Workgroup: ohai
Internet-Draft:
draft-schwartz-ohai-consistency-doublecheck-03
Published: 19 October 2022
Intended Status: Standards Track
Expires: 22 April 2023
Authors: B. M. Schwartz
Google LLC

Key Consistency by Double-Checking via a Semi-Trusted Proxy

Abstract

Several recent IETF privacy protocols require clients to acquire bootstrap information for a service in a way that guarantees both authenticity and consistency, e.g., encrypting to the same key as many other users. This specification defines a procedure for transferring arbitrary HTTP resources in a manner that provides these guarantees. The procedure relies on access to a semi-trusted HTTP proxy, under the same security assumptions as an Oblivious HTTP Relay.

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

Copyright Notice

Table of Contents

1. Introduction

Oblivious HTTP [I-D.ietf-ohai-ohttp] enables an HTTP client to send requests to a target in such a way that no single party can determine both the identity of the client and the contents of its

request. Oblivious HTTP identifies four parties to each exchange: the Client, Relay, Gateway and Target.

In order for Oblivious HTTP's privacy 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 Gateway, KeyConfig, and Target, regardless of their prior activity. Otherwise, the encrypted request identifies the Client to the Gateway.

4. (optional) The Gateway should 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.

The Privacy Pass protocol [I-D.ietf-privacypass-protocol] allows a Client to retrieve tokens from an Issuer, and present them to an Origin, in such a way that the Origin can verify the validity of the tokens but cannot identify the Client, even if it colludes with the Issuer. Privacy Pass requires a similar set of preconditions for its privacy guarantees:

1. The Client's transport to the Origin must not reveal the client's identity (e.g. via the client IP address).

2. The Client must hold an authentic Issuer Directory Object. Otherwise, issuance will fail (resulting in a denial of service, not a privacy violation).

3. All Clients with the same transport metadata must be using the same Issuer Directory Object for this Issuer. Otherwise, these factors together uniquely identify the client.

4. (optional) The Origin should not learn the IP addresses of the Clients collectively, as above.

Pre-requisite 1 on this list can be achieved by employing a shared transport proxy, on the assumption that at least one of the proxy or the Origin is non-malicious. (This is the same assumption that an Oblivious HTTP Client makes about its Relay).

This specification assumes that a shared, "semi-trusted" proxy is available (fulfilling precondition 1 on each list). It defines a three-party protocol for a Client, Proxy, and Origin to fetch an HTTP resource in a manner that ensures authenticity and consistency among Clients of a single Proxy. When used to initialize Oblivious HTTP or Privacy Pass, it guarantees preconditions 2, 3, and optionally 4.

The input to this protocol is a Desired Resource URI: an "https" URI on the Origin that is assumed to have been distributed to Clients in a globally consistent fashion. For example, this URI 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 Desired Resource URI into a fresh copy of that resource, i.e. to fetch the resource.

In principle, the Desired Resource could have been distributed directly through the assumed globally consistent channel. However, these ad hoc publication channels may not be fast enough to support frequent updates (e.g., key rotations), especially if updates require user intervention.

This draft is an instantiation of the Shared Proxy with Key Confirmation strategy defined in Section 4.3 of [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 for the Desired Resource. One uses the Proxy as an HTTP request proxy (terminating TLS and HTTP), and the other optionally uses it as a transport proxy (using TLS end-to-end with the Origin). The Proxy will forward the first request to the Origin only if the response is not in cache.

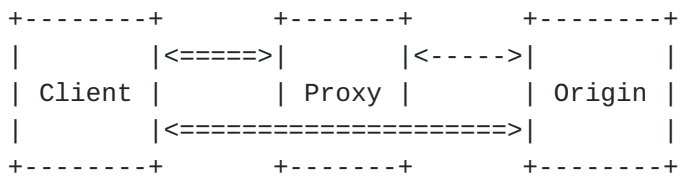

Figure 1: Overview of Key-Consistency Double-Check

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

4. Requirements

4.1. Origin

The Desired Resource **MUST** include a "strong validator" ETag (Section 2 of [RFC7232]) in any response to a GET request, 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 Desired Resource changes, and the Origin receives a request for the resource 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. Proxy

The Proxy **MUST** offer HTTP request proxy capability, and **SHOULD** also offer TCP proxy, UDP proxy, and encrypted DNS resolution capabilities. (An associated encrypted DNS resolver enables reliable use of HTTPS records [SVCB], improves metadata confidentiality, and allows EDNS Client Subnet to be disabled reliably.) For example, a proxy in the recommended configuration could be described by the following Access Service Description [I-D.schwartz-masque-access-descriptions]:

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

Figure 2: Example Proxy Access Service Description

The Proxy **MUST** allow use of the GET method to proxy small responses, and **SHOULD** make ample cache space available in order to avoid eviction of Desired Resources. The proxy **SHOULD** share cache state among all clients, to ensure that they observe the same resource contents. 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.

Proxies **MUST** preserve the ETag response header on cached responses, and **MUST** add an Age header ([RFC9111], Section 5.1) to all proxied responses. Proxies **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]). Proxies also **MUST NOT** accept PUSH_PROMISE frames from the target.

Proxies **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.

Proxies that are not intended for general-purpose use **MAY** impose strict transfer limits or rate limits on transport proxy usage.

If the Proxy offers an Encrypted DNS service, it **MUST NOT** enable EDNS Client Subnet support [RFC7871].

4.3. Client

The Client is assumed to know the "https" URI of the Desired Resource. To retrieve that resource, it **MUST** perform the following "double-check" procedure:

1. Send a GET request for the Desired Resource URL via the Proxy's HTTP request proxy function.

2. Record the response (A).

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

4. Establish a transport tunnel through the Proxy to the Origin (**OPTIONAL**).

5. Fetch the Desired Resource (through this tunnel if available), 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 retrieved copy of the Desired Resource is authentic and is identical to the one held by any other users of this proxy. Once response A or B expires, the client **MUST** refresh it before continuing to use this resource, 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.

4.4. Negative responses

If the Desired Resource does not exist, the Double Check requirements apply without modification. The Cache-Control headers ensure that the negative response (e.g., HTTP 404) is cacheable regardless of its status code. If Double Check succeeds with a negative response, the Client can be confident that any other Clients of this proxy are holding the same negative response.

5. Example: Oblivious DoH

In this example, the client has been configured with a Proxy via the following Access Service Description:

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

The client has recently been instructed to use an Encrypted DNS server identified as "doh.example.com" that might support Oblivious DoH. To discover and access this Oblivious DoH service, the client attempts to retrieve two Desired Resources on the Origin "https://doh.example.com":

- The DNS Server's Access Service Description, at "/.well-known/access-services" (Section 5 of [I-D.schwartz-masque-access-descriptions]).
  - This conveys the DoH URI template associated with this origin.
- The Oblivious Gateway KeyConfig, at "/.well-known/oblivious-gateway" (Section 5 of [I-D.pauly-ohai-svcb-config]).
  - This conveys the public keys to use for Oblivious HTTP.

To prevent client-targeting attacks, the client retrieves both of these resources via DoubleCheck. The HTTP requests for the Access Service Description appears as follows:

```
HEADERS
:method = GET
:scheme = https
:authority = relay.example.org
:path = /http?target_uri=
    https%3A%2F%2Fdoh.example.com%2F.well-known%2Faccess-services
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
                      {
                        "dns": {
                          "template":
                              "https://doh.example.com/foo{?dns}"
                        }
                      }
```

```
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 = /.well-known/access-services
accept: application/access-services+json
if-match = ABCD1234
```

```
                    HEADERS
                    :status = 200
                    cache-control: public, immutable, \
                        no-transform, s-maxage=86400
                    etag: ABCD1234
                    content-type: application/access-services+json
                    {
                      "dns": {
                        "template":
                            "https://doh.example.com/foo{?dns}"
                      }
                    }
```

Having successfully fetched the DoH 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.

This concludes the DoubleCheck procedure. The Oblivious Gateway KeyConfig is retrieved by a similar procedure. These two DoubleCheck procedures can run in parallel, and **MAY** share a single transport proxy tunnel for efficiency.

Once the client has double-checked the DoH Service Description and the KeyConfig, it can use the Oblivious DoH service by forming DNS-over-HTTPS requests for "https://doh.example.com/foo{?dns}" as Binary HTTP requests, encrypting them to the KeyConfig, and POSTing the ciphertext to "https://relay.example.org/http?target_uri=https%3A%2F%2Fdoh.example.com%2F.well-known%2Foblivious-gateway".

6. Performance Implications

6.1. Latency

Suppose that the Client-Proxy Round-Trip Time (RTT) is A, and the Proxy-Origin RTT is B. Suppose additionally that the Client has a persistent connection to the Proxy that is already running. Then the

procedure described in Section 4.3, with a CONNECT-UDP transport proxy, requires:

*A for the GET request to the Proxy

-+B if the Desired Resource is not in cache

-+B if the Proxy does not have a TLS session ticket for the Origin

*A for the CONNECT-UDP request to the Proxy

*A + B for a QUIC handshake to the Origin

*A + B for the GET request to the Origin

This is a total of 4A + 4B in the worst case. However, clients can reduce the latency by issuing the requests to the Proxy in parallel, and by using CONNECT-UDP's "false start" support. The Origin 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 Desired Resource 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 Proxy and Desired Resource will have locally cached copies with the same expiration time. When this copy expires, all active clients will send refresh GET requests to the Proxy at their next request. Proxies **SHOULD** use "request coalescing" to avoid duplicate cache-refresh requests to the Origin.

If the Desired Resource has changed, these clients will all initiate GET requests to the Origin (via transport proxy if applicable) to double-check the new contents. Proxies and Origins **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. Authenticity

A malicious Proxy could attempt to forge the Desired Resource by replacing the response body (e.g., returning an Oblivious HTTP KeyConfig containing a Public Key controlled by the Relay). This is

prevented by the Client's requirement that the Desired Resource be served to it by the Origin over HTTPS (Section 4.3).

7.1.2. Consistency

A malicious Origin could attempt to break the consistency guarantee by issuing each Client a unique, persistent variant of the Desired Resource. This attack is prevented by the Client's requirement that the resource be fresh according to the Proxy's cache (Section 4.3).

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

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

* By also acting as a Client and sending requests designed to replace the Desired Resource 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 Proxy's obligation to respect these directives strictly (Section 4.2).

  - By filling the cache with new entries, causing the cached copy of the resource to be evicted. Section 4.2 describes some possible mitigations.

7.1.3. Temporal Correlation Attacks

A malicious Origin could attempt to identify or link different requests for the Desired Resource, in order to link proxied requests that follow shortly after. If a transport proxy is used (as recommended), this is prevented by fully isolating each request (Section 4.3), and by disabling EDNS Client Subnet (Section 4.2).

7.1.4. 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 Proxy and the Desired Resource 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 Origin. If a large fraction of the Clients are malicious, they could conspire to flood the Proxy's cache with entries that seem popular, leading to rapid eviction of any cached resources. Similar concerns apply if a malicious Origin can compel naive Clients to fetch a very large number of distinct resources through the Proxy.

Even when a transport proxy is used, the Client's requests for the Desired Resource may become linkable if they have distinctive TLS ClientHellos, QUIC Initials, HTTP/3 Settings, RTT, or other protocol features observable through the transport proxy. This specification does not offer specific mitigations for protocol fingerprinting.

8. IANA Considerations

No IANA action is requested.

9. References

9.1. Normative References

[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>.

[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>.

9.2. Informative References

[I-D.ietf-ohai-ohttp] Thomson, M. and C. A. Wood, "Oblivious HTTP", Work in Progress, Internet-Draft, draft-ietf-ohai-ohttp-05, 26 September 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-ohai-ohttp-05>.

[I-D.ietf-privacypass-protocol] Celi, S., Davidson, A., Faz-Hernández, A., Valdez, S., and C. A. Wood, "Privacy Pass Issuance Protocol", Work in Progress, Internet-Draft, draft-ietf-privacypass-protocol-06, 6 July 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-privacypass-protocol-06>.

[I-D.pauly-ohai-svcb-config] Pauly, T. and T. Reddy.K, "Discovery of Oblivious Services via Service Binding Records", Work in Progress, Internet-Draft, draft-pauly-ohai-svcb-config-03, 27 July 2022, <https://datatracker.ietf.org/doc/html/draft-pauly-ohai-svcb-config-03>.

[I-D.schwartz-masque-access-descriptions] Schwartz, B. M., "HTTP Access Service Description Objects", Work in Progress, Internet-Draft, draft-schwartz-masque-access-descriptions-03, 18 October 2022, <https://datatracker.ietf.org/doc/html/draft-schwartz-masque-access-descriptions-03>.

[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-03, 17 August 2022, <https://datatracker.ietf.org/doc/html/draft-wood-key-consistency-03>.

[SVCB] Schwartz, B. M., 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-11, 11 October 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-dnsop-svcb-https-11>.

Acknowledgments

TODO acknowledge.

Author's Address

Benjamin M. Schwartz
Google LLC

Email: bemasc@google.com