen {
-       uint256 queryId = uint256(
-           keccak256(
-               abi.encodePacked(
-                   queryWitness.caller,
-                   queryWitness.userSalt,
-                   queryWitness.queryHash,
-                   queryWitness.callbackHash,
-                   queryWitness.refundee
-               )
-           )
-       );
      if (queryWitness.callbackHash != keccak256(abi.encodePacked(callback.target,
↪   callback.extraData))) {
          revert CallbackHashDoesNotMatchQueryWitness();
      }

      AxiomProofCallbackData memory proofCallbackData = _verifyAndWriteResult(mmrWitness, proof);
      if (queryWitness.queryHash != proofCallbackData.queryHash) {
          revert QueryHashDoesNotMatchProof();
      }
+     if (proofCallbackData.computeResultsHash != keccak256(abi.encodePacked(computeResults))) {
+         revert ComputeResultsHashDoesNotMatch();
+     }

+       uint256 queryId = uint256(
+           keccak256(
+               abi.encodePacked(
+                   queryWitness.caller,
+                   queryWitness.userSalt,
+                   queryWitness.queryHash,
+                   queryWitness.callbackHash,
+                   queryWitness.refundee
+               )
+           )
+       );

      if (queries[queryId].state != AxiomQueryState.Active) {
          revert CannotFulfillIfNotActive();
      }
-     if (proofCallbackData.computeResultsHash != keccak256(abi.encodePacked(computeResults))) {
-         revert ComputeResultsHashDoesNotMatch();
-     }
  // ...
```

- AxiomV2Query.sol#L301-L329:

```
  function fulfillOffchainQuery(
      IAxiomV2HeaderVerifier.MmrWitness calldata mmrWitness,
      bytes32[] calldata computeResults,
      bytes calldata proof,
      AxiomV2Callback calldata callback,
      address caller,
      bytes32 userSalt
  ) external onlyRole(PROVER_ROLE) onlyNotFrozen {
      AxiomProofCallbackData memory proofCallbackData = _verifyAndWriteResult(mmrWitness, proof);

+     if (proofCallbackData.computeResultsHash != keccak256(abi.encodePacked(computeResults))) {
+         revert ComputeResultsHashDoesNotMatch();
+     }

      uint256 queryId = uint256(
          keccak256(
              abi.encodePacked(
                  caller,
                  userSalt,
                  proofCallbackData.queryHash,
                  keccak256(abi.encodePacked(callback.target, callback.extraData)),
                  address(0)
              )
          )
      );

-     if (proofCallbackData.computeResultsHash != keccak256(abi.encodePacked(computeResults))) {
-         revert ComputeResultsHashDoesNotMatch();
-     }

      if (queries[queryId].state != AxiomQueryState.Inactive) {
          revert CannotFulfillFromOffchainIfNotInactive();
      }
  // ...
```

- AxiomV2Query.sol#L351-L385:

```
  function refundQuery(AxiomV2QueryWitness calldata queryWitness) external onlyNotFrozen {
+     if (msg.sender != queryWitness.refundee) {
+         revert CannotRefundIfNotRefundee();
+     }
      uint256 queryId = uint256(
          keccak256(
              abi.encodePacked(
                  queryWitness.caller,
                  queryWitness.userSalt,
                  queryWitness.queryHash,
                  queryWitness.callbackHash,
                  queryWitness.refundee
              )
          )
      );

-     if (msg.sender != queryWitness.refundee) {
-         revert CannotRefundIfNotRefundee();
-     }

      if (queries[queryId].state != AxiomQueryState.Active) {
          revert CannotRefundIfNotActive();
      }
      if (block.number <= queries[queryId].deadlineBlockNumber) {
          revert CannotRefundBeforeDeadline();
      }

      balances[queryWitness.refundee] += queries[queryId].payment;

      queries[queryId] = AxiomQueryMetadata({
          state: AxiomQueryState.Inactive,
          deadlineBlockNumber: 0,
          payee: address(0),
          payment: 0
      });

      emit QueryRefunded(queryId, queryWitness.refundee);
  }
```

- AxiomV2Query.sol#L457-L471:

```
  function _sendQuery(uint256 queryId, uint64 maxFeePerGas, uint32 callbackGasLimit, address caller,
↪   uint256 deposit)
      internal
  {
      if (queries[queryId].state != AxiomQueryState.Inactive) {
          revert QueryIsNotInactive();
      }

-     uint256 maxQueryPri = _getMaxQueryPri(maxFeePerGas, callbackGasLimit);
      if (deposit > 0) {
          _recordDeposit(caller, deposit);
      }

+     uint256 maxQueryPri = _getMaxQueryPri(maxFeePerGas, callbackGasLimit);

      if (maxQueryPri > balances[caller]) {
          revert EscrowAmountExceedsBalance();
      }
```

Axiom: We decided not to make these changes to retain clarity (since the gas benefits are relatively small and only in the case that validation fails).

Spearbit: Acknowledged.

5.4 Informational

5.4.1 Naive provers may spend more funds to compute than the amount awarded to them

Severity: Informational

Context: AxiomV2Query.sol#L573

Description: In the current implementation, the `maxFeePerGas` within a query can be set to values close to zero. In the case of a naive prover fulfilling the query without checking for these low values, the prover may earn less funds than what costed them to perform the computations (for example if the query has set up a malicious callback that does expensive computation, as sometimes there are actual ways to profit from this, such as having the callback mint the XEN token).

Recommendation: Consider adding a warning to make sure that the prover is selective about each query's `maxFeePerGas`. Plus, making it something the prover calculates off-chain (versus adding more complexity in the contract to solve this) could save gas and be cheaper overall.

Axiom: We will offer an off-chain SDK which will choose a sensible value, and it seems the prover will need to be careful with query fulfillment / settings when faced with adversarial queries. Also, added a lower bound on `maxFeePerGas` in PR 100.

Spearbit: Technically the `increaseQueryGas()` function doesn't verify the `newMaxFeePerGas` against the lower bound. Do you think it's worth adding this for completeness? The function *does* check that `newMaxQueryPri > oldAmount` anyways, so it's not a big deal to leave it as-is.

Axiom: Agree, it makes sense to add validation. Implemented in PR 132.

Spearbit: Fixed.

5.4.2 Remove redundant code to compute `queryId`

Severity: Informational

Context: AxiomV2Query.sol#L253, AxiomV2Query.sol#L311, AxiomV2Query.sol#L352, AxiomV2Query.sol#L392

Description: The `queryId` variable is computed manually several times within `AxiomV2Query`, although there exists an internal function called `_computeQueryId` that performs a similar computation.

Recommendation: Consider refactoring `_computeQueryId` to a more general case to substitute the manual computations of `queryId` in the affected lines. This change would contribute to an improved code modularity and maintenance.

Axiom: This issue has been addressed in PR 98.

Spearbit: Fixed, together with a subsequent PR 104 to have all `queryId` computations utilise `_computeQueryId()`.

5.4.3 Passing ETH to callback target is unsupported

Severity: Informational

Context: AxiomV2Query.sol#L287, AxiomV2Query.sol#L336

Description: The current implementation does not allow to send ETH in the callback, because the `fulfillQuery` and `fulfillOffchainQuery` functions aren't payable and the call forwards 0 ETH by default. Some applications that may be called via this callback could work with WETH tokens, but others might specifically need ETH.

Recommendation: Consider documenting this limitations to avoid unsuccessful callbacks.

Axiom: This issue has been addressed in PR 97.

Spearbit: Fixed. The 0 ETH forwarding limitation has been documented in `IAxiomV2Query`:

```
/// The callback will correspond to the function signature of `axiomV2Callback` or
/// `axiomV2OffchainCallback` in `IAxiomV2Client`.  It is not payable, and no ETH will
/// be forwarded.
```

5.4.4 Callback call success is not monitored

Severity: Informational

Context: AxiomV2Query.sol#L287, AxiomV2Query.sol#L336,

Description: Within `fulfillQuery` and `fulfillOffchainQuery`, Axiom allows calling a callback target at the end of execution. The return of the low-level call is never checked nor emitted.

Recommendation: For the sake of monitoring and transparency, consider emitting either the success or failure of the callback call, as other operations may depend on it.

Axiom: This issue has been addressed in PR 97.

Spearbit: Verified. Note that if no callback is requested, the new event will have `callbackSucceeded` as `false`. Documentation about this behavior has been explicitly added in PR 133.

5.4.5 `frozen` state in `withdraw` introduces a centralization risk

Severity: Informational

Context: AxiomV2Query.sol#L421

Description: The system currently has an `onlyNotFrozen` mechanism implemented for both asset inflows (deposits) and outflows (withdrawals), as well as for payable mechanisms. While this serves as a protective measure against unforeseen bugs and malicious activities, it also introduces a centralization risk. The ability to freeze withdrawals means that assets could become locked within the system, reducing user trust and potentially harming the credibility of the platform.

Recommendation: To mitigate this centralization risk and increase user trust in the system, it is recommended to allow withdrawals even in a paused or frozen state. This ensures that users can retrieve their assets regardless of the system's state, preventing asset lock-up.

However, it is important to balance this change with the need for security:

> "We had initially chosen `onlyNotFrozen` on `withdraw` in order to avoid scenarios where some unforeseen bug allows a user to drain all the funds in the system. It's a fair point that removing that can decrease centralization, especially as we don't intend the system to hold substantial funds beyond what's currently being used for transacting."

The team's initial decision to restrict withdrawals under the `onlyNotFrozen` condition was made to protect against potential vulnerabilities that could lead to asset drainage. To address this concern while still enabling withdrawals during a freeze or pause state, additional security measures and monitoring should be implemented to quickly detect and respond to any unusual or malicious activity.

Axiom: The fix has been implemented in PR 99.

In deployment, as part of our standard procedures we plan to actively monitor:

- Any activity on permissioned multisigns.
- Any unusual withdrawals or balance transfers.

Spearbit: Fixed. Withdrawals can no longer be frozen.

5.4.6 Missing indexed event parameters

Severity: Informational

Context: ExampleV2Client.sol#L7, IAxiomV2HeaderVerifier.sol#L15, IAxiomV2Query.sol#L163-L195, IAxiomV2Query.sol#L248, IAxiomResultStore.sol#L8, IAxiomV2Prover.sol#L10, IAxiomV2Events.sol#L19-L32

Description: Axiom contracts make extensive use of events to log changes in state and facilitate off-chain integrations. However, multiple events across the code don't have any parameter indexed. Indexing fields within events allows off-chain tools to filter and access these fields more quickly, improving the efficiency of data retrieval.

Notice that, while indexing is beneficial for data retrieval, it comes with a cost for each one. Each indexed field in an event costs additional gas when the event is emitted, therefore, looking for a balance where the most significant parameters are the ones indexed it's really important.

Recommendation: Consider adding indexed keyword to events where some fields truly need to be indexed for later analysis.

Axiom: This issue has been addressed in PR 95.

Spearbit: Fixed.

5.4.7 Use named imports to improve clarity and efficiency

Severity: Informational

Context: IAxiomV2Query.sol#L4-L5, AxiomV2Client.sol#L4, AxiomV2Core.sol#L12, AxiomV2HeaderVerifier.sol#L15, AxiomV2Query.sol#L14

Description: `IAxiomV2Query` contract currently imports the entire `IAxiomV2Oracle` and `IAxiomV2Client` interfaces, even though it may not be utilizing all the functions or variables defined within these interfaces. This approach can lead to unnecessary bloating of the contract's bytecode, potentially increasing deployment costs and making the codebase harder to read and maintain.

Another relevant case is regarding `AxiomV2Configuration`, used multiple times in the code to retrieve different constant values but not using most of them.

Recommendation: Adopt named imports to explicitly state which functions or variables are being used from the imported interfaces, this would improve readability and maintainability by making the dependencies of the contract more explicit.

Axiom: This issue has been addressed in PR 86.

Spearbit: Fixed.

5.4.8 Usage of `axiomProverAddress` can be simplified

Severity: Informational

Context: AxiomV2Query.sol

Description: For an address to fulfill a query in the `AxiomV2Query` contract, it needs to have the `PROVER_ROLE`. At initialization, this role is granted to the `axiomProverAddress`, which is an entirely separate contract that has its own access control management.

So, there are technically two different ways that a query can be fulfilled by an address:

1. If the address is granted the `PROVER_ROLE` in the `AxiomV2Query` contract directly.
2. If the address has sufficient permission in the `AxiomV2Prover` (this is based on a combination of its own `PROVER_ROLE` and the `allowedProvers` mapping).

Recommendation: Since the `AxiomV2Prover` has its own access management, consider simplifying the `AxiomV2Query` to not use the `PROVER_ROLE`.

Instead, the `fulfillQuery()` and `fulfillOffchainQuery()` functions could directly require that `msg.sender == axiomProverAddress`. Since the `axiomProverAddress` is not currently used in the `AxiomV2Query` contract (other than as a public storage variable), it might have been intended that the `PROVER_ROLE` was only ever granted to the `axiomProverAddress`.

Axiom: Fixed in PR 87.

Spearbit: Fixed.

5.4.9 Open up access to `addAllowedProver` and `removeAllowedProver`

Severity: Informational

Context: AxiomV2Prover.sol#L80-L87

Description: Permission management of provers to callback targets is restricted to the `TIMELOCK_ROLE`, which could be too restrictive. This is because the callback target is a user-specified param, so it might be better to allow the callback target to add & remove provers by themselves.

Recommendation: Instead of `onlyRole(TIMELOCK_ROLE)`, restrict the function caller to `target`.

Axiom: When implementing, we realized allowing both `TIMELOCK_ROLE` and `target` is not that easy in our current permissions model, so we decided to postpone any changes here for the future.

Spearbit: Acknowledged.

5.4.10 PMMR `size` initialization can be more accurate

Severity: Informational

Context: AxiomV2HeaderVerifier.sol#L100-L109, AxiomV2HeaderVerifier.sol#L198-L209

Description: In both main `verifyQueryHeaders()` cases, a PMMR is initialized with a size that may not be divisible by `BLOCK_BATCH_SIZE`, despite the `paddedLeaf` always being initialized to `bytes32(0)`. So, the `size` can initially be too large for what the PMMR represents. Fortunately, this is always resolved in a follow-up call to `updatePaddedLeaf()`, but it may be easier to understand the code if the `size` is initialized correctly.

Recommendation: Change the PMMR `size` initialization as follows:

```
PaddedMerkleMountainRange.PMMR memory proofPmmr = PaddedMerkleMountainRange.PMMR({
    paddedLeaf: bytes32(0),
    completeLeaves: MerkleMountainRange.MMR({
        peaks: _copyPeaks(
            mmrWitness.proofMmrPeaks, BLOCK_BATCH_DEPTH, mmrWitness.proofMmrPeaks.length -
↪   BLOCK_BATCH_DEPTH
            ),
        peaksLength: uint32(mmrWitness.proofMmrPeaks.length) - BLOCK_BATCH_DEPTH
    }),
-   size: proofMmrSize
+   size: proofMmrSize - (proofMmrSize % BLOCK_BATCH_SIZE)
});
```

```
PaddedMerkleMountainRange.PMMR memory snapshotPmmr = PaddedMerkleMountainRange.PMMR({
    paddedLeaf: bytes32(0),
    completeLeaves: MerkleMountainRange.MMR({
        peaks: _copyPeaks(
            mmrWitness.mmrComplementOrPeaks,
            BLOCK_BATCH_DEPTH,
            uint32(mmrWitness.mmrComplementOrPeaks.length) - BLOCK_BATCH_DEPTH
            ),
        peaksLength: uint32(mmrWitness.mmrComplementOrPeaks.length) - BLOCK_BATCH_DEPTH
    }),
-   size: mmrWitness.snapshotPmmrSize
+   size: mmrWitness.snapshotPmmrSize - (mmrWitness.snapshotPmmrSize % BLOCK_BATCH_SIZE)
});
```

Axiom: Fixed in PR 89.

Spearbit: Fixed.

5.4.11 Idempotent checks may be performed to avoid emitting events with no changes

Severity: Informational

Context: AxiomV2Query.sol#L114-L131, AxiomV2Core.sol#L257-L270

Description: `TIMELOCK_ROLE` may call some important functions as `updateAxiomProverAddress`, `updateAxiomHeaderVerifierAddress` or `updateVerifierAddress` with the actual value or call twice the same script that set these values.

This can confuse people listening for important event emissions such as the change of the different actors addresses, that in fact, didn't changed.

Recommendation: Consider to add a check to validate if the input value it's not the actual value.

Axiom: Acknowledged. We expect these functions to be called only for:

- Major protocol upgrades.
- Recoveries from security incidents.

As a result, we expect them to be called extremely infrequently and would expect these event emissions to be subject to manual inspection. As a result, we don't think filtering out idempotent updates would be valuable here.

Spearbit: Acknowledged.

5.4.12 Favor Yul `switch-case` statements over multiple `if` statements

Severity: Informational

Context: MerkleTree.sol#L33-L63

Description: `MerkleTree` has a function called `getEmptyHash()` that returns the hash image of the Merkle root of a tree of a user specified depth `idx` that has all zeroes as leaves. For the sake of optimization, these hashes are pre-computed and returned by checking the user input `idx` on a series of `if` statements.

The aforementioned logic can be easily implemented in `Yul` through the `switch-case` statement, which is cheaper gas-wise and makes the cases more readable, at the cost of adding some obscurity to the revert case (which occurs when `idx >= BLOCK_BATCH_DEPTH`).

An example implementation would look as follows:

```
function getEmptyHash(uint256 depth) internal pure returns (bytes32 emptyHash) {
    assembly {
        switch depth
        case 0 { emptyHash := 0x0 }
        case 1 { emptyHash := 0xad3228b676f7d3cd4284a5443f17f1962b36e491b30a40b2405849e597ba5fb5 }
        case 2 { emptyHash := 0xb4c11951957c6f8f642c4af61cd6b24640fec6dc7fc607ee8206a99e92410d30 }
        case 3 { emptyHash := 0x21ddb9a356815c3fac1026b6dec5df3124afbadb485c9ba5a3e3398a04b7ba85 }
        case 4 { emptyHash := 0xe58769b32a1beaf1ea27375a44095a0d1fb664ce2dd358e7fcbfb78c26a19344 }
        case 5 { emptyHash := 0x0eb01ebfc9ed27500cd4dfc979272d1f0913cc9f66540d7e8005811109e1cf2d }
        case 6 { emptyHash := 0x887c22bd8750d34016ac3c66b5ff102dacdd73f6b014e710b51e8022af9a1968 }
        case 7 { emptyHash := 0xffd70157e48063fc33c97a050f7f640233bf646cc98d9524c6b92bcf3ab56f83 }
        case 8 { emptyHash := 0x9867cc5f7f196b93bae1e27e6320742445d290f2263827498b54fec539f756af }
        case 9 { emptyHash := 0xcefad4e508c098b9a7e1d8feb19955fb02ba9675585078710969d3440f5054e0 }
        default { // This could be further optimized if needed
            let ptr := mload(0x40)
            mstore(
                ptr,
                // bytes4(keccak256(Error(string))) << 224
                0x08c379a000000000000000000000000000000000000000000000000000000000
            )
            mstore(add(ptr, 0x04), 0x20)
            mstore(add(ptr, 0x24), 30)
            mstore(add(ptr, 0x44), "depth must be in range [0, 10)")
            revert(ptr, 0x64)
        }
    }
}
```

Recommendation: Consider using Yul `switch-case` over multiple consecutive `if` statements for the sake of readability and reducing gas costs.

Axiom: Acknowledged. We think keeping the existing `if` structure will be simpler for clarity for now.

Spearbit: Acknowledged.

5.4.13 Goerli testnet will stop working in 2024

Severity: Informational

Context: AxiomV2Configuration.sol#L64

Description: The Goerli testnet is scheduled to be deprecated and will stop working in 2024. As this testnet is the only environment for testing and development purposes in the project, it is important to start planning a change to ensure a smooth transition and continuous operation of our applications and smart contracts.

Recommendation: It is recommended to add support for alternative testnets such as Sepolia or Holesky to ensure that there is a environment for testing and development once Goerli is no longer available.

Axiom: This issue has been addressed in PR 76 by adding the additional testnet chain IDs in `AxiomV2Configuration.sol`.

Spearbit: Fixed.

5.4.14 Variable naming improvements

Severity: Informational

Context: AxiomV2Query.sol#L456-L457, AxiomV2HeaderVerifier.sol#L226

Description: The following variables can be renamed to avoid shadowing, or for greater clarity:

```
- uint256 deposit
+ uint256 depositAmount

- uint256 appendLeft
+ uint256 appendRemaining
```

Recommendation: As per description.

Axiom: Fixed in PR 78.

Spearbit: Fixed.

5.4.15 Incorrect `peaksLength` value set for MMRs

Severity: Informational

Context: AxiomV2HeaderVerifier.sol#L113, AxiomV2HeaderVerifier.sol#L113

Description: `proofBatchMmr` and the MMR used for the `snapshotLeaf` generation has their sizes incorrectly set to `BLOCK_BATCH_SIZE` instead of `BLOCK_BATCH_DEPTH`.

There aren't any repercussions in the current implementation because `getComplementMerkleRoot(BLOCK_BATCH_DEPTH, ...)` / `getZeroPaddedMerkleRoot(BLOCK_BATCH_DEPTH)` is called respectively after, which only utilises `BLOCK_BATCH_DEPTH` peaks, with no further usage of these structures.

Recommendation:

```
- peaksLength: BLOCK_BATCH_SIZE
+ peaksLength: BLOCK_BATCH_DEPTH
```

Axiom: The issue has been addressed in PR 79.

Spearbit: Fixed.

5.4.16 `virtual` is redundant in interfaces and it can be removed

Severity: Informational

Context: IAxiomV2Client.sol#L15-L31

Description: The functions `axiomV2Callback()` and `axiomV2OffchainCallback()` in the `IAxiomV2Client` interface are both declared with the `virtual` keyword. This is no longer necessary as of Solidity version 0.8.0, as functions in interfaces are implicitly `virtual`.

Recommendation: Remove the `virtual` keyword from these function declarations to clean up the code and adhere to best practices. This will not change the functionality of the code, but it will make it cleaner and more in line with current Solidity standards.

Axiom: This issue has been addressed in PR 77.

Spearbit: Fixed.

5.4.17 Unused logic and statements

Severity: Informational

Context: AxiomV2HeaderVerifier.sol#L4, AxiomV2HeaderVerifier.sol#L12, AxiomV2Prover.sol#L4, IAxiomV2Verifier.sol#L4, IAxiomV2HeaderVerifier.sol#L4, IAxiomV2Query.sol#L4, IAxiomV2Query.sol#L144, IAxiomV2Query.sol#L150, IAxiomV2Query.sol#L152, IAxiomV2Query.sol#L156, PaddedMerkleMountainRange.sol#L4, AxiomAccess.sol#L58-L63, AxiomAccess.sol#L73-L75, AxiomV2HeaderVerifier.sol#L275-L277

Description: The affected files contain unused logic or statements. This may be a leftover from either the development or testing stages of the protocol. See below for the detailed list:

- Unused imports: AxiomV2HeaderVerifier.sol#L4, AxiomV2HeaderVerifier.sol#L12, AxiomV2Prover.sol#L4, IAxiomV2Verifier.sol#L4, IAxiomV2HeaderVerifier.sol#L4, IAxiomV2Query.sol#L4, PaddedMerkleMountainRange.sol#L4.
- Unused errors: IAxiomV2Query.sol#L144, IAxiomV2Query.sol#L150, IAxiomV2Query.sol#L152, IAxiomV2Query.sol#L156.
- Unused modifiers: AxiomAccess.sol#L58-L63.
- Unused functions: AxiomAccess.sol#L73-L75.
- Redundant logic: AxiomV2HeaderVerifier.sol#L275-L277.

Recommendation: Consider removing the statements and logic from the affected files to improve readability.

Axiom: This issue has been addressed in PR 80, except for AxiomAccess.sol#L73-L75 to conform to the OpenZeppelin convention.

Spearbit: Fixed.

5.4.18 Absent/incomplete natspec affects readability and maintenance

Severity: Informational

Context: IAxiomV2Client.sol#L4-L32, AxiomTimelock.sol#L24-L29, AxiomTimelock.sol#L19-L22, AxiomProxy.sol#L10, IAxiomV2State.sol#L17, IAxiomV2Query.sol#L115, AxiomV2Configuration.sol#L35, Hash.sol#L5-L11, PaddedMerkleMountainRange.sol#L76, PaddedMerkleMountainRange.sol#L103, IAxiomV2Verifier.sol#L30, AxiomV2Core.sol#L310, AxiomV2Query.sol#L532, MerkleMountainRange.sol#L87, MerkleMountainRange.sol#L117, PaddedMerkleMountainRange.sol#L22, PaddedMerkleMountainRange.sol#L58, IAxiomV2Query.sol#L36, AxiomV2HeaderVerifier.sol#L71, AxiomV2HeaderVerifier.sol#L82, IAxiomV2Events.sol#L5-L11, IAxiomV2HeaderVerifier.sol#L7-L11, IAxiomV2Verifier.sol#L9-L15, AxiomAccess.sol#L73-L89, Hash.sol#L4, MerkleMountainRange.sol#L91, AxiomV2Core.sol#L52, AxiomAccess.sol#L55, AxiomV2Client.sol#L17-L65, AxiomV2Client.sol#L6-L16, ExampleV2Client.sol#L6-L14, ExampleV2Client.sol#L16-L42, IAxiomV2Query.sol#L8-L18, AxiomProxy.sol#L10

Description: Comments are key to understanding the codebase logic. In particular, Natspec comments provide rich documentation for functions, return variables and more. This documentation aids users, developers and auditors in understanding what the functions within the contract are meant to do.

However, some functions within the codebase contain issues with respect to their comments with either no Natspec or incomplete Natspec annotations, leading to partial descriptions of the functions.

Recommendation: Comments are key to understanding the codebase logic. In particular, Natspec comments provide rich documentation for functions, return variables and more. This documentation aids users, developers and auditors in understanding what the functions within the contract are meant to do.

However, some functions within the codebase contain issues with respect to their comments with either no Natspec or incomplete Natspec annotations, leading to partial descriptions of the functions.

- Missing Natspec: IAxiomV2Client.sol#L4-L32, AxiomTimelock.sol#L24-L29, AxiomTimelock.sol#L19-L22, AxiomProxy.sol#L10, AxiomV2Core.sol#L52, AxiomV2Client.sol#L6-L16, ExampleV2Client.sol#L6-L14, AxiomProxy.sol#L10
- Incomplete Natspec:
  - IAxiomV2State.sol#L17. Missing `startBlockNumber` param natspec.
  - IAxiomV2Query.sol#L115
  - AxiomV2Configuration.sol#L35
  - Hash.sol#L5-L11
  - PaddedMerkleMountainRange.sol#L76. Missing `paddingSize` param natspec.
  - PaddedMerkleMountainRange.sol#L103. Missing `paddingSize` param natspec.
- Unclear comment
  - IAxiomV2Verifier.sol#L30. The description seems like it should be similar to the natspec for `endHashProofs[i][j]` of `updateHistorical()`: `endHashProofs[i][j] is the sibling of the Merkle node at depth j, for j = 0, ..., 9.`
  - AxiomV2Core.sol#L310. It's unclear which expression the comment is referring to that evaluates to 0 / 1, and which variable on the left / right is referred to. Perhaps it should be something like `// if i-th bit = 1, proof is on the left, else, proof is on the right.`
  - IAxiomV2Query.sol#L8-L18. Missing documentation regarding difference in status from off-chain and on-chain query status. "We call a query on-chain if it is requested on-chain with `sendQuery` or `sendQueryWithIpfs` data. We also allow a query to be fulfilled on-chain without an on-chain request (which we call off-chain), but to make sure the off-chain queries don't interfere with on-chain queries, we require that they are in status `Inactive`."
- Incorrect comment
  - AxiomV2Query.sol#L532:

```
- 16 groups of 32
+ 14 groups of 32
```

  - MerkleMountainRange.sol#L87:

```
- appendSingle
+ appendLeaf
```

  - MerkleMountainRange.sol#L117:

```
- to_add
+ toAdd
```

  - PaddedMerkleMountainRange.sol#L22:

```
- The an MMR of the complete leaves of the PMMR
+ An MMR of the complete leaves of the PMMR
```

  - PaddedMerkleMountainRange.sol#L58:

```
- concatenatation
+ concatenation
```

  - IAxiomV2Query.sol#L36:

```
- AxiomResultstore
+ AxiomResultStore
```

  - AxiomV2HeaderVerifier.sol#L71:

```
- committment
+ commitment
```

  - AxiomV2HeaderVerifier.sol#L82:

```
- transation
+ transaction
```

  - IAxiomV2HeaderVerifier.sol#L29-L30:

```
It is expected to be a Merkle root of a padded leaf exactly when
-  snapshotPmmrSize - (snapshotPmmrSize % BLOCK_BATCH_SIZE) >= proofPmmrSize
+  snapshotPmmrSize % BLOCK_BATCH_SIZE != 0
```

- Missing comment / description
  - IAxiomV2Events.sol#L5-L11
  - IAxiomV2HeaderVerifier.sol#L7-L11
  - IAxiomV2Verifier.sol#L9-L15
  - AxiomAccess.sol#L73-L89
  - Hash.sol#L4
  - MerkleMountainRange.sol#L91. Users of this library may assume that the values of leaves are unchanged. It's best to leave a warning that leaves will be mutated (also have the warning in PMMR.appendCompleteLeaves() since it also uses this function.
  - AxiomAccess.sol#L55 is missing `* Granting the AXIOM role to address(0) will enable this role for everyone.`
  - AxiomProxy.sol#L10
- `@inheritdoc` can be used for inherited Natspec: AxiomV2Client.sol#L17-L65, ExampleV2Client.sol#L16-L42.

Axiom: This issue has been addressed in PR 75.

All changes have been made aside from:

- IAxiomV2Verifier.sol#L30. We believe the comment is actually accurate.

Spearbit: Mostly fixed, although a couple more spelling mistakes were caught:

```
- pulbic
+ public

// From PR 106
- OnlyPayeeCanFullfillOffchainQuery
+ OnlyPayeeCanFulfillOffchainQuery
```

Axiom: Addressed in PR 131, thanks for catching these.

Spearbit: Fixed.

6 Additional Comments

The Spearbit team reviewed Axiom's contracts-v2-working holistically on PR 74 and determined that all issues were resolved and no new issues were identified.