Design Overview

The malware URL lookup service design will need to take into consideration various technical challenges that are likely to be faced. Some of these are specifically outlined in the exercise requirements, while others can be ascertained by the nature of service being provided. Some of the areas where specific technical challenges may be encountered are captured below.

Technical Challenges

As always, identifying areas of specific technical challenge will be important in order to derisk a project, and keep moving the design of the system forward. Early prototyping and benchmarking should be employed in order to determine how any proposed technology choices will impact performance.

Request/Response Latency

One factor that is likely to have an outsized impact on request latency is the ability to maintain the flagged URL string set in memory. As this will be a necessary technical requirement in order to meet reasonable overall latency figures, the choices in how data is stored and accessed will become important in trying to find ways to reduce round trip times.
A couple of areas to be looked at are the following. The choice of an in-memory caching strategy that can deal with a large and growing data set without greatly increasing the lookup time. The use of a high performance message protocol and a maintained connection between proxies and services will likely provide additional latency gains.

Memory for Large URL Set

If in memory URL set sizes are utilizing significant portions of typical server memory capacities, it will be necessary to look at approaches to horizontally scale these data sets across physical servers.

Design Assumptions

In the interest of conserving space, any URL being passed in by the Proxy for malware cache lookup would consist of only the domain portions of the URL. It may even be acceptable to reduce all to the second level domain to reduce data sets, and speed cache lookups.

Authentication and authorization will need to be addressed, but will be considered outside the scope of this document.

Design Recommendations

The following recommendations should be kept in mind as the system is implemented, and bottlenecks and performance issues surface as the system scales out and grows.

Horizontally Scaling URL Data Sets

Splitting the URL dataset across multiple physical servers will allow more addressable physical memory, and may also provide an advantage in reducing lookup latencies. Even though there will be an additional layer of indirection, the benefit of drastically reducing the size of the data set where the lookup is performed could have large consequences in reducing the lookup portion of the request.
If no suitable 3rd party technology can be found, it would be possible to come up with an approach to have the front end API call handler of this service maintain a knowledge of where a particular URL would be located if it was. A simple strategy could use a non cryptographic fast hash of the URL to provide a seed for a modulo against the set of running services.   There could also be additional data redundancies to cover calling a secondary server in the case of primary outages or degradations.

Use of HTTP/2 based RPC

As the corporate owned Proxy server is the sole consumer of this service, it makes sense to look at using a fast HTTP/2 based service API with a maintained connection. Open source packages such as gRPC are well tested and can provide significant benefits over Web API calls.

- Binary messages provide efficient serialization
- HTTP/2 provides faster data transfer and reduced latency
- Maintained bidirectional connection reduces connection setup and overhead

Use of efficient URL cache

Even with a horizontal scaling strategy for the URL data sets, the size of the overall data will necessitate the use of very large in memory instances on any one server. Due to this it will be important to choose technology that predictably and gently degrades in lookup time even for large data sets.

One potential area of investigation is the use of trie structures that may be well suited for efficient lookup of strings in large data set situations. These tree data structures maintain O(n) lookup, insert, and delete operation performance with large data sets.

Bloom filters are also an area that should be investigates, although there are some behaviours that may make them unsuitable for the use in this situation. Specifically, they can be fast and memory efficient, but are susceptible to false positives.