CoRE Working Group
Internet-Draft
Updates: RFC7959 (if approved)
Intended status: Standards Track
Expires: January 2, 2018

C. Amsuess
Energy Harvesting Solutions
J. Mattsson
G. Selander
Ericsson AB
July 01, 2017

Repeat And Request-Tag
draft-amsuess-core-repeat-request-tag-00

Abstract

This document defines two optional extensions to the Constrained Application Protocol (CoAP): the Repeat option and the Request-Tag option. Each of these options when integrity protected, such as with DTLS or OSCOAP, protects against certain attacks on CoAP message exchanges.

The Repeat option enables a CoAP server to verify the freshness of a request by requiring the CoAP client to make another request and include a server-provided challenge. The Request-Tag option allows the CoAP server to match message fragments belonging to the same request message, fragmented using the CoAP Block-Wise Transfer mechanism. This document also specifies additional processing requirements on Block1 and Block2 options.

Status of This Memo

Table of Contents

1. Introduction

The initial CoAP suite of specifications ([RFC7252], [RFC7641], [RFC7959]) was designed with the assumption that security could be provided on a separate layer, in particular by using DTLS ([RFC6347]). However, for some use cases, additional functionality or extra processing is needed to support secure CoAP operations.

This document specifies two server-oriented CoAP options, the Repeat option and the Request-Tag option, addressing the security features request freshness and fragmented message body integrity, respectively. These options in themselves do not replace the need for a security protocol; they specify the format and processing of data which, when integrity protected in a message, e.g. using DTLS ([RFC6347]) or OSCOAP ([I-D.ietf-core-object-security]), provide those security features. The Request-Tag option and also the ETag option are mandatory to use with Block1 and Block2, respectively, to secure blockwise operations with multiple representations of a particular resource as is specified in this document.

1.1. Request Freshness

A CoAP server receiving a request may not be able to verify when the request was sent by the CoAP client. This remains true even if the request was protected with a security protocol, such as DTLS. This makes CoAP requests vulnerable to certain delay attacks which are particularly incriminating in the case of actuators ([I-D.mattsson-core-coap-actuators]). Some attacks are possible to mitigate by establishing fresh session keys (e.g. performing the DTLS handshake) for each actuation, but in general this is not a solution suitable for constrained environments.

A straightforward mitigation of potential delayed requests is that the CoAP server rejects a request the first time it appears and asks the CoAP client to prove that it intended to make the request at this point in time. The Repeat option, defined in this document, specifies such a mechanism which thereby enables the CoAP server to verify the freshness of a request. This mechanism is not only important in the case of actuators, or other use cases where the CoAP operations require freshness of requests, but also in general for synchronizing state between CoAP client and server.

1.2. Fragmented Message Body Integrity

CoAP was designed to work over unreliable transports, such as UDP, and include a lightweight reliability feature to handle messages which are lost or arrive out of order. In order for a security protocol to support CoAP operations over unreliable transports, it must allow out-of-order delivery of messages using e.g. a sliding replay window such as described in Section 4.1.2.6 of DTLS ([RFC6347]).

The Block-Wise Transfer mechanism [RFC7959] extends CoAP by defining the transfer of a large resource representation (CoAP message body) as a sequence of blocks (CoAP message payloads). The mechanism uses a pair of CoAP options, Block1 and Block2, pertaining to the request

and response payload, respectively. The blockwise functionality does not support the detection of interchanged blocks between different message bodies to the same endpoint having the same block number. This remains true even when CoAP is used together with a security protocol such as DTLS or OSCOAP, within the replay window ([I-D.amsuess-core-request-tag]), which is a vulnerability of CoAP when using RFC7959.

A straightforward mitigation of mixing up blocks from different messages is to use unique identifiers for different message bodies, which would provide equivalent protection to the case where the complete body fits into a single payload. The ETag option [RFC7252], set by the CoAP server, identifies a response body fragmented using the Block2 option. This document defines the Request-Tag option for identifying the request body fragmented using the Block1 option, similar to ETag, but ephemeral and set by the CoAP client.

1.3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Unless otherwise specified, the terms "client" and "server" refers to "CoAP client" and "CoAP server", respectively, as defined in [RFC7252].

The terms "payload" and "body" of a message are used as in [RFC7959]. The complete interchange of a request and a response body is called a (REST) "operation", while a request and response message (as matched by their tokens) is called an "exchange". An operation fragmented using [RFC7959] is called a "blockwise operation". A blockwise operation which is fragmenting the request body is called a "blockwise request operation". A blockwise operation which is fragmenting the response body is called a "blockwise response operation".

Two blockwise operations between the same endpoint pair on the same resource are said to be "concurrent" if a block of the second request is exchanged even though the client still intends to exchange further blocks in the first operation. (Concurrent blockwise request operations are impossible with the options of [RFC7959] because the second operation's block overwrites any state of the first exchange.).

The Repeat and Request-Tag options are defined in this document. The concept "Request-Tag value" is defined in Section 3.1.

2. The Repeat Option

The Repeat option is a server-driven challenge-response mechanism for CoAP. The Repeat option value is a challenge from the server to the client included in a CoAP response and echoed in a CoAP request.

2.1. Option Format

The Repeat Option is elective, safe-to-forward, not part of the cache-key, and not repeatable, see Figure 1. Note that the Repeat option has nothing to do with the property of an option being repeatable, i.e. be allowed to occur more than once in a message, as defined in Section 5.4.5 of [RFC7252].

No.	C	U	N	R	Name	Format	Length	Default	E
TBD					Repeat	opaque	8-40	(none)	x

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable,
E=Encrypt and Integrity Protect (when using OSCOAP)

Figure 1: Repeat Option Summary

The value of the Repeat option MUST be a (pseudo-)random bit string of a length of at least 64 bits. A new (pseudo-)random bit string MUST be generated by the server for each use of the Repeat option.

2.2. Repeat Processing

It is important to identify under what conditions a CoAP request to a resource is required to be fresh. These conditions can for example include what resource is requested, the request method and other data in the request, and conditions in the environment such as the state of the server or the time of the day.

A server MAY include the Repeat option in a response. The Repeat option MUST NOT be used with empty CoAP requests. If the server receives a request which has freshness requirements, and the request does not contain the Repeat option, the server SHOULD send a 4.03 Forbidden response with a Repeat option. The server SHOULD cache the transmitted Repeat option value and the response transmit time (here denoted t0).

Upon receiving a response with the Repeat option within the EXCHANGE_LIFETIME ([RFC7252]) of the original request, the client

SHOULD echo the Repeat option with the same value in a new request to the server. Upon receiving a 4.03 Forbidden response with the Repeat option in response to a request within the EXCHANGE_LIFETIME of the original request, the client SHOULD resend the original request. The client MAY send a different request compared to the original request.

If the server receives a request which has freshness requirements, and the request contains the Repeat option, the server MUST verify that the option value equals a cached value; otherwise the request is not processed further. The server MUST calculate the round-trip time RTT = (t1 - t0), where t1 is the request receive time. The server MUST only accept requests with a round-trip time below a certain threshold T, i.e. RTT < T, otherwise the request is not processed further, and an error message MAY be sent. The threshold T is application specific, its value depends e.g. on the freshness requirements of the request. An example message flow is illustrated in Figure 2.

When used to serve freshness requirements, CoAP messages containing the Repeat option MUST be integrity protected, e.g. using DTLS or OSCOAP ([I-D.ietf-core-object-security]).

If the server loses time synchronization, e.g. due to reboot, it MUST delete all cached Repeat option values and response transmission times.

```
                   Client  Server
                      |      |
                      +------>|        Code: 0.03 (PUT)
                      | PUT  |       Token: 0x41
                      |      |    Uri-Path: lock
                      |      |     Payload: 0 (Unlock)
                      |      |
                      |<------+ t0     Code: 4.03 (Forbidden)
                      | 4.03 |       Token: 0x41
                      |      |      Repeat: 0x6c880d41167ba807
                      |      |
                      +------>| t1     Code: 0.03 (PUT)
                      | PUT  |       Token: 0x42
                      |      |    Uri-Path: lock
                      |      |      Repeat: 0x6c880d41167ba807
                      |      |     Payload: 0 (Unlock)
                      |      |
                      |<------+        Code: 2.04 (Changed)
                      | 2.04 |       Token: 0x42
                      |      |
```

Figure 2: Repeat option message flow

Constrained server implementations can use the mechanisms outlined in Appendix A to minimize the memory impact of having many unanswered Repeat responses.

2.3. Applications

1. Actuation requests often require freshness guarantees to avoid accidental or malicious delayed actuator actions.

2. To avoid additional roundtrips for applications with multiple actuator requests in rapid sequence between the same client and server, the server may use the Repeat option (with a new value) in response to a request containing the Repeat option. The client then uses the Repeat option with the new value in the next actuation request, and the server compares the receive time accordingly.

3. If a server reboots during operation it may need to synchronize state with requesting clients before continuing the interaction. For example, with OSCOAP it is possible to reuse a persistently stored security context by synchronizing the Partial IV (sequence number) using the Repeat option.

4. When a device joins a multicast/broadcast group the device may need to synchronize state or time with the sender to ensure that the received message is fresh. By synchronizing time with the broadcaster, time can be used for synchronizing subsequent broadcast messages. A server MUST NOT synchronize state or time with clients which are not the authority of the property being synchronized. E.g. if access to a server resource is dependent on time, then the client MUST NOT set the time of the server.

5. A server that sends large responses to unauthenticated peers and wants to mitigate the amplification attacks described in Section 11.3 of [RFC7252] (where an attacker would put a victim's address in the source address of a CoAP request) can ask a client to Repeat its request to verify the source address. This needs to be done only once per peer, and limits the range of potential victims from the general Internet to endpoints that have been previously in contact with the server. For this application, the Repeat option can be used in messages that are not integrity protected, for example during discovery.

3. The Request-Tag Option

The Request-Tag is intended for use as a short-lived identifier for keeping apart distinct blockwise request operations on one resource from one client. It enables the receiving server to reliably assemble request payloads (blocks) to their message bodies, and, if it chooses to support it, to reliably process simultaneous blockwise request operations on a single resource. The requests must be be integrity protected in order to protect against interchange of blocks between different message bodies.

3.1. Option Format

The Request-Tag option has the same properties as the Block1 option: it is critical, unsafe, not part of the cache-key, and not repeatable, see Figure 3.

No.	C	U	N	R	Name	Format	Length	Default	E
TBD	x	x	-		Request-Tag	opaque	0-8	(none)	*

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable,
E=Encrypt and Integrity Protect (when using OSCOAP)

Figure 3: Request-Tag Option Summary

[Note to RFC editor: If this document is not released together with OSCOAP but before it, the following paragraph and the "E" column above need to move into OSCOAP.]

Request-Tag, like the Block1 option, is a special class E option in terms of OSCOAP processing (see Section 4.3.1.2 of [I-D.ietf-core-object-security]): The Request-Tag MAY be an inner or outer option. The inner option is encrypted and integrity protected between client and server, and provides message body identification in case of end-to-end fragmentation of requests. The outer option is visible to proxies and labels message bodies in case of hop-by-hop fragmentation of requests.

Every message in which the Block1 option is set is considered to carry a "Request-Tag value" that can be compared for equality with the value of any other such message. Absence of the Request-Tag option implies a value that is distinct from any value of a message with the Request-Tag option set, and equal to that of any other message without the Request-Tag option. Messages with the Request-Tag option set have equal Request-Tag values if and only if their option lengths and option values are equal.

The value of the Request-Tag option is generated by the client for every blockwise request operation. Clients are encouraged to generate compact Request-Tag values. It MUST be different from any other Request-Tag value used in the same security context on the same resource, but MAY be reused when all earlier operations with the same value are concluded. What constitutes a concluded operation depends on the application, and is outlined individually in Section 3.3.

3.2. Request-Tag Processing

A server MUST NOT act on any two blocks in the same blockwise request operation that have different Request-Tag values. This also means that a block cannot overwrite kept context when the Request-Tag does not match (cf. [RFC7959] Section 2.5). The server is still under no obligation to keep state of more than one transaction. When an operation is in progress and a second one cannot be served at the same time, the server MUST respond to the second request with a 5.03 (Service Unavailable) response code and SHOULD indicate the time it is willing to wait for additional blocks in the first operation using the Max-Age option, as specified in Section 5.9.3.4 of [RFC7252].

A server receiving a Request-Tag MUST treat it as opaque and make no assumptions about its content or structure.

Two messages arriving at the server with the same Request-Tag value do not necessarily belong to the same operation. They can still be

treated as independent messages by the server (e.g. when it sends 2.01/2.04 responses for every block), or initiate a new operation (overwriting kept context) when sending the first block again.

The Request-Tag option is not used in responses.

If a request that uses Request-Tag is rejected with 4.02 Bad Option, the client MAY retry the operation without it, but then it MUST serialize all operations that affect the same resource. Security requirements can forbid dropping the Request-Tag option.

3.3. Applications

3.3.1. Body Integrity Based on Payload Integrity

When a client fragments a request body into multiple message payloads, even if the individual messages are integrity protected, it is still possible for a man-in-the-middle to maliciously replace later operation's blocks with earlier operation's blocks (see Section 3.2 of [I-D.amsuess-core-request-tag]). Therefore, the integrity protection of each block does not extend to the operation's request body.

In order to gain that protection, use the Request-Tag mechanism as follows:

- The message payloads MUST be integrity protected end-to-end between client and server.

- The client MUST NOT reuse a Request-Tag value within a security association unless all previous blockwise request operations on the same resource that used the same Request-Tag value have concluded.

  Note that the server needs to verify that all blocks within an operation come from the same security association, because the security association is a part of the endpoint as per [RFC7252].

- The client MUST NOT regard a blockwise request operation as concluded unless all of the messages the client previously sent in the operation have been confirmed by the message integrity protection mechanism, or are considered invalid by the server if replayed.

  Typically, in OSCOAP, these confirmations can result either from the client receiving an OSCOAP response message matching the request (an empty ACK is insufficient), or because the message's

sequence number is old enough to be outside the server's receive window.

In DTLS, this can only be confirmed if the request message was not retransmitted, and was responded to.

Authors of other documents (e.g. [I-D.ietf-core-object-security]) are invited to mandate this behavior for clients that execute blockwise interactions over secured transports. In this way, the server can rely on a conforming client to set the Request-Tag option when required, and thereby conclude on the integrity of the assembled body.

Note that this mechanism is implicitly implemented when the security layer guarantees ordered delivery (e.g. CoAP over TLS [I-D.tschofenig-core-coap-tcp-tls]). This is because with each message, any earlier operation can be regarded as concluded by the client, so it never needs to set the Request-Tag option unless it wants to perform concurrent operations.

3.3.2. Multiple Concurrent Blockwise Operations

CoAP clients, especially CoAP proxies, may initiate a blockwise request operation to a resource, to which a previous one is already in progress, and which the new request should not cancel. One example is when a CoAP proxy fragments an OSCOAP messages using blockwise (so-called "outer" blockwise, see Section 4.3.1. of [I-D.ietf-core-object-security])), where the Uri-Path is hidden inside the encrypted message, and all the proxy sees is the server's "/" path.

When a client fragments a message as part of a blockwise request operation, it can do so without a Request-Tag option set. For this application, an operation can be regarded as concluded when a final Block1 option has been sent and acknowledged, or when the client chose not to continue with the operation (e.g. by user choice, or in the case of a proxy when it decides not to take any further messages in the operation due to a timeout). When another concurrent blockwise request operation is made (i.e. before the operation is concluded), the client can use a different Request-Tag value (as specified in Section 3.1). The possible outcomes are:

o The server responds with a successful code.

  The concurrent blockwise operations can then continue.

o The server responds 4.02 Bad Option.

This can indicate that the server does not support Request-Tag. The client should wait for the first operation to conclude, and then try the same request without a Request-Tag option.

o The server responds 5.03 Service Unavailable with a Max-Age option to indicate when it is likely to be available again.

  This can indicate that the server supports Request-Tag, but still is not prepared to handle concurrent requests. The client should wait for as long as the response is valid, and then retry the operation, which may not need to carry a Request-Tag option by then any more.

In the cases where a CoAP proxy receives an error code, it can indicate the anticipated delay by sending a 5.03 Service Unavailable response itself. Note that this behavior is no different from what a CoAP proxy would need to do were it unaware of the Request-Tag option.

4. Block2 / ETag Processing

The same security properties as in Section 3.3.1 can be obtained for blockwise response operations. The threat model here is not an attacker (because the response is made sure to belong to the current request by the security layer), but blocks in the client's cache.

Analogous rules to Section 3.2 are already in place for assembling a response body in Section 2.4 of [RFC7959].

To gain equivalent protection to Section 3.3.1, a server MUST use the Block2 option in conjunction with the ETag option ([RFC7252], Section 5.10.6), and MUST NOT use the same ETag value for different representations of a resource.

5. IANA Considerations

[TBD: Fill out the option templates for Repeat and Request-Tag]

6. Security Considerations

Servers that store a Repeat challenge per client can be attacked for resource exhaustion, and should consider minimizing the state kept per client, e.g. using a mechanism as described in Appendix A.

7. References

7.1. 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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <http://www.rfc-editor.org/info/rfc7959>.

7.2. Informative References

[I-D.amsuess-core-request-tag]
Amsuess, C., "Request-Tag option", draft-amsuess-core-request-tag-00 (work in progress), March 2017.

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-03 (work in progress), May 2017.

[I-D.mattsson-core-coap-actuators]
Mattsson, J., Fornehed, J., Selander, G., and F. Palombini, "Controlling Actuators with CoAP", draft-mattsson-core-coap-actuators-02 (work in progress), November 2016.

[I-D.tschofenig-core-coap-tcp-tls]
Bormann, C., Lemay, S., Technologies, Z., and H. Tschofenig, "A TCP and TLS Transport for the Constrained Application Protocol (CoAP)", draft-tschofenig-core-coap-tcp-tls-05 (work in progress), November 2015.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <http://www.rfc-editor.org/info/rfc7641>.

Appendix A. Performance Impact When Using the Repeat Option

The Repeat option requires the server to keep some state in order to later verify the repeated request.

Instead of caching Repeat option values and response transmission times, the server MAY use the encryption of the response transmit time t0 as Repeat option value. Such a scheme needs to ensure that the server can detect a replay of a previous encrypted response transmit time.

For example, the server MAY encrypt t0 with AES-CCM-128-64-64 using a (pseudo-)random secret key k generated and cached by the server. A unique IV MUST be used with each encryption, e.g. using a sequence number. If the server loses time synchronization, e.g. due to reboot, then k MUST be deleted and replaced by a new random secret key. When using encrypted response transmit times, the Repeat processing is modified in the following way: The verification of cached option value in the server processing is replaced by the verification of the integrity of the encrypted option value using the cached key and IV (e.g. sequence number).

The two methods - (a) the list of cached values, and (b) the encryption of transmit time - have different impact on the implementation:

- size of cached data (list of cached values vs. key and IV)

- size of message (typically larger with encrypted time)

- computation (encryption + decryption vs. generation new nonce + cache + lookup)

In general, the encryption of transmission times is most useful if the number of concurrent requests is high.

A hybrid scheme is also possible: the first Repeat option values are cached, and if the number of concurrent requests reach a certain threshold, then encrypted times are used until there is space for storing new values in the list. In that case, the server may need to make both verifications - either that the Repeat value is in the list, or that it verifies in decryption - and in either case that the transmission time is valid.

Appendix B. Request-Tag Message Size Impact

In absence of concurrent operations, the Request-Tag mechanism for body integrity (Section 3.3.1) incurs no overhead if no messages are lost (more precisely: in OSCOAP, if no operations are aborted due to repeated transmission failure; in DTLS, if no packages are lost), or when blockwise request operations happen rarely (in OSCOAP, if only one request operation with losses within the replay window).

In those situations, the Request-Tag value of no Request-Tag option present can be reused over and over again.

When the "no-Request-Tag value" is used-up within a security context, the Request-Tag value of a present but empty option can be used (1 Byte overhead), and when that is used-up, 256 values from one byte long options (2 Bytes overhead) can be used.

In situations where those overheads are unacceptable (e.g. because the payloads are known to be at a fragmentation threshold), the absent Request-Tag value can be made usable again:

o In DTLS, a new session can be established.

o In OSCOAP, the sequence number can be artificially increased so that all lost messages are outside of the replay window by the time the first request of the new operation gets processed, and all earlier operations can therefore be regarded as concluded.

Authors' Addresses

Christian Amsuess
Energy Harvesting Solutions

Email: c.amsuess@energyharvesting.at

John Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com