CoRE
Internet-Draft
Intended status: Experimental
Expires: 14 January 2021

C. Amsüss

M. Tiloca
RISE AB

13 July 2020

Cachable OSCORE
draft-amsuess-core-cachable-oscore-00

Abstract

OSCORE group communication can secure CoAP group communication across untrusted proxies, but in doing so sidesteps the proxies' caching abilities. This restores cachability of requests by introducing consensus requests which any client in a group can send.

Note to Readers

Discussion of this document takes place on the CORE Working Group mailing list (core@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/core/ (https://mailarchive.ietf.org/arch/browse/core/).

Source for this draft and an issue tracker can be found at https://gitlab.com/chrysn/core-cachable-oscore/ (https://gitlab.com/chrysn/core-cachable-oscore/-/tree/master).

Table of Contents

1. Introduction

With OSCORE group communication, requests and responses can be read by other group members can be read by any group member as long as pairwise mode is not used. While this can populate a proxy's cache if the proxy is a member of the group, and the proxy can use the cache to respond if is recognized by the client as an eligible server, untrusted proxies only see opaque uncachable ciphertext.

This document introduces cachability in responses in two stages, initially building on concepts developed in [I-D.tiloca-core-observe-multicast-notifications]. Caching is thus enabled for proxies that are not members of the OSCORE group, and are unaware of OSCORE in general. Allowing them to cache requests is traded against some request privacy: For clients that participate in this scheme, the proxy (and any other party that can read the network traffic) can see which clients request the same resource, and how the resource's representation changes in size over time.

As in [I-D.tiloca-core-observe-multicast-notifications], clients and servers are assumed to already be members of a suitable OSCORE group.

1.1. Procedural Status

[

This is an early idea that would bring back some concepts to OSCORE that were present as OSCON in its early drafts.

The main purpose of publishing the draft at this stage is to fathom whether the concept of a deterministic client has a chance of living up the standards of the IETF community (no pun intended).

]

2. Terminology

The reader is expected to be familiar with the terms of [I-D.ietf-core-oscore-groupcomm].

This document introduces the following new terms:

Consensus Request A Group OSCORE request that can be used repeatedly to access a particular resource.

It has all the properties relevant to caching, but its transport dependent properties (e.g. Token or Message ID) are not defined.

Thus, different requests on the wire can both be said to "be the Consensus Request" even if they have different tokens or client addresses.

Ticket Request A Consensus Request generated by the server itself.

The Phantom Request of [I-D.tiloca-core-observe-multicast-notifications] is the prototypical Ticket Request.

Deterministic Client A fictious member of an OSCORE group with no Sender Sequence Number and no Recipient Context.

The Deterministic Client is set up in the group manager,

has only the minimum common set of privileges shared by all group members.

Deterministic Request A Consensus Request generated by the Deterministic Client.

3. Simple Cachability using Ticket Requests

Building on phantom requests and informative responses of [I-D.tiloca-core-observe-multicast-notifications], basic proxying operation is already possible with the mechanisms described there:

A server can, instead of sending a regular response, send an informative response, which is a protected 5.03 error message whose payload contains the phantom request (which is a Ticket Request in this document's broader terminology).

Even though the request is not necessarily an observe request, the server picks FETCH as the outer code of the request in order to make the request cachable.

The client verifies that the ticket request is indeed equivalent to its original request, and - and this is where the process starts to deviate from multicast notifications - sends the ticket request to the server through the proxy.

As with multicast notifications, this check especially verifies that the request URI, including protocol and host name, is identical between the original and the Ticket Request. Any difference there would indicate URI aliasing, which is not allowed initially.

When the server receives the ticket request, it produces a regular response, but puts a non-zero Max-Age option as an outer option. (There is no point in putting in an inner Max-Age option, as the client could not pin it in time).

When another client later asks for the same resource, its new request produces a cache miss at the proxy (as it uses a different KID and Partial IV), but the server responds with the same Ticket Request. The Ticket Request can then be served from the proxy's cache.

When multiple proxies are in use, or the response was expires from the proxy's cache, the server will receive the Ticket Request multiple times. It is a matter of perspective whether it treats that as an acceptable replay (given that this whole mechansim only makes sense on side effect free requests), or whether it is conceptualized as having an internal proxy where the request produces a cache hit.

3.1. Usefullness

As all clients' requests produce an initial cache miss and thus hit the origin server, the caching benefits of such an approach are limited to two cases:

* observations (where this can be used to set up multicast notifications through proxies), and

* large representations that are use outer block-wise mode (which are probably rare compared to inner block-wise mode).

For any other case, the benefit of caching a single response of only up to 1kB in size is probably outweighed by the necessity to have an additional round trip, or at least drastically reduces the gains.

The mechanism could probably be extended to work for inner block-wise as well (by introducing an option by which the server sends the next-block Ticket Request along with the response). However, there has to be a better way...

4. Deterministic Requests

This section introduces a method of arriving at a Consensus Request inside the client: Rather than relying on the server to decree a Token Request, clients build their request in as reproducible a fashion as possible (where some disagreement might be eventually unavoidable, but won't have more severe a consequence than two requests for the same resource occupying space in the caches).

The hard part is arriving at a consensus nonce, while avoiding nonce reuse.

A suitable nonce can be produced by applying a cryptographic hash function to the complete input of the encryption operation, which is the plaintrext of the COSE object and the AAD (with the partial IV set to 0).

(The precise hashing mechanism is yet to be defined, but any non-malleable cryptographically secure hash should do, and malleable hashes can be permitted if the input material is adaequately encapsulated before hashing).

As the 40 bit available in the nonce are by far insufficient to ensure that the deterministic client's nonce is not reused, (and even with some trickery based on the deterministic client never responding to other members' requests with their nonces, the common algorithms' nonces would still be too short), the hash has to be fed into the key generation rather than the encyption's nonce in a new mechanism.

4.1. ID-Detail

A new field in the OSCORE object is defined, named ID-Detail.

It is in every way analogous to ID-Context, is concatenated onto ID-Context when deriving keys in the info input to the KDF, and is only distinct from ID-Context to allow using both ID-Context and ID-Detail at the same time.

It goes into the "unprotected" bucket, and is serialized in the compressed OSCORE option using an indicator flag in the 8-63 range.

[Once that is specified, it would be much easier to execute OSCORE B.2 mode on that field rather than on the ID-Context - the effect is the same, but it does not collide with namespacing of ID-Context any more.]

4.2. Use of Deterministic Requests

A client that sends a request for which it hopes to get a cached response can ask the group manager for the key details of the Deterministic Client.

In addition to the public key data, it also receives the private key generated by the group manager.

It builds the request, hashes it, places the hash (or some truncation thereof) in the ID-Detail field, and finishes derivation of its security context for this request. It uses 0 as the Partial IV, and encrypts the message. It uses FETCH as the outer code to make it cachable, even if no observation is requested. As the key is derived using material from the whole request, this key/nonce pair is only used for this very message and deterministically encrypted unless there is a hash collision between two deterministic requests.

It then sends the request through the proxy to the server.

The server applies the regular processing to it, deriving the security context based on the ID-Detail.

As the recipient context for a deterministic client does not have a sequence number to strike out of the replay window, the server needs to apply the reasoning of [I-D.amsuess-lwig-oscore] to treat the potential replay as answerable if the request is side effect free.

By setting a non-zero Max-Age option, the server makes the request usable for the proxy cache.

5. Open questions

* Can the informative response be unprotected?

  Otherwise, how would a proxy forwarding the Ticket Request to a multicast-notification network learn the relevant token?

  (The client shouldn't really trust the server's statement about the requests' equivalence anyway).

* What can go wrong if we use a shared private key?

6. Unsorted further ideas

* All or none of the deterministic requests should have an inner observe option. Preferably none - that makes messages shorter, and clients need to ignore that option either way when checking whether a Consensus Request matches their intended request.

* An outer ETag does make sense here; an easy value for the server is the response Partial IV.

7. References

7.1. Normative References

[I-D.tiloca-core-observe-multicast-notifications]
Tiloca, M., Hoeglund, R., Amsuess, C., and F. Palombini, "Observe Notifications as CoAP Multicast Responses", Work in Progress, Internet-Draft, draft-tiloca-core-observe-multicast-notifications-03, 13 July 2020, <http://www.ietf.org/internet-drafts/draft-tiloca-core-observe-multicast-notifications-03.txt>.

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", Work

in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-09, 23 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-core-oscore-groupcomm-09.txt>.

7.2. Informative References

[I-D.amsuess-lwig-oscore]
Amsuess, C., "OSCORE Implementation Guidance", Work in Progress, Internet-Draft, draft-amsuess-lwig-oscore-00, 29 April 2020, <http://www.ietf.org/internet-drafts/draft-amsuess-lwig-oscore-00.txt>.

Authors' Addresses

Christian Amsüss
Austria

Email: christian@amsuess.com

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden

Email: marco.tiloca@ri.se