Redacted Cartel Security Review

Auditors

Christoph Michel, Lead Security Researcher

High byte, Security Researcher

Report prepared by: Pablo Misirov

September 19, 2023

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 High Risk . . . 4
5.1.1 Rate limit circumvention & backend memory exhaustion . . . 4
5.2 Medium Risk . . . 5
5.2.1 `getLogs` returns duplicate logs at the chunk boundaries . . . 5
5.2.2 Slow down request middleware does currently not work . . . 5
5.3 Low Risk . . . 6
5.3.1 Ribbon task not checking proposal deadline . . . 6
5.3.2 Imprecise gauge proposal fetching in tasks . . . 6
5.3.3 0x price impact protection . . . 6
5.3.4 `getCurrentBlock` returns block number of block that can still be re-orged . . . 7
5.3.5 Imprecise `ERC20.permit` support detection . . . 7
5.3.6 Limit CORS to bare necessities . . . 8
5.4 Informational . . . 8
5.4.1 Add tests and documentation . . . 8
5.4.2 `chainId` is not appended to error . . . 8
5.4.3 Gnosis chain is used in scripts but not defined in `constants.js` . . . 9
5.4.4 leverage finally block when doing early return . . . 9
5.4.5 `fetchBribesByProtocolAndTxHash` uses same deadline for coin-historical-records . . . 10
5.4.6 `try/catch` blocks for async-redis are never hit . . . 11

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

Off-chain reconciliation services that power Hidden Hand markets

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of hidden-hand-backend 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 5 days in total, Redacted Cartel engaged with Spearbit to review the hidden-hand-backend protocol. In this period of time a total of **15** issues were found.

Summary

Project Name	Redacted Cartel
Repository	hidden-hand-backend
Commit	9c8deb...e7c2
Type of Project	Back-End, Web2
Audit Timeline	Aug 7 - Aug 11
Two week fix period	Aug 11 - Aug 25

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	2	1	1
Low Risk	6	2	4
Gas Optimizations	0	0	0
Informational	6	2	4
Total	**15**	**6**	**9**

5 Findings

5.1 High Risk

5.1.1 Rate limit circumvention & backend memory exhaustion

Severity: High Risk

Context: util/get-user-ip-address.js#L1

Description: The client IP address is first fetched from the `x-forwarded-for` header if it exists. The left-most (first) value of the comma-separated list is used.

```
module.exports = req => (req.headers['x-forwarded-for'] || '').split(',')[0]
  || req.connection.remoteAddress
  || req.socket.remoteAddress
  || (req.connection.socket ? req.connection.socket.remoteAddress : null);
```

Reverse proxies and other infrastructure append the connected socket IP to the right of this header when forwarding the request. The left-most IP may not be used to identify the client IP as this value can be spoofed by the client through setting the header to a custom fake IP address in the initial request.

Therefore, the current rate-limiting can be circumvented. Furthermore, it is not checked if the value is indeed an IP address. An attacker can submit long strings in the size of megabytes which will then be stored in the rate limiter object. An attacker can fill up the backend's memory using this technique leading to memory exhaustion and likely crashing of the backend app.

Recommendation: The right-most trusted value in the `x-forwarded-for` header should be used instead. For example, if there is one reverse proxy deployed in the infrastructure before the backend app, the last value should be used. For two reverse proxies, the second last, etc. If the backend is directly connected to the internet, the header shouldn't be used and one should directly use the client IP. We also recommend trimming the values after comma-separation (because the standard is ambiguous about whitespace) and checking that the value is a valid IP address.

For a further deep dive on this subject of getting the real client IP, also see this article.

> If you're going to do something security-related, you need to use the IP you trust – the rightmost-ish. The obvious example here is rate-limiting. If you use the leftmost-ish IP for this, an attacker can just spoof a different XFF prefix value with each request and completely avoid being limited.

Redacted: Fixed in commit 044619f3.

Spearbit: The code now takes the right-most x-forwarded-for header to match their deployment configuration. Fixed.

5.2 Medium Risk

5.2.1 `getLogs` returns duplicate logs at the chunk boundaries

Severity: Medium Risk

Context: common.js#L184

Description: The `getLogs` function gets the logs in chunks of `blockRange`, starting from the `startBlock` until the `endBlock`. The start block of the next iteration is set to the old end block:

```
const toBlock =
  endBlock - fromBlock > blockRange ? fromBlock + blockRange : endBlock;
const logs = await contract.queryFilter(event, fromBlock, toBlock);

allLogs.push(...logs);

fromBlock = toBlock;
```

Both bounds are **inclusive**, events in `fromBlock` and `toBlock` will also be returned. This leads to returning duplicate logs for the blocks at the inner boundaries of the iteration. For example, duplicate bribe deposit events could be detected.

Recommendation: Consider setting the `fromBlock = toBlock + 1` for subsequent iterations and ensure it is not greater than the `maxBlock`.

Redacted: Fixed in commit 9a948341.

Spearbit: Fixed.

5.2.2 Slow down request middleware does currently not work

Severity: Medium Risk

Context: index.js#L55

Description: Both the rate limiter and speed limiter take effect after 2000 requests / minute per client IP.

> Slow down the remaining requests before blocking it for exceeding the request limit set above

However, this currently does not work as all requests go through both middlewares, the slow down is only activated at the same time the rate limiter is already hit. The idea is to:

Recommendation: The values should not be the same, consider reducing the speed limiter's `delayAfter` value. 2000 request / minute per IP is also a lot of requests that are allowed, consider reducing the values.

Redacted: Currently the rate limiting feature isn't exactly being useful after the v2 migration. Will be removed altogether once vercel migration is completed.

Spearbit: Acknowledged.

5.3 Low Risk

5.3.1 Ribbon task not checking proposal deadline

Severity: Low Risk

Context: tasks/ribbon-proposal.js#L49

Description: The ribbon task fetches the latest proposal but does not check if the proposal is active.

Recommendation: Consider checking `proposalDeadline > currentTimestamp`, similar to what the Aura task is doing.

Redacted: Addressed on commit 0a59f251.

Spearbit: Fixed.

5.3.2 Imprecise gauge proposal fetching in tasks

Severity: Low Risk

Context: tasks/ribbon-proposal.js#L47, tasks/floordao-proposal.js#L47, tasks/aura-proposal.js#L110

Description: Some tasks fetch proposals and check for gauge proposals by looking for words like "Gauge" in the title. If anyone can create proposals for the protocol, it's easy to fake a gauge proposal and the script will pick up the wrong one.

Recommendation: Consider improving the checks on the proposal filtering. For example, if gauge proposals are always proposed by a certain governance address, check the proposal owner for that address.

Redacted: The decision was made based on each protocol's team rep confirmation on the usage of such pattern for gauge voting proposals for consistency sake. In the case where there's any changes in the way the title is composed, this would be updated appropriately.

Spearbit: Acknowledged.

5.3.3 0x price impact protection

Severity: Low Risk

Context: tasks/harvester-swap-arbitrum.js#L214

Description: The 0xAPI swap quote uses the `slippagePercentage` field to protect against slippage but no price impact protection with the `priceImpactProtectionPercentage` parameter. See the documentation for the difference between these two terms.

Recommendation: Consider using the `priceImpactProtectionPercentage` field to protect against trading order sizes that are too large for illiquid tokens.

Redacted: Addressed on commit f250f73f.

Spearbit: Fixed.

5.3.4 `getCurrentBlock` returns block number of block that can still be re-orged

Severity: Low Risk

Context: lib/common.js#L29, lib/common.js#L175

Description: The `getCurrentBlock` function uses the `latest` block-tag for the head block which is defined as:

- `earliest`: The lowest numbered block the client has available;
- `finalized`: The most recent crypto-economically secure block, cannot be re-orged outside of manual intervention driven by community coordination;
- `safe`: The most recent block that is safe from re-orgs under honest majority and certain synchronicity assumptions;
- `latest`: The most recent block in the canonical chain observed by the client, this block may be re-orged out of the canonical chain even under healthy/normal conditions;
- `pending`: A sample next block built by the client on top of `latest` and containing the set of transactions usually taken from local mempool.

Before the merge transition is finalized, any call querying for `finalized` or `safe` block MUST be responded to with `-39001: Unknown block` error

See api-documentation `getBlockByNumber`.

This block number is used throughout the code, also to fetch logs. It could be that it fetches logs for a head block that is re-orged later and does not include the transactions leading to the logs anymore.

Recommendation: Consider using a safer block tag that can't be re-orged for ETH mainnet and chains that support this RPC setting, and a buffer for other chains.

Spearbit: Marked as acknowledged.

5.3.5 Imprecise `ERC20.permit` support detection

Severity: Low Risk

Context: lib/common.js#L134-L153

Description: If a token supports permit is currently done by checking if any of these is true:

- `token.DOMAIN_SEPARATOR()` returns a value
- `token.PERMIT_TYPEHASH()` returns a value
- `token.nonces(address)` returns a value

The first and last indicators are also used if the contract uses EIP712 signatures but not necessarily permit.

Recommendation: Consider improving the `permit` detection, or manually keep a false positive list that is checked against, in case false positives are ever encountered.

Spearbit: Marked as acknowledged.

5.3.6 Limit CORS to bare necessities

Severity: Low Risk

Context: cors.sol#L26

Description: Currently the cors config is set to '*' which means no restriction. It is best practice to limit access only to the expected endpoints.

Recommendation: Change allowed origins to the exact front-end origin. Possibly also limit which methods are allowed to reduce attack surface.

Redacted: We originally used cors but was causing issue with our setup in the front-end side, so we removed it for the time being. Post migration to vercel the setting will no longer be needed.

Spearbit: Acknowledged.

5.4 Informational

5.4.1 Add tests and documentation

Severity: Informational

Context: Entire code base.

Description: The code base currently has no tests.

Recommendation: We strongly recommend testing the codebase to ensure the correctness of the scripts and tasks related to the distributions. Also consider documenting the interplay of the various tasks and scripts, with a list of what happens each round.

Spearbit: Marked as acknowledged.

5.4.2 `chainId` is not appended to error

Severity: Informational

Context: tasks/pendle-proposal.js#L121-L127

Description: The function throws an error with a message like this: `throw new Error('Missing credentials', chainId);`. However, the `Error` constructor does not take two arguments, the `chainId` will not be appended to the message.

Recommendation: Consider using a string template with the chainId:

```
throw new Error(`Missing credentials ${chainId}`)
```

Redacted: Acknowledged, with potential fixes on commit f623cdd2, which also covers similar issues in other part of the codebase

Spearbit: Fixed.

5.4.3 Gnosis chain is used in scripts but not defined in `constants.js`

Severity: Informational

Context: tasks/balancer-proposal.js#L35

Description: Chain ID 100 which corresponds to Gnosis is used in the `tasks/balancer-proposal` script but the `constants.js` file that should list all known chains does not list it.

Recommendation: Consider adding Gnosis chain to the constants. Also consider using the constants mapping to get the chain IDs from human readable chain names instead of hardcoding them in the `task` and `script` files.

Redacted: Originally tasks were not part of the repo (and will be moved out on the serverless migration). The gnosis chain was recently added after balancer added gauges for pools in the chain (and also only for the purpose of identifying gauge name/title), and is not currently being used anywhere else on the original backend codebase, thus not being added on the main constants file.

Spearbit: Acknowledged.

5.4.4 leverage finally block when doing early return

Severity: Informational

Context: schedulers such as ribbon.js#L68 and others

Description: In schedulers you use the common pattern, generally:

```
  try {
    if (await isSchedulerLocked(schedulerType)) {
      console.log('Identical scheduler is still running');
      return;
    }

    await enableSchedulerLock(schedulerType);

    ...
}
```

but sometimes there is an early return which requires releasing the lock as well, eg:

```
    if (!latestDeadline) {
      // No proposal found for the protocol
      disableSchedulerLock(schedulerType);
      return;
    }
```

Note two things:

1. This can be avoided by using `finally` block to release the lock, thus reducing amount of repetitive code and avoiding pitfalls in the future in case of forgetting to release a lock.
2. Notice that `isSchedulerLocked` is currently inside the `try-catch` block whereas a leading `disableSchedulerLock` follows outside it. this means if `isSchedulerLock` throws for whatever reason, the lock will be attempted to be released.

I recommend following best practice of DRY and perhaps consider moving the `isSchedulerLocked` outside the `try-catch` (maybe just let such exception bubble up?)

Redacted: Fixed in commit b892a2c0.

Spearbit: Fixed.

5.4.5 `fetchBribesByProtocolAndTxHash` uses same deadline for coin-historical-records

Severity: Informational

Context: controllers/bribe-controller.js#L21-L94, controllers/proposal-controller.js#L67-L104

Description: The `fetchBribesByProtocolAndTxHash` function iterates over bribes and sets coin-historical-records. The same `historicalDeadline`, the proposal deadline of the first bribe in the list, is used for all these records, whereas the price is fetched for each bribe's proposal deadline.

```
    // Cache historical price when available
    if (!historicalRecord && proposalDeadline < currentTimestamp) {
      if (!historicalDeadline) {
        historicalDeadline = proposalDeadline; // @audit sets historicalDeadline to some
        ↪  proposalDeadline
      }

      // For legacy native ETH support, we use `coingecko:ethereum` as identifier
      historicalQueue.push({
        chainLabel: token === '0x0' ? 'coingecko' : chainLabel,
        token: token === '0x0' ? 'ethereum' : token,
        proposalDeadline,
      });
    }
  }
});

if (historicalQueue.length) {
  // Attempt to get and cache historical prices for available tokens
  try {
    const historicalQuery = historicalQueue
      .map( // @audit fetches price for proposalDeadline
        ({ chainLabel, token, proposalDeadline }) =>
          `%22${chainLabel}:${token}%22:%5B${proposalDeadline}%5D`
      )
      .join(',');
    ...
    await async.forEachOfSeries(
      historicalData.data.coins,
      async (data, coinId) => {
        ...

        await setCoinHistoricalRecord(
          chainLabel === 'coingecko' ? 'ethereum' : chainLabel,
          tokenAddress === 'ethereum' ? '0x0' : tokenAddress,
          historicalDeadline, // @audit uses historicalDeadline
          price
        );
      }
    );
  } catch (error) {
    console.log(error);
  }
}
```

Recommendation: It's unclear why the same `historicalDeadline` is used for all bribes instead of the bribe's proposal deadline. The records can have a price for a different time than what is indicated in the record.

Redacted: The records fetched for the specified protocol and deadline (or tx hash param) pairs will have a list of bribes deposited for the same deadline value, thus we can batch them together under the same API call to get historical prices.

Spearbit: Marked as acknowledged.

5.4.6 `try/catch` blocks for async-redis are never hit

Severity: Informational

Context: database/redis/coin.js#L37, all `client().function()` calls in `database/redis/*`, throughout the code base

Description: The redis client used is from the `async-redis` NPM package and all functions return a promise. Catching errors on promises is done by appending a `.catch()` to the promise or by `await`**ing** a `try/catch` block. The code uses `try/catch` blocks but does not await the promises inside the functions:

```
const setAddressToCoinRecord = async (chain, address, coin) => {
  try {
    return client().hset(
      addressToCoinHash,
      getAddressToCoinHashKey(chain, address),
      coin
    );
  } catch (error) { // @audit this is not caught if an error in `client().hset` happens
    throw error;
  }
};
```

Recommendation: As the `catch` block is just rethrowing the error, consider removing it. Alternatively, `await` the `client().hset` call.

Spearbit: Marked as acknowledged.