Key Consistency for Oblivious HTTP by Double-Checking

Benjamin Schwartz, OHAI @ IETF 114

Example: Platform telemetry

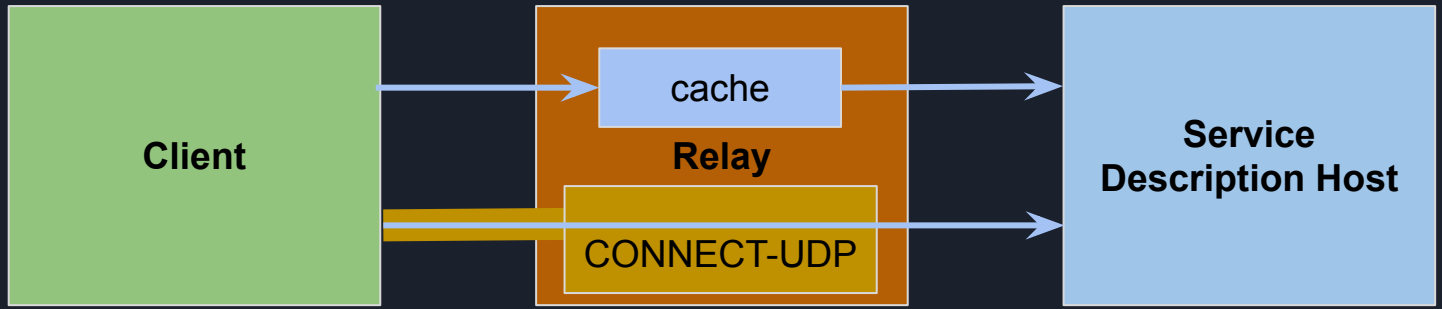
- My OS installation image came configured to report telemetry to a default telemetry service that supports OHTTP.
- I believe my OS image is the same as everyone else's, and I trust the code running locally, but **I want to prevent the telemetry service from linking my reports together**.
  - **Otherwise the OHTTP Relay is unnecessary!**
- I have configured my OS with an OHTTP Relay that I trust not to collude with the telemetry service, but **I don't trust the Relay with the contents of my telemetry reports**.
  - **Otherwise the OHTTP Gateway is unnecessary!**
- **Problem**: How do I ensure that the Gateway URL, KeyConfig, and Target URL are authentic and the same as everyone else's?

Easy answer: Hardcode everything!

- Bake the Gateway URL, Target URL, and KeyConfig right into the OS image
  - or in general distribute them through a trusted, consistent “bootstrap channel”.
- **Problem**: This prevents key rotation and other operational adjustments.

This proposal: **Bootstrap through the Relay**

- Store the Gateway URL, KeyConfig, and Target URL(s) in a config file at a fixed URL.
  - This URL's origin is the "Service Description Host"
- Fetch this config URL through the Relay, acting as a caching forward proxy
  - Guarantees consistency but not authenticity
- Fetch it again directly from the Service Description Host
  - Guarantees authenticity but not consistency
  - Do this using CONNECT-UDP through the Relay to conceal the client IP
- **Check that they match!**



Many details

- Stitched together from standard HTTP cache and proxy components
  - Headers used: Cache-Control, ETag, If-Match, Age
  - Some additional requirements beyond general HTTP compliance.
- Defenses against different attackers
  - Malicious Relays (KeyConfig forgery)
  - Malicious Service Description Hosts (cache wiping, fetch timing correlation)
  - Colluding malicious Clients (cache wiping, cache eviction)
- Performance considerations
  - Various recommended optimizations
  - Overall latency is generally 2 RTT through the proxy

Seeking eventual adoption in OHAI