Workgroup: Privacy Pass
Internet-Draft:
draft-ietf-privacypass-consistency-mirror-00
Published: 30 January 2024
Intended Status: Standards Track
Expires: 2 August 2024
Authors: B. Beurdouche (Mozilla)
M. Finkel (Apple Inc.)
S. Valdez (Google LLC)
C. A. Wood (Cloudflare)
T. Pauly (Apple Inc.)

Checking Resource Consistency with HTTP Mirrors

Abstract

This document describes the mirror protocol, an HTTP-based protocol for fetching mirrored HTTP resources. The primary use case for the mirror protocol is to support HTTP resource consistency checks in protocols that require clients have a consistent view of some protocol-specific resource (typically, a public key) for security or privacy reasons, including Privacy Pass and Oblivious HTTP. To that end, this document also describes how to use the mirror protocol to implement these consistency checks.

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-ietf-privacypass-consistency-mirror/.

Discussion of this document takes place on the Privacy Pass Working Group mailing list (mailto:privacy-pass@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/privacy-pass/. Subscribe at https://www.ietf.org/mailman/listinfo/privacy-pass/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-privacypass/draft-ietf-privacypass-consistency-mirror.

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 August 2024.

Table of Contents

1. Introduction

Privacy-enhancing protocols such as Privacy Pass [PRIVACYPASS] and Oblivious HTTP [OHTTP] require clients to obtain and use a public key for execution. In Privacy Pass, public keys are used by clients when issuing and redeeming tokens for anonymous authorization. In

Oblivious HTTP (OHTTP), clients use public keys to encrypt messages to a gateway server.

Deployments of protocols such as Privacy Pass and OHTTP requires that very large sets of clients share the same key, or even that all clients globally share the same key. This is because the privacy properties depend on the client anonymity set size. In other words, the key that's used determines the set to which a particular client belongs. Using a unique, client-specific key would yield an anonymity set of size one, therefore violating the desired privacy goals of the system. Clients that use the same key as one another are said to have a consistent view of the key.

[CONSISTENCY] describes this notion of consistency in more detail. It also outlines several designs that can be used as the basis for consistency systems. This document is a concrete instantiation of one of those designs, "Shared Cache Discovery". In particular, this document describes the mirror protocol, which is a protocol for fetching a cached copy of an HTTP resource from so-called mirrors. In this context, a mirror is an HTTP resource that fetches and caches copies of other HTTP resources that can be returned to clients. In turn, clients can then use these cached resource copies for consistency checks, i.e., to compare their expected representation of the resource against that which the mirrors provide.

The mirror protocol can be run one or more times by an application to achieve the desired consistency criteria. For example, the mirror protocol can be run once with a trusted mirror, or more than once with many, potentially less trusted mirrors, and used for determining consistency. Section 5 provides general guidance for using the mirror protocol for consistency checks, along with specific profiles of the protocol for Privacy Pass and OHTTP in Section 5.4 and Section 5.5, respectively.

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. Terminology

The following terms are used throughout this document:

*Resource: A HTTP resource identified by a URL.

*Normalized resource representation: A unique or otherwise protocol-specific representation that is derived from an HTTP

resource. The process of normalization is specific to a protocol and the resource in question.

*Mirror: A HTTP resource that fetches and caches HTTP resources.

4. Mirror Protocol

The mirror protocol is a simple HTTP-based protocol similar to a reverse proxy. Each mirror resource, henceforth referred to as a mirror, is identified by a Mirror URI Template [RFC6570]. The scheme for the Mirror URI Template **MUST** be "https". The Mirror URI Template uses the Level 3 encoding defined Section 1.2 of [RFC6570] and contains one variables: "target", which is the percent-encoded URL of a HTTP resource to be mirrored. Example Mirror URI Templates are shown below.

```
https://mirror.example/mirror{?target}
https://mirror.example/{target}
```

The Mirror URI Template **MUST** contain the "target" variable exactly once. The variable **MUST** be within the path or query components of the URI.

In addition, each mirror is configured with a MIN_VALIDITY_WINDOW parameter, which is an integer indicating the minimum time for resources the mirror will cache according to their "max-age" response directive. We refer to the validity window of the mirror response as the period of time determined by the Cache-Control headers as the response.

Clients send requests to mirror resources after being configured with their corresponding Mirror URI Template. Clients **MUST** ignore configurations that do not conform to this template.

Upon receipt of a mirror request, mirrors validate the incoming request. If the request is invalid or malformed, e.g., the "target" parameter is not a correctly encoded URL, the mirror aborts and returns a a 4xx (Client Error) to the client. The mirror **SHOULD** check that the target resource identified by the "target" parameter is allowed by policy, e.g., so that it is not abused to fetch arbitrary resources. One way to implement this check is via an allowlist of target URLs.

If the request is valid and allowed, the mirror checks to see if it has a cached version of the resource identified by the target URL. Mirrors can provide a cached response to a client request if the following criteria are met:

1. The target URL matches that of a cached response.

2. The cached response is fresh according to its Cache-Control header (see Section 4.2 of [CACHING]).

If both criteria are met, the mirror encodes the cached response using Binary HTTP [BHTTP] and returns it to the client in a response. The mirror response includes a Cache-Control header with "max-age" directive set to that of the cached response.

Otherwise, mirrors send a GET request to the target resource URL, copying the Accept header from the client request if present. If this request fails, the mirror returns a 4xx error to the client. If this request suceeeds, the mirror checks it for validity. The response is considered valid and stored in the mirror's cache if the following criteria are met:

1. The response can be cached according to the rules in Section 3 of [CACHING]. In particular, if the request had a Vary header, this is used in determining whether the mirror's response is valid.

2. The Cache-Control header is present, has a "max-age" response directive that is greater than or equal to MIN_VALIDITY_WINDOW, and does not have a "no-store" or "private" directive.

Mirrors purge this cache when the response is no longer valid according to the Cache-Control headers.

To complete the client request, the mirror then encodes the response using Binary HTTP [BHTTP] and returns it to the client in a response. The mirror response incldues a Cache-Control header with "max-age" directive set to that of the cached response.

Clients recover the target's mirrored response by Binary HTTP decoding the mirror response content.

4.1. Mirror Request and Response Example

The following example shows two mirror request and response examples. The first one yields a mirror cache miss and the second one yields a mirror cache hit. The Mirror URI Template is "https://mirror.example/mirror{?target}", and the target URL is "https://issuer.example/.well-known/private-token-issuer-directory".

The first client request to the mirror might be the following.

```
:method = GET
:scheme = https
:authority = mirror.example
:path = /mirror?target=https%3A%2F%2Fissuer.example%2F.well-known%2Fpriv
accept = application/private-token-issuer-directory
```

Upon receipt, the mirror decodes the "target" parameter, inspects its cache for a copy of the resource, and then constructs a HTTP request to the target URL to fetch the content. If present, the relay copies the Accept header from the client request to the request sent to the target. This mirror request to the target might be the following.

```
:method = GET
:scheme = https
:authority = target.example
:path = /.well-known/private-token-issuer-directory
accept = application/private-token-issuer-directory
```

The target response is then returned to the mirror, like so:

```
:status = 200
content-type = application/private-token-issuer-directory
content-length = ...
cache-control: max-age=3600

<Bytes containing a private token issuer directory>
```

The mirror caches this response content for the target URL, encodes it using Binary HTTP [BHTTP], and then returns the response to the client:

```
:status = 200
content-length = ...
cache-control: max-age=3600

<Bytes containing the target's BHTTP-encoded response>
```

When a second client asks for the same request by the mirror it can be served with the cached copy. The second client's request might be the following:

```
:method = GET
:scheme = https
:authority = mirror.example
:path = /mirror?target=https%3A%2F%2Fissuer.example%2F.well-known%2Fpriv
```

The mirror validates the request, locates the cached copy of the "https://issuer.example/.well-known/private-token-issuer-directory" content, and then returns it to the client without updating its cached copy.

```
:status = 200
content-length = ...
cache-control: max-age=3600

<Bytes containing the target's BHTTP-encoded response>
```

5. Using Mirrors for Consistency Checks

Clients can use mirrors to implement consistency checks for a candidate HTTP resource. In particular, in possession of the target URL at which the resource was obtained, as well as an authoritative representation of the resource, clients can check to see if this resource is consistent with that of the mirror's as follows:

1. Send a mirror request to the mirror for the target URL. If the request fails, fail this consistency check.

2. Otherwise, compute the first valid representation of the resource based on the mirror's response.

3. Compare the computed representation to the input resource representation. If they do not match, fail this consistency check. Otherwise, this consistency check succeeds.

The benefits of using the mirror protocol to check consistency depend on a multitude of factors, including, but not limited to, the number of clients interacting with a particular mirror, whether or not the mirror is trustworthy, and application requirements for dealing with consistency check failures.

5.1. Handling Consistency Failures

If a consistency check fails because the mirrored resource did not match, the client **MUST NOT** use the original resource. For cases where the check failed because the client was unable to communicate with the mirror, client policy dictates whether or not to assume the resource is consistent. Client behavior for what to do in the case of inconsistency can vary depending on the protocol, availability of alternative services, and client policy.

If the client has multiple options for equivalent services, it can choose to fall back from a service that failed a consistency check to one that passed all consistency checks. For example, if a client has the option of using one of a set of Privacy Pass token issuers, it can choose an issuer that passes all consistency checks.

If the service that failed the consistency check is an optional optimization for the client, the client can simply choose to not use the service. For example, if a Privacy Pass token is used to avoid showing the user a CAPTCHA, but the Privacy Pass token issuer fails the consistency check, the client can fall back to showing the user a CAPTCHA.

For cases where the client has no alternate services to use, and the service is required in order to perform user-facing functionality, the client **SHOULD** report the error in a visible way that presents the

error to the user or an administrator. This functionality can be similar to how invalid TLS certificates are reported.

5.2. Selecting Mirror Servers

In many of these systems where the mirror protocol might be used, including common configurations for Privacy Pass and OHTTP, there is already a party who is necessarily trusted to protect the user's privacy, and whose operational availability is already a prerequisite for using the system. In OHTTP, this is the Relay; in Privacy Pass it might be the Attester (in Split Mode) or a transport proxy.

When such a party exists, it is **RECOMMENDED** that they operate a mirror service for their users, and that clients do not use any other mirrors for the purposes of consistency checks. This avoids revealing any metadata about the client's activity to additional parties and reduces the likelihood of an outage. More information for implementing this check in the context of Privacy Pass and OHTTP is provided in Section 5.4 and Section 5.5, respectively.

In some cases, this trusted party can provide consistency enforcement through a protocol-specific mechanism (e.g., [I-D.pw-privacypass-in-band-consistency] for Privacy Pass in Split Mode). Protocol-specific consistency mechanisms may be preferable to protocol-agnostic consistency checks based on the mirror protocol, especially if they provide equivalent consistency guarantees with better performance or reliability.

5.3. Consistency Validity Period

Executing a consistency check provides a client with some assurance that other clients may be using the same resource Section 6. However, this result only reflects a mirror's view of a resource at a particular point in time. The client should periodically re-check consistency. The frequency of re-checking depends on the requirements of the client Section 5. Clients should re-confirm consistency of a cached resource if it is not fresh (see Section 4.2 of [CACHING]).

Two strategies for maintaining consistency are: 1. Pre-emptively execute a consistency check for a resource that is expiring soon; and 1. Execute a consistency check of an expired resource at the time of its next use.

For strategy 1, clients that execute consistency checks at the same time can induce a thundering herd that overwhelms the mirror resource. Since coordinating consistency checks across clients is difficult, clients can instead execute consistency checks at random times before the resource expires. Clients that have more information about a mirror's available capacity can choose different implementations for strategy 1.

Strategy 2 might be preferrable for a service and resource that is infrequently used. However, a consequence of this strategy is that it can reveal a client's usage patterns to the mirror.

When the origin server has multiple versions of a resource corresponding to a URL, it should respond with the resource that is both currently valid and will remain fresh for the longest amount of time in the future.

5.4. Privacy Pass Profile

Clients are given as input an issuer token key from an origin server and want to check whether it is consistent with the key that is given to other clients. Let the input key be denoted token_key and its identifier be token_key_id. Clients are also given as input the name of the issuer, from which they can construct the target URL for the issuer directory. If clients have already checked this issuer’s token key, i.e., they’ve previously run a consistency check, they can simply reuse the result up to its expiration. Otherwise, clients invoke a mirror-based consistency check in parallel with the issuance protocol.

Each issuer directory can yield one or more normalized representations that clients use in the consistency check. For example, given a mirrored token directory resource like the following:

```
{
  "issuer-request-uri": "https://issuer.example.net/request",
  "token-keys": [
    {
      "token-type": 2,
      "token-key": "MI...AB",
      "not-before": 1686913811,
    },
    {
      "token-type": 2,
      "token-key": "MI...AQ",
    }
  ]
}
```

Clients compute the first valid representation of this directory, i.e., the first entry in the list that the client can use, which might be the key ID of the first key in the "token-keys" list (depending on the "not-before" value), or the key ID of the second key in the "token-keys" list. The key ID is computed as defined in Section 6.5 of [PRIVACYPASS-ISSUANCE].

5.5. Oblivious HTTP Profile

Clients can run consistency checks for OHTTP in several ways depending on the deployment. In practice, common deployments are as follows:

1. Clients are configured with gateway configurations; and

2. Clients fetch gateway configurations before use.

In both cases, clients begin with a gateway configuration and want to check it for consistency. In OHTTP, there is exactly one representation for a gateway configuration – the configuration itself. Before using the configuration to encrypt a binary HTTP message to the gateway, clients can run a consistency check with their configured mirror(s) to ensure that this configuration is correct for the given gateway.

6. Security Considerations

Consistency checks assume that the client-configured set of mirrors is honest. Under this assumption, the consistency properties of consistency checks based on the mirror protocol are as follows:

1. With honest mirrors, clients that successfully check a resource are assured that they share the same copy of the resource with the union of mirror clients for each configured mirror.

2. Consistency only holds for the period of time of the minimum mirror validity window.

3. With at least one dishonest mirror, the probability of discovering an inconsistency is 1 - (1 / 2^(k-1)), where k is the number of disjoint consistency checks. This is the probability that each individual consistency check succeeds.

Unless all clients share the same configured mirrors, consistency checks using the mirror protocol do not achieve global consistency as is defined in [CONSISTENCY].

7. IANA Considerations

This document has no IANA actions.

8. References

8.1. Normative References

[BHTTP]

Thomson, M. and C. A. Wood, "Binary Representation of HTTP Messages", RFC 9292, DOI 10.17487/RFC9292, August 2022, <https://www.rfc-editor.org/rfc/rfc9292>.

[CONSISTENCY] Davidson, A., Finkel, M., Thomson, M., and C. A. Wood, "Key Consistency and Discovery", Work in Progress, Internet-Draft, draft-ietf-privacypass-key-consistency-01, 10 July 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-privacypass-key-consistency-01>.

[OHTTP] Thomson, M. and C. A. Wood, "Oblivious HTTP", Work in Progress, Internet-Draft, draft-ietf-ohai-ohttp-10, 25 August 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-ohai-ohttp-10>.

[PRIVACYPASS] Davidson, A., Iyengar, J., and C. A. Wood, "The Privacy Pass Architecture", Work in Progress, Internet-Draft, draft-ietf-privacypass-architecture-16, 25 September 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-privacypass-architecture-16>.

[PRIVACYPASS-ISSUANCE] Celi, S., Davidson, A., Valdez, S., and C. A. Wood, "Privacy Pass Issuance Protocol", Work in Progress, Internet-Draft, draft-ietf-privacypass-protocol-16, 3 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-privacypass-protocol-16>.

[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>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/rfc/rfc6570>.

[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>.

8.2. Informative References

[CACHING] 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>.

[DOUBLE-CHECK] Schwartz, B. M., "Key Consistency by Double-Checking via a Semi-Trusted Proxy", Work in Progress, Internet-

Draft, draft-schwartz-ohai-consistency-doublecheck-03, 19 October 2022, <https://datatracker.ietf.org/doc/html/draft-schwartz-ohai-consistency-doublecheck-03>.

[I-D.pw-privacypass-in-band-consistency] Pauly, T. and C. A. Wood, "Privacy Pass In-Band Key Consistency Checks", Work in Progress, Internet-Draft, draft-pw-privacypass-in-band-consistency-00, 10 July 2023, <https://datatracker.ietf.org/doc/html/draft-pw-privacypass-in-band-consistency-00>.

Acknowledgments

This document is based on the [DOUBLE-CHECK] protocol from Benjamin Schwartz.

Authors' Addresses

Benjamin Beurdouche
Mozilla

Email: ietf@beurdouche.com

Matthew Finkel
Apple Inc.

Email: sysrqb@apple.com

Steven Valdez
Google LLC

Email: svaldez@chromium.org

Christopher A. Wood
Cloudflare

Email: caw@heapingbits.net

Tommy Pauly
Apple Inc.

Email: tpauly@apple.com