Workgroup: ohai
Internet-Draft:
draft-schwartz-ohai-consistency-doublecheck-02
Published: 1 July 2022
Intended Status: Standards Track
Expires: 2 January 2023
Authors: B. M. Schwartz
Google LLC

Key Consistency for Oblivious HTTP by Double-Checking

Abstract

The assurances provided by Oblivious HTTP depend on the client's ability to verify that it is using the same Gateway, Target, and KeyConfig as many other users. This specification defines a protocol to enable this verification.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-schwartz-ohai-consistency-doublecheck/.

Source for this draft and an issue tracker can be found at https://github.com/bemasc/access-services.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 2 January 2023.

Table of Contents

1. Introduction

Oblivious HTTP [I-D.ietf-ohai-ohttp] identifies four parties to each exchange: the Client, Relay, Gateway and Target. When used properly, Oblivious HTTP enables the Client to send requests to the Target in such a way that the Target and Gateway cannot tell whether two requests came from the same Client, and the Relay cannot see the contents of the requests.

The Target and Gateway are tightly coupled, as the Gateway can see and modify all cleartext data to and from the Target. For ease of description, we will refer to the Target and Gateway collectively

(including the URI and KeyConfig for the Gateway and the URI of the Target) as the "Service".

Oblivious HTTP's threat model assumes that at least one of the Relay and the Service is acting properly, i.e. complying with the protocol and keeping certain information confidential. If either Relay or Service misbehaves, the only effect must be a denial of service.

In order for these security guarantees to hold, several preconditions must be met:

1. The Client must be one of many users who might be using the Relay. Otherwise, use of the Relay reveals the user's identity to the Gateway.

2. The Client must hold an authentic KeyConfig for the Gateway. Otherwise, the Client could be speaking to the Relay, impersonating the Gateway.

3. All users of the Relay must be equally likely to use this Service, regardless of their prior activity. Otherwise, the encrypted request identifies the Client to the Service.

4. (optional) The Gateway must not learn the IP addresses of the Clients, collectively. Otherwise, the Gateway might be able to deanonymize requests by correlating them with external information about the Clients.

This specification defines behaviors for the Client, Relay, and Service that achieve preconditions 2-4. (This specification does not address precondition 1.)

This specification assumes that the Service is identified by an Access Description [I-D.schwartz-masque-access-descriptions], which we call the "Service Description". For this specification to meet its goals, the Service Description's URL must have been distributed to clients in a globally consistent fashion. For example, the Service Description URL might be the default value of a software setting, or it might be published on a third party's website. This specification allows clients to convert the static, long-lived Service Description URL into a fresh Service Description without losing the privacy guarantees of Oblivious HTTP.

In principle, Services could achieve a similar effect by distributing their Service Descriptions directly through this globally consistent channel. However, these ad hoc pubication channels may not be fast enough to support frequent updates (e.g., key rotations), especially if updates require user intervention.

This draft combines elements of the "Direct Discovery", "Single Proxy Discovery", and "Independent Verification" strategies defined in [I-D.wood-key-consistency].

2. Conventions and Definitions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Overview

In the Key Consistency Double-Check procedure, the Client emits two HTTP GET requests: one to the Relay, and one through the Relay to the Service Description Host using CONNECT-UDP. (The Service Description Host, Gateway, and Target are most commonly expected to be a single origin.) The Relay will forward the first request to the Service Description Host if the response is not in cache.

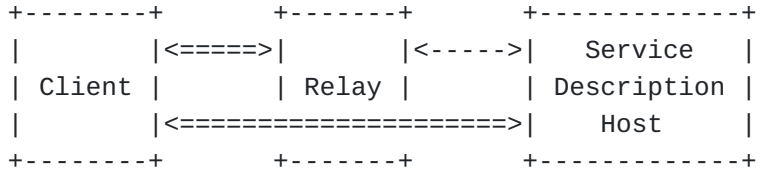

Figure 1: Overview of Key-Consistency Double-Check

The Relay caches the response, ensuring that all clients share it during its freshness lifetime. The client checks this against the authenticated response from the Service Description Host, preventing forgeries.

4. Requirements

4.1. Oblivious Service

The Oblivious Service **MUST** publish an Access Description [I-D.schwartz-masque-access-descriptions] containing the "ohttp.gateway" key, e.g.:

```
{
  "ohttp": {
    "gateway": {
      "uri": "https://example.com/ohttp/",
      "key": "(KeyConfig in Base64)"
    }
  }
}
```

This Access Description is called the Service Description, and its origin is called the Service Description Host. This origin **MUST** support HTTP/3 [RFC9114], so that it can be accessed via the proxy's CONNECT-UDP service (see Section 4.2).

The Service Description Host **MUST** include a "strong validator" ETag (Section 2 of [RFC7232]) in any response to a GET request for this Service Description, and **MUST** support the "If-Match" HTTP request header (Section 3 of [RFC7232]). The response **MUST** indicate "Cache-Control: public, no-transform, s-maxage=(...), immutable" [RFC9111] [RFC8246]. For efficiency reasons, the max age **SHOULD** be at least 60 seconds, and preferably much longer.

If the Service Description changes, and the resource receives a request whose "If-Match" header identifies a previously served version that has not yet expired, it **MUST** return a success response containing the previous version. This response **MAY** indicate "Cache-Control: private".

4.2. Oblivious Relay

The Oblivious Relay **MUST** also provide CONNECT-UDP service [I-D.ietf-masque-connect-udp], and **SHOULD** also offer DNS over HTTPS [RFC8484], to enable the use of HTTPS records [SVCB] with CONNECT-UDP. This corresponds to an Access Description that includes the "ohttp.relay", "udp", and "dns" keys:

```
{
  "dns": {
    "template": "https://doh.example.com/dns-query{?dns}",
  },
  "udp": {
    "template":
        "https://proxy.example.org/masque{?target_host,target_port}"
  },
  "ohttp": {
    "relay": {
      "template": "https://relay.example.org/ohttp{?request_uri}"
    }
  }
}
```

Figure 2: Example Relay Access Service Description

The Oblivious Relay **MUST** allow use of the GET method to retrieve small JSON responses, and **SHOULD** make ample cache space available in order to avoid eviction of Service Descriptions. The Relay **SHOULD** share cache state among all clients, to ensure that they use the same Service Descriptions for each Oblivious Service. If the cache must be partitioned for architectural or performance reasons, operators **SHOULD** keep the number of users in each partition as large as possible.

Oblivious Relays **MUST** preserve the ETag response header on cached responses, and **MUST** add an Age header ([RFC9111], Section 5.1) to all proxied responses. Oblivious Relays **MUST** respect the "Cache-Control: immutable" directive, and **MUST NOT** revalidate fresh immutable cache entries in response to any incoming requests. (Note that this is different from the general recommendation in Section 2.1 of [RFC8246]). Oblivious Relays also **MUST NOT** accept PUSH_PROMISE frames from the target.

Relays **SHOULD** employ defenses against malicious attempts to fill the cache. Some possible defenses include:

- Rate-limiting each client's use of GET requests.

- Prioritizing preservation of cache entries that have been served to many clients, if eviction is required.

Oblivious Relays that are not intended for general-purpose proxy usage **MAY** impose strict transfer limits or rate limits on HTTP CONNECT and CONNECT-UDP usage.

If the Relay offers a DNS over HTTPS resolver, it **MUST NOT** enable EDNS Client Subnet support [RFC7871].

4.3. Client

The Client is assumed to know an "https" URI of the relevant Service Description. Before using that Service Description, it **MUST** perform the following "double-check" procedure:

1. Send a GET request to the Oblivious Relay's template (ohttp.proxy.template) with request_uri set to the Service Description URI.

2. Record the response (A).

3. Check that response A's "Cache-Control" values indicates "public" and "immutable".

4. Establish a CONNECT-UDP tunnel through the proxy to the Service Description URI's origin.

5. Fetch the Service Description URI through this tunnel, using a GET request with "If-Match" set to response A's ETag.

6. Record the response (B).

7. Check that responses A and B were successful and the contents are identical, otherwise fail.

This procedure ensures that the Service Description is authentic and will be shared by all users of this proxy. Once response A or B expires, the client **MUST** refresh it before continuing to use this Service Description, and **MUST** repeat the "double-check" process if either response changes.

Clients **MUST** perform each fetch to the origin (step 4) as a fully isolated request. Any state related to this origin (e.g. cached DNS records, CONNECT-UDP tunnels, QUIC transport state, TLS session tickets, HTTP cookies) **MUST NOT** be shared with prior or subsequent requests.

5. Example: Oblivious DoH

In this example, the client has been configured with an Oblivious DoH server and an Oblivious Relay. The Oblivious DoH server is identified by a Service Description at "https://doh.example.com/config.json" with the following contents:

```
{
  "dns": {
    "template": "https://doh.example.com/dns-query{?dns}",
  },
  "ohttp": {
    "gateway": {
      "uri": "https://example.com/ohttp/",
      "key": "(KeyConfig in Base64)"
    }
  }
}
```

The Oblivious Relay is identified as "proxy.example.org", which implies an Access Description at "https://proxy.example.org/.well-known/access-services". This resource's contents are:

```
{
  "dns": {
    "template": "https://proxy.example.org/dns-query{?dns}",
  },
  "udp": {
    "template":
        "https://proxy.example.org/masque{?target_host,target_port}"
  },
  "ohttp": {
    "relay": {
      "template": "https://relay.example.org/ohttp{?request_uri}"
    }
  }
}
```

The following exchanges then occur between the client and the proxy:

```
HEADERS
:method = GET
:scheme = https
:authority = relay.example.org
:path = /ohttp?request_uri=https%3A%2F%2Fdoh.example.com%2Fconfig.json
accept: application/access-services+json
```

```
                      HEADERS
                      :status = 200
                      cache-control: public, immutable, \
                          no-transform, s-maxage=86400
                      age: 80000
                      etag: ABCD1234
                      content-type: application/access-services+json
                      [Service Description contents here]
```

```
HEADERS
:method = CONNECT
:protocol = connect-udp
:scheme = https
:authority = proxy.example.org
:path = /masque?target_host=doh.example.com,target_port=443
capsule-protocol = ?1
```

```
                            HEADERS
                            :status = 200
                            capsule-protocol = ?1
```

   The client now has a CONNECT-UDP tunnel to doh.example.com, over
   which it performs the following GET request using HTTP/3:

```
HEADERS
:method = GET
:scheme = https
:authority = doh.example.com
:path = /config.json
if-match = ABCD1234
```

```
                      HEADERS
                      :status = 200
                      cache-control: public, immutable, \
                          no-transform, s-maxage=86400
                      etag: ABCD1234
                      content-type: application/access-services+json
                      [Service Description contents here]
```

   Having successfully fetched the Service Description from both
   locations, the client confirms that:

     *The responses are identical.

*The cache-control response from the proxy contained the "public" and "immutable" directives.

The client can now use the KeyConfig in this Service Description to reach the Oblivious DoH server, by converting DNS-over-HTTPS requests into Binary HTTP requests for "https://doh.example.com/dns-query", encrypting them to ohttp.gateway.key, and POSTing the encrypted requests to "https://relay.example.org/ohttp?request_uri=https%3A%2F%2Fexample.com%2Fohttp%2F".

6. Performance Implications

6.1. Latency

Suppose that the Client-Relay Round-Trip Time (RTT) is A, and the Relay-Service Description Host RTT is B. Suppose additionally that the Client has a persistent connection to the Relay that is already running. Then the procedure described in Section 4.3 requires:

*A for the GET request to the Relay

-+B if the requested Service Description is not in cache

-+B if the Relay does not have a TLS session ticket for the Service Description Host

*A for the CONNECT-UDP request to the Relay

*A + B for the QUIC handshake to the Service Description Host

*A + B for the GET request to the Service Description Host

This is a total of 4A + 4B in the worst case. However, clients can reduce the latency by issuing the requests to the Relay in parallel, and by using CONNECT-UDP's "false start" support. The Service Description Host can also optimize performance, by issuing long-lived TLS session tickets. With these optimizations, the expected total time is 2A + 2B.

This procedure only needs to be repeated if the Service Description has expired. To enable regular key rotation and operational adjustments, a cache lifetime of 24 hours may be suitable. Clients **MAY** perform this procedure in advance of an expiration to avoid a delay.

6.2. Thundering Herds

All clients of the same Relay and Service will have locally cached Service Descriptions with the same expiration time. When this entry expires, all active clients will send refresh GET requests to the

proxy at their next request. Relays **SHOULD** use "request coalescing" to avoid duplicate cache-refresh requests to the target.

If the Service Description has changed, these clients will initiate GET requests through the Relay to the Service Description Host to double-check the new contents. Relays and Service Description Hosts **MAY** use an HTTP 503 response with a "Retry-After" header to manage load spikes.

7. Security Considerations

7.1. In scope

7.1.1. Forgery

A malicious Relay could attempt to learn the contents of the Oblivious HTTP request by forging a Service Description containing its own KeyConfig. This is prevented by the client's requirement that the KeyConfig be served to it by the Service Description Host over HTTPS (Section 4.3).

7.1.2. Deanonymization

A malicious Service could attempt to link multiple Oblivious HTTP requests together by issuing each Client a unique, persistent KeyConfig. This attack is prevented by the Client's requirement that the KeyConfig be fresh according to the Relay's cache (Section 4.3).

A malicious Service could attempt to rotate its entry in the Relay's cache in several ways:

* Using HTTP PUSH_PROMISE frames. This attack is prevented by disabling PUSH_PROMISE at the Relay (Section 4.2).

* By also acting as a Client and sending requests designed to replace the Service Description in the cache before it expires:

  - By sending GET requests with a "Cache-Control: no-cache" or similar directive. This is prevented by the response's "Cache-Control: public, immutable" directives, which are verified by the Client (Section 4.3), and by the Relay's obligation to to respect these directives strictly (Section 4.2).

  - By filling the cache with new entries, causing its previous Service Description to be evicted. Section 4.2 describes some possible mitigations.

A malicious Service could attempt to link different requests for the Service Description, in order to link the Oblivious HTTP requests that follow shortly after. This is prevented by fully isolating each

request (Section 4.3), and by disabling EDNS Client Subnet (Section 4.2).

7.1.3. Abusive traffic

A malicious client could use the proxy to send abusive traffic to any destination on the internet. Abuse concerns can be mitigated by imposing a rate limit at the proxy (Section 4.2).

7.2. Out of scope

This specification assumes that the client starts with identities of the Relay and Service that are authentic and widely shared. If these identities are inauthentic, or are unique to the client, then the security goals of this specification are not achieved.

This specification assumes that at most a small fraction of Clients are acting on behalf of a malicious Service. If a large fraction of the Clients are malicious, they could conspire to flood the Relay's cache with entries that seem popular, leading to rapid eviction of the malicious Service's Service Descriptions. Similar concerns apply if a malicious Service can compel naive Clients to fetch a very large number of Service Descriptions.

A Client's requests for the Service Description may become linkable if they have distinctive QUIC Initials, HTTP/3 Settings, RTT, or other protocol features observable through the Relay. This specification does not offer specific mitigations for protocol fingerprinting.

8. IANA Considerations

No IANA action is requested.

9. References

9.1. Normative References

[I-D.ietf-masque-connect-udp]
Schinazi, D., "Proxying UDP in HTTP", Work in Progress, Internet-Draft, draft-ietf-masque-connect-udp-15, 17 June 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-masque-connect-udp-15>.

[I-D.ietf-ohai-ohttp] Thomson, M. and C. A. Wood, "Oblivious HTTP", Work in Progress, Internet-Draft, draft-ietf-ohai-

ohttp-01, 15 February 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-ohai-ohttp-01>.

[I-D.schwartz-masque-access-descriptions] Schwartz, B. M., "HTTP Access Service Description Objects", Work in Progress, Internet-Draft, draft-schwartz-masque-access-descriptions-01, 28 June 2022, <https://datatracker.ietf.org/doc/html/draft-schwartz-masque-access-descriptions-01>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <https://www.rfc-editor.org/rfc/rfc7232>.

[RFC7871] Contavalli, C., van der Gaast, W., Lawrence, D., and W. Kumari, "Client Subnet in DNS Queries", RFC 7871, DOI 10.17487/RFC7871, May 2016, <https://www.rfc-editor.org/rfc/rfc7871>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8246] McManus, P., "HTTP Immutable Responses", RFC 8246, DOI 10.17487/RFC8246, September 2017, <https://www.rfc-editor.org/rfc/rfc8246>.

[RFC8484] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484, DOI 10.17487/RFC8484, October 2018, <https://www.rfc-editor.org/rfc/rfc8484>.

[RFC9111] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/rfc/rfc9111>.

[RFC9114] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/rfc/rfc9114>.

9.2. Informative References

[I-D.wood-key-consistency] Davidson, A., Finkel, M., Thomson, M., and C. A. Wood, "Key Consistency and Discovery", Work in

Progress, Internet-Draft, draft-wood-key-consistency-02, 4 March 2022, <https://datatracker.ietf.org/doc/html/draft-wood-key-consistency-02>.

[SVCB] Schwartz, B., Bishop, M., and E. Nygren, "Service binding and parameter specification via the DNS (DNS SVCB and HTTPS RRs)", Work in Progress, Internet-Draft, draft-ietf-dnsop-svcb-https-10, 24 May 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-dnsop-svcb-https-10>.

Acknowledgments

TODO acknowledge.

Author's Address

Benjamin M. Schwartz
Google LLC

Email: bemasc@google.com