CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 30, 2018

G. Selander
F. Palombini
Ericsson AB
K. Hartke
Universitaet Bremen TZI

July 29, 2017

Requirements for CoAP End-To-End Security
draft-hartke-core-e2e-security-reqs-03

Abstract

This document analyses threats to CoAP message exchanges traversing proxies and derives security requirements for mitigating those threats.

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 http://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 January 30, 2018.

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a Web application protocol designed for constrained nodes and networks [RFC7228]. CoAP makes use of Datagram Transport Layer Security (DTLS) [RFC6347] for security. At the same time, CoAP relies on proxies for scalability and efficiency. Proxies reduce response time and network bandwidth use by serving responses from a shared cache or enable clients to make requests that these otherwise could not make.

CoAP proxies need to perform a number of operations on requests and responses to fulfill their purpose, which requires the DTLS security associations to be terminated at each proxy. The proxies therefore do not only have access to the data required for performing the desired functionality, but are also able to eavesdrop on or manipulate any part of the CoAP payload and metadata exchanged between client and server, or inject new CoAP messages without being protected or detected by DTLS.

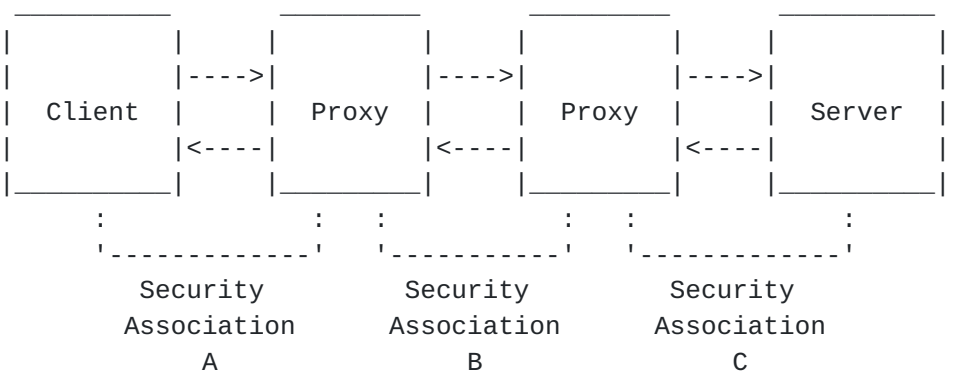

Figure 1: Hop-by-Hop Security

One way to mitigate this threat is to secure CoAP communication at the application layer using an object-based security mechanism such as CBOR Object Signing and Encryption (COSE) [RFC8152] instead of or in addition to the security mechanisms at the network layer or transport layer. Such a mechanism can provide "end-to-end security"

at the CoAP layer (Figure 2) in contrast to the "hop-by-hop security" that DTLS provides at the CoAP layer (Figure 1).

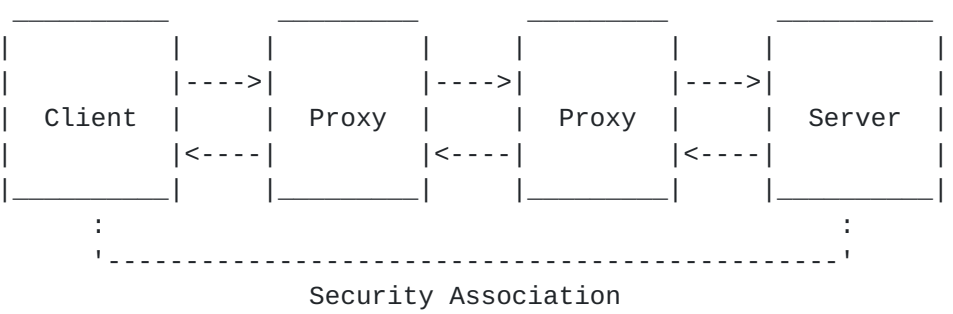

Figure 2: End-to-End Security

This document analyses security aspects of sensor and actuator communications over CoAP that involve proxies (Section 2) and publish-subscribe brokers (Section 3). The analysis is based on the identification of assets associated with these communications and considering the potential threats posed by proxies to these assets. The threat analysis provides the basis for deriving security requirements that a solution for CoAP end-to-end security should meet.

1.1. Assets and Scope

In general, there are the following assets that need to be protected:

- The devices at the two ends and their (often very constrained) system resources such as available memory, storage, processing power and energy.

- The physical environment of the devices fitted with sensors and actuators. Access to the physical environment is assumed to be provided through CoAP resources that allow a remote entity to retrieve information about the physical environment (such as the current temperature) or to produce an effect on the physical environment (such as the activation of a heater).

- The communication infrastructure linking the two devices, which often contains some very constrained networks.

- The data generated and stored in the involved devices.

An intermediary can directly interfere with the interactions between the two ends and thereby have an impact on all these assets. For example, flooding a device with messages has an impact on system resources, and the successful manipulation of an actuator command

(data generated by an involved device) can have a severe impact on the physical environment. An intermediary can also affect the communication infrastructure, e.g., by dropping messages.

Even if an intermediary is trustworthy, it may be an attractive target for an attack, since such nodes are aggregation points for message flows and may be an easier target from the Internet than the sensor and actuator nodes residing behind them. An intermediary may become subject to intrusion or be infected by malware and perform the attacks of a man-in-the-middle.

The focus of this document is on threats from intermediaries to interactions between two CoAP endpoints. Other types of threats, for example, attacks involving physical access to the CoAP-speaking devices, are out of scope of this document.

Since intermediaries may perform a service for the interacting endpoints, there is a trade-off between the intermediaries' desired functionality and the ability to mitigate threats to the endpoints executed through an intermediary.

1.2. Terminology

Readers are expected to be familiar with the terms and concepts described in [RFC7252] and [RFC7641].

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 [RFC2119].

The key word "NOT REQUIRED" is interpreted as synonymous with the key word "OPTIONAL".

2. Proxying

To assess what impact various threats have to the assets, we need to specify and analyse how the proxies operate.

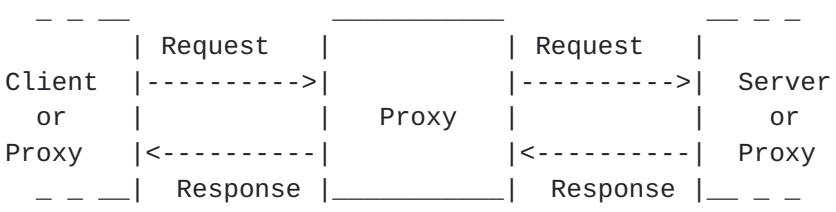

Figure 3: A Proxy

Generally speaking, the functionality of a proxy is to receive a request from a client and to send a response back to that client. There are two ways for the proxy to satisfy the request:

- The proxy constructs and sends a request to the server indicated in the client's request, receives a response from that server and uses the received data to construct the response to the client.

- The proxy uses cached data to construct the response to the client.

In both cases, the proxy needs to read some parts both of the request from the client and the response from the server to accomplish its task.

The following subsections analyse the threats posed by a proxy from the perspective of the client on the one hand (Section 2.1.1) and the perspective of the server on the other hand (Section 2.1.2). Section 2.2 then presents the design space for possible security solutions to mitigate the threats.

2.1. Threats and Security Requirements

2.1.1. Client-side

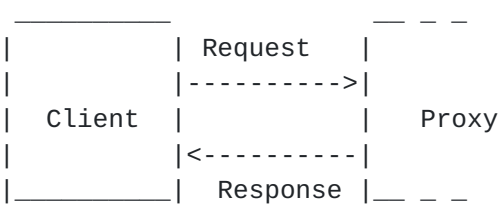

Figure 4: The Client End

The client sends a request to the proxy and waits for a response.

From the perspective of the client, there are three possible flows:

- The client receives a response.
  Reasons include:
  - The proxy duly processed the request and returns a response based on data it obtained from the origin server.
  - The proxy encountered an unexpected condition and returns an error response according to specification (e.g., 5.02 Bad Gateway or 5.04 Gateway Timeout).
  - (Threat 1:) The proxy spoofs a response. For example, the proxy could return a stale or outdated response based on data it previously obtained from the server or some fourth party, or could craft an illicit response itself.
  - (Threat 2:) The proxy duly processed the request but delays the return of the response.
- The client does not receive a response.
  Reasons include:
  - The client times out too early.
  - (Threat 3:) The proxy withholds the response.
- The client receives too many responses.
  Reasons include:
  - (Threat 4:) The proxy floods the client with responses.

Furthermore, there are threats related to privacy:

- (Threat 5:) The proxy eavesdrops on the data in the request from the client.

- (Threat 6:) The proxy measures the size, frequency or distribution of requests from the client.

Note that "cache poisoning" -- the case of caching injected incorrect responses -- is covered from the point of view of the client: it may result in the client receiving a spoofed message or being flooded, or affect other nodes such that the client times out too early.

2.1.1.1. Threat 1: Spoofing

With one exception (see below), this threat is REQUIRED to be mitigated by the security solution: the client MUST verify that the response is an "authentic response" before processing it.

The definition of an "authentic response" depends on the desired proxy functionality and protection level (see Section 2.2), but usually means that the client can obtain proof for some or all of the following items:

- that the requested action was executed by the origin server;

- that the data originates from the origin server and has not been altered on the way;

- that the data matches the specifications of the request (such as the target resource);

- that the data is fresh (when the data is cacheable);

- that the data is in sequence (when observing a resource).

The proof can, for example, involve a message authentication code that the proxy obtains from the origin server and includes in the response or an additional challenge-response roundtrip.

Exception: A CoAP proxy is specified to return an error response (such as 5.02 Bad Gateway or 5.04 Gateway Timeout) when it encounters an error condition. Since the condition occurs at the proxy and not at the origin server, the response will not be an "authentic response" according to the above definition. (A proxy cannot obtain a proof that the server is unreachable from an unreachable server.) Thus, a client cannot tell if the proxy sends the response according to specification or if it spoofs the response. This threat is NOT REQUIRED to be mitigated by the security solution.

2.1.1.2. Threat 2: Delaying

This threat is REQUIRED to be mitigated by the security solution. Delay attacks are important to mitigate in certain applications, e.g., when using CoAP with actuators. A detailed problem statement and candidate solution can be found in [I-D.mattsson-core-coap-actuators].

2.1.1.3. Threat 3: Withholding

This threat is NOT REQUIRED to be mitigated by the security solution, since a client cannot tell if the proxy does not send a response because it is hasn't received a response from the origin server yet or if it intentionally withholds the response.

2.1.1.4. Threat 4: Flooding

A CoAP client is specified to reject any response that it does not expect. This can happen before the client verifies whether the response is authentic. Therefore, a flood of responses is primarily a threat to the system resources of the client, in particular to its energy. This threat is NOT REQUIRED to be mitigated by the security solution, but a client SHOULD generally defend against flooding attacks.

2.1.1.5. Threat 5: Eavesdropping

This threat is REQUIRED to be mitigated by the security solution: clients MUST confidentiality protect the data in the requests they send.

Note that this requirement is in conflict with the requirement that the proxy needs to be able to read some parts of the requests in order to accomplish its task. Section 2.2 analyses which parts can be encrypted depending on the desired proxy functionality and protection level. In general, a security solution SHOULD confidentiality protect all data that is not needed by the proxy to accomplish its task.

The keys used for confidentiality protection MUST provide forward secrecy.

2.1.1.6. Threat 6: Traffic Analysis

This threat is NOT REQUIRED to be mitigated by the security solution.

It is RECOMMENDED that applications analyse the risks associated with application information leaking from the messages flow and assess the

feasibility to protect against various threats, e.g., by obfuscating parameters transported in plain text, aligning message flow and traffic between the different cases, adding padding so different messages become indistinguishable, etc.

2.1.2. Server-side

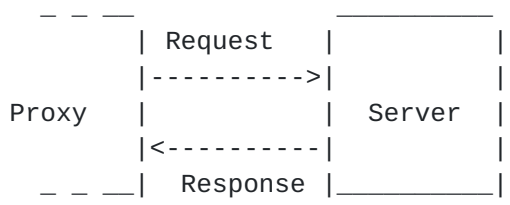

Figure 5: The Server End

A server listens for a request and returns a response.

From the perspective of the server, there are three possible flows:

- The server receives a request.
  Reasons include:
  - The proxy makes a request on behalf of a client according to specification.
  - The proxy makes a request (e.g., to validate cached data) on its own behalf.
  - (Threat 1:) The proxy spoofs a request.
  - (Threat 2:) The proxy sends a request with delay.
- The server does not receive a request.
  Reasons include:
  - The proxy does not need to send a request right now.
  - (Threat 3:) The proxy withholds a request.
- The server receives too many requests.
  Reasons include:
  - (Threat 4:) The proxy floods the server with requests.

A proxy eavesdropping or inferring information from messages it operates on has an impact on a server in the same way as on a client:

- (Threat 5:) The proxy eavesdrops on the data in the response from the server.
- (Threat 6:) The proxy measures the size, frequency or distribution of responses from the server.

2.1.2.1. Threat 1: Spoofing

With one exception (see below), this threat is REQUIRED to be mitigated by the security solution: the server MUST verify that the request is an "authentic request" before processing it.

The definition of an "authentic request" depends on the desired proxy functionality and protection level (Section 2.2), but usually means that the server can obtain proof for some or all of the following items:

- that the proxy acts on behalf of a client;
- that the data originates from the client and has not been altered on the way;
- that the request has not been received previously.

The proof can, for example, involve a message authentication code that the proxy obtains from the client and includes in the request or a challenge-response roundtrip.

Exception: A CoAP proxy may make certain requests without acting on behalf of a client (e.g., to validate cached data). Since such a request does not originate from a client, the server cannot tell if the proxy sends the request according to specification or if it spoofs the request. It is up to the security solution how this issue is addressed.

2.1.2.2. Threat 2: Delaying

This threat is REQUIRED to be mitigated by the security solution; see Section 2.1.1.2.

2.1.2.3. Threat 3: Withholding

This threat is NOT REQUIRED to be mitigated by the security solution, since a server cannot tell if the proxy does not send a request because it has no work to do or if it intentionally withholds a request.

2.1.2.4. Threat 4: Flooding

This threat is NOT REQUIRED to be mitigated by the security solution in particular, but a server SHOULD generally defend against flooding attacks.

2.1.2.5. Threat 5: Eavesdropping

This threat is REQUIRED to be mitigated by the security solution; see Section 2.1.1.5.

2.1.2.6. Threat 6: Traffic Analysis

This threat is NOT REQUIRED to be mitigated by the security solution; see Section 2.1.1.6.

2.2. Solutions

A security solution has to find a trade-off between desired proxy functionality (such as caching) and the provided level of protection. From this trade-off results the definition of what constitutes an "authentic request" or "authentic response" and when a request or response is considered confidentiality protected.

This section presents two exemplary choices of trade-offs:

- The first case focuses on a high protection level by tying requests and responses uniquely together and confidentiality protecting as much as possible, at the cost of reduced proxy functionality.

- The second case aims to preserve proxy functionality as much as possible, at the cost of reduced confidentiality protection.

For both cases, this section presents an overview of the functionality and processing rules of the proxy and analyses the required authenticity and confidentiality properties of requests and responses. Due to space constraints, the analysis is limited to the CoAP header, the request and response options shown in Table 1, and the payload.

Requests	Responses
Accept	Content-Format
Content-Format	ETag
ETag	Location-Path
If-Match	Location-Query
If-None-Match	Max-Age
Observe	Observe
Proxy-Scheme	
Proxy-Uri	
Uri-Host	
Uri-Port	
Uri-Path	
Uri-Query	

Table 1: Analysed CoAP Options

Note that, since CoAP was not designed with end-to-end security in mind, a security solution extends the applicability of CoAP beyond its original scope.

2.2.1. Forwarding

In this case we study the functionality of a CoAP forward proxy and assume that caching is disabled. This is applicable to many security critical use cases where a response needs to be securely linked to a unique request from a client and cannot be re-used with another request.

There may be a unique response for each request (see Figure 6) or multiple responses for each request (see Figure 7).

2.2.1.1. Examples

Examples of the need for unique response for each request include alarm status retrieval and actuator command confirmation.

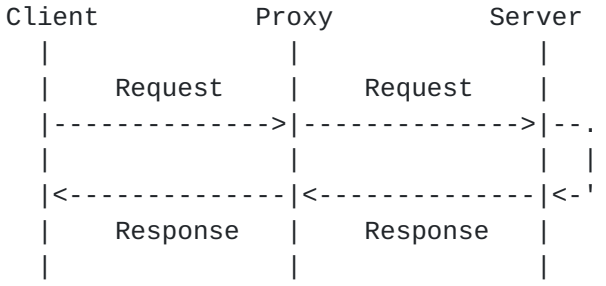

Figure 6: Message Flow with a Unique Response for Each Request

Example: Alarm status retrieval

Figure 6 can be seen as an illustration of a message exchange for a client requesting the alarm status (e.g., GET /alarm_status) from a server. Since the client wants to ensure that the alarm status received is reflecting the current alarm status and not a cached or spoofed response to the same resource, it must be able to verify that the received response is a response to this particular request made by the client. Therefore, the response must be securely linked to the request.

Example: Actuation confirmation

Another example for which Figure 6 serves as illustration is the confirmation of an actuator request. In this case a client, say in an industrial control system, requests a server that a valve should be turned to a certain level, e.g. PUT /valve_42/level with payload "3". In order for the client to correctly evaluate the result of a potential changed valve level, it is important that the client gets a confirmation how the server responded to the requested change, e.g., whether the request was performed or

not. Again, the client wants to ensure that the response is reflecting the result of this particular actuation request made by the client and not a cached or spoofed response. Therefore, the response must be securely linked to the request.

An example of the use of multiple responses for each request is in security critical monitoring scenarios where time synchronization cannot be guaranteed. By avoiding repeated requests from the same client to the same resource, constrained node resources and bandwidth is saved.

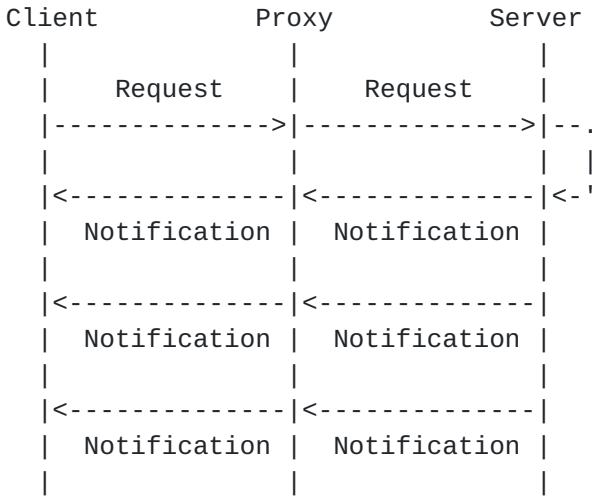

Figure 7: Message Flow of Notifications of Linked to a Unique Request

Example: Secure parameter monitoring

Figure 7 can be seen as an illustration of a message exchange for a client monitoring an important parameter measured by the server, e.g., in a medical or process industry application. The client makes a subscription request for a resource and the server responds with notifications, e.g. providing updates to the parameter on regular time intervals.

The client wants to ensure that the first received notification reflects the current parameter value and that subsequent notifications are timely updates of the initial request. Since notifications may be lost or reordered, the client needs to be able to verify the order of the messages, as sent by the server. By monitoring the received messages and the time they are received, the client can detect missing notifications and take appropriate action.

2.2.1.2. Functional Requirements

FR1.1 The caching functionality MUST be inhibited; the CoAP option Max-Age of the responses SHALL be 0 (see Section 5.7.1 of [RFC7252]).

FR1.2 To limit information leaking about the resource (see Section 2.2.1.5) the Proxy-Uri does not contain Uri-Path or Uri-Query.

2.2.1.3. Processing Rules

In this case, the desired proxy functionality is to forward a translated request to the determined destination. There are two modes of operation for requests: Either using the Proxy-Uri option (PR1.1) or using the Proxy-Scheme option together with the Uri-Host, Uri-Port, Uri-Path and Uri-Query options (PR1.2).

PR1.1 The Proxy-Uri option contains the request URI including request scheme (e.g. "coaps://"); the Proxy-Scheme and Uri-* options are not present.

If the proxy is configured to forward requests to another proxy, then it keeps the Proxy-Uri option; otherwise, it splits the option into its components, adds the corresponding Uri-* options and removes the Proxy-Uri option. Then it makes the request using the request scheme indicated in the Proxy-Uri.

PR1.2 The Proxy-Scheme option and the Uri-* options together contain the request URI; the Proxy-Uri option is not present.

If the proxy is configured to forward requests to another forwarding proxy, then it keeps the Proxy-Scheme and Uri-* options; otherwise, it removes the Proxy-Scheme option. Then it makes the request using the request scheme indicated in the removed Proxy-Scheme option.

PR1.3 Responses are forwarded by the proxy, without any modification.

2.2.1.4. Authenticity

A request is considered authentic by the server (Section 2.1.2.1) if the server can obtain proof for all of the following items:

A1.1 that the proxy acts on behalf of a client;

A1.2 that the following parts of the request originate from the client and have not been altered on the way:

* the CoAP version,
* the request method,
* all options except Proxy-Uri, Proxy-Scheme, Uri-Host, Uri-Port, Uri-Path and Uri-Query, and
* the payload, if any.

A1.3 that the effective request URI originates from the client and has not been altered on the way;

A1.4 that the request has not been received previously;

A1.5 that the request from the client to the proxy was sent recently.

A response is considered authentic by the client (Section 2.1.1.1) if the client can obtain proof for all of the following items:

A1.6 that the following parts of the response originate from the server and have not been altered on the way:

* the CoAP version,
* the response code,
* all options, and
* the payload, if any.

A1.7 that the response corresponds uniquely to the request sent by the client.

A1.8 that the response has not been received previously;

A1.9 that the response from the server to the proxy was sent recently;

A1.10 that the response is in sequence if there are multiple responses.

2.2.1.5. Confidentiality

The following parts of the message are confidentiality protected (Section 2.1.1.5):

- all options except Proxy-Uri, Proxy-Scheme, Uri-Host and Uri-Port;
- the payload, if any.

2.2.2. Caching

In this case we study caching: how a proxy may serve the same cached response to multiple clients requesting the same resource.

The caching functionality protects communication-constrained servers from repeated requests for the same resources, possibly originating from different clients. This saves system resources, bandwidth, and round-trip time.

There may be one response for each request (see Figure 8) or multiple responses for each request (see Figure 9).

2.2.2.1. Examples

The first example is a simple case of caching.

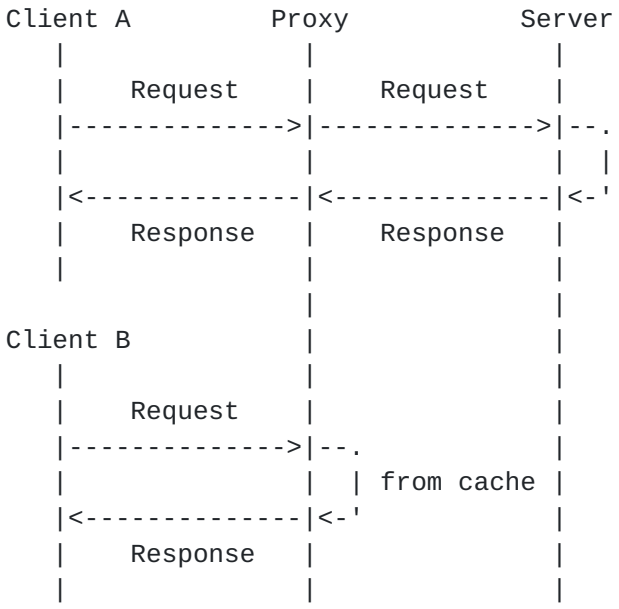

Figure 8: Message Flow for Cached Responses

Example: Caching

In Figure 8, client A requests the proxy to make a certain request to the server and to return the server's response. The proxy services the request by making a request message to the server according to the processing rules. If the server returns a cacheable response, then the proxy stores the response in its cache, performs any necessary translations, and forwards it to the client. Later, client B makes an equivalent request to the proxy that the proxy services by returning the response from its cache. Both client A and B want to verify that the response is valid.

In addition to multiple clients' requests being served by one response, each request may result in multiple responses. The difference compared to Section 2.2.1 is that in this example multiple clients may be served with the same response, further saving server resources.

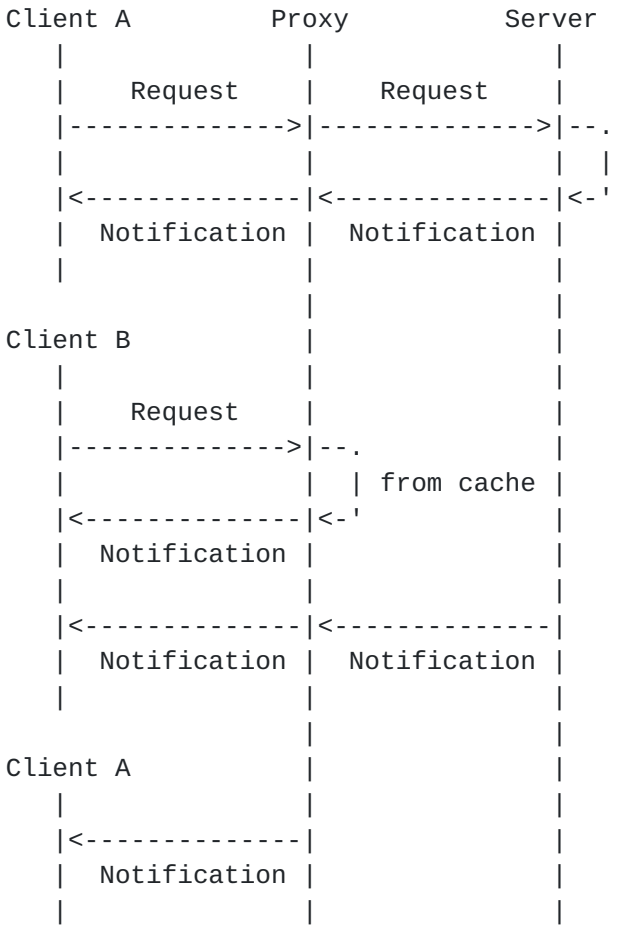

Figure 9: Message Flow for Observe with Multiple Observers

Example: Observe with caching

In Figure 9, the server exposes an observable resource (e.g., the current reading of a temperature sensor). Multiple clients are interested in the current state of the resource and observe it using the CoAP resource observation mechanism [RFC7641]. The goal is to keep the state observed by the clients closely in sync with the actual state of the resource at the server. Another goal is to minimize the burden on the server by moving the task to fan out notifications to multiple clients from the server to the proxy.

2.2.2.2. Functional Requirements

The security solution SHOULD protect requests and responses in a way that a proxy can perform the following tasks:

FR2.1 Storing a cacheable response in a cache. This requires that the proxy is able to calculate the cache-key of the request. Cacheable responses include 2.05 (Content) responses and all error responses.

FR2.2 Returning a fresh response from its cache without contacting the server.

FR2.3 Performing validation of a response cached by the proxy as well as validation of a response cached by the client.

FR2.4 Observing a resource on behalf of one or more clients.

2.2.2.3. Processing Rules

The proxy complies with the forwarding rules PR1.1 - 1.3 (Section 2.2.1.3) and the rules below. The rules below have priority.

PR2.1 If the proxy receives a request where the cache key matches that of a cached fresh response, then the proxy with that response; otherwise, it makes a request towards the server.

PR2.2 The proxy caches and forwards cacheable responses. If there is already a response in the cache with the cache key of the corresponding request, then the old response in the cache is marked as stale.

PR2.3 If the proxy receives a request that contains an ETag option and the proxy has a fresh response with the same cache key and ETag, then the proxy replies to the request with a 2.03 (Valid) response without payload, else it forwards a translated request.

PR2.4 The proxy updates the Max-Age option according to the Max-Age associated with the resource representation it receives, decreasing its value to reflect the time spent in the cache.

PR2.5 If the request contains an Accept option and if there is a fresh response that matches the cache key for the corresponding request except for the Accept option and if the Content-Format of the response matches that of the Accept option, then the proxy forwards the cached response to the requesting client.

2.2.2.4. Authenticity

A request is considered authentic by the server (Section 2.1.2.1) if the server can obtain proof for all of the following items:

A2.1 that the following parts of the request originate from the client and have not been altered on the way:

* the CoAP version,
* the request method,
* all options except ETag, Observe, Proxy-Uri, Proxy-Scheme, Uri-Host, Uri-Port, Uri-Path and Uri-Query, and
* the payload, if any.

A2.2 that the effective request URI originates from the client and has not been altered on the way;

A response is considered authentic by the client (Section 2.1.1.1) if the client can obtain proof for all of the following items:

A2.3 that the following parts of the response originate from the server and have not been altered on the way:

* the CoAP version,
* the response code,
* all options except Max-Age and Observe, and
* the payload, if any.

A2.4 that the response matches the specifications of the request;

A2.5 that the data is fresh (when the response is cacheable);

A2.6 that the response is in sequence (when observing a resource).

2.2.2.5. Confidentiality

No parts of a request are confidentiality protected (Section 2.1.2.5).

A response is considered confidentiality protected (Section 2.1.2.5) if the payload of the response is confidentiality protected.

3. Publish-Subscribe

Much of the concerns about proxies as described previously in this document also applies to other kinds of intermediary nodes. In this section we study brokers in a publish-subscribe setting [I-D.ietf-core-coap-pubsub]. The case of combining brokers and proxies is out of scope for this version of the document.

There are different ways for a pub-sub broker to operate. We consider the following broker operations:

- The broker receives a request for a topic from a subscriber.
- The broker receives a request for a publication to a topic from a publisher and forwards the publication to the subscribers of the topic.

We consider the setting where there is a security association between publisher and subscriber such that the publications can be protected during transfer, see Figure 10.

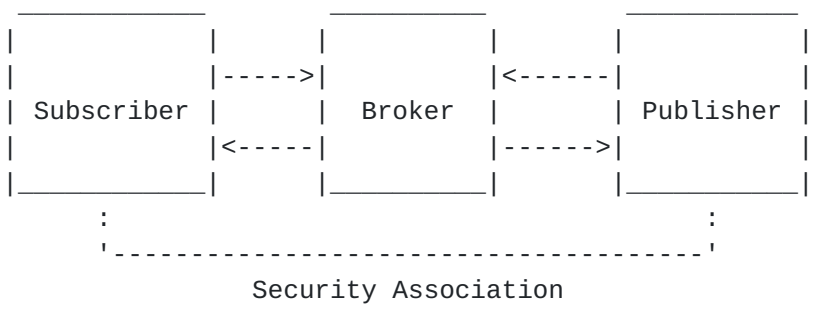

Figure 10: Publisher-to-Subscriber Security

Since there is no security association with the broker, we only consider the subscribe and publish functionality of the broker. Note that the broker needs to read the topic to accomplish this task.

3.1. Threats and Security Requirements

3.1.1. Subscriber-side

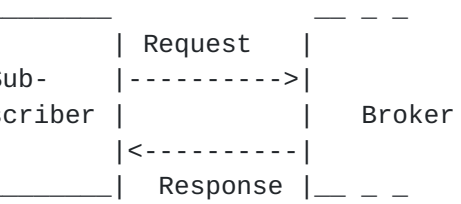

Figure 11: The Subscriber End

The subscriber sends a subscription request to the broker and waits for a response.

From the perspective of the subscriber, there are three possible flows:

- The subscriber receives a response.
  Reasons include:
  - The broker duly processed the request and returns a response based on data it obtained from a publisher.
  - The subscriber made a bad request and the broker returns an error response accordingly (e.g., 4.04 Not Found).
  - The broker encountered an unexpected condition and returns an error response accordingly (e.g., 5.03 Service Unavailable).
  - (Threat 1:) The broker spoofs a response.
  - (Threat 2:) The broker duly processed the request but delays the return of a response.
- The subscriber does not receive a response.
  Reasons include:
  - The subscriber times out too early.
  - (Threat 3:) The broker withholds a response.
- The subscriber receives too many responses.
  Reasons include:
  - (Threat 4:) The broker floods the subscriber with responses.

Furthermore, there are threats related to privacy:

- (Threat 5:) The broker eavesdrops on the data in the request from the subscriber.

- (Threat 6:) The broker measures the size, frequency or distribution of requests from the subscriber.

Note that "topic poisoning" -- the case of storing injected incorrect publications -- is covered from the point of view of the subscriber: it may result in the subscriber receiving a spoofed message, or being flooded, or affect other nodes such that the subscriber times out too early.

3.1.1.1. Threat 1: Spoofing

With one exception (see below), this threat is REQUIRED to be mitigated by the security solution: the subscriber MUST verify that a response is an "authentic publication" before processing it.

The definition of an "authentic publication" depends on the setting (Section 3.2), but usually means that the subscriber can obtain proof for some or all of the following items:

- that the data matches the specifications of the request (such as the topic);
- that the data originates from a publisher that is authorized to publish to the topic;
- that the data has not been altered on the way between publisher and subscriber;
- that the data is fresh (when the data is cacheable);
- that the data is in sequence (when observing a topic).

The proof can, for example, include a message authentication code that the proxy obtains from the origin server and includes in the response or an additional challenge-response roundtrip.

Exception: A CoAP server like the broker is specified to return an error response (such as 4.04 Not Found or 5.03 Service Unavailable) when it encounters an error condition. Since the condition occurs at the broker and not at the publisher, the response will not be an "authentic response" according to the above definition. Thus, a subscriber cannot tell if the broker sends the error response according to specification or if it spoofs the response. This threat is NOT REQUIRED to be mitigated by the security solution.

3.1.1.2. Threat 2: Delaying

This threat is NOT REQUIRED to be mitigated by the security solution.

3.1.1.3. Threat 3: Withholding

This threat is NOT REQUIRED to be mitigated by the security solution, since a subscriber cannot tell if the broker does not send a response because it is hasn't received a publication from the publisher yet or if it intentionally withholds the response.

3.1.1.4. Threat 4: Flooding

A CoAP client like the subscriber is specified to reject any response that it does not expect. This can happen before the subscriber verifies if the response is authentic. Therefore, a flood of responses is primarily a threat to the system resources of the client, in particular to its energy. This threat is NOT REQUIRED to be mitigated by the security solution, but a subscriber SHOULD generally defend against flooding attacks.

3.1.1.5. Threat 5: Eavesdropping

This threat is NOT REQUIRED to be mitigated: The broker needs to read all parts of the request from the subscriber to accomplish its task.

It is RECOMMENDED that applications analyse the risks associated with application information leaking from the messages flow and assess the feasibility to protect against various threats, e.g., by obfuscating topic content.

3.1.1.6. Threat 6: Traffic Analysis

This threat is NOT REQUIRED to be mitigated by the security solution.

It is RECOMMENDED that applications analyse the risks associated with application information leaking from the messages flow and assess the feasibility to protect against various threats, e.g., by obfuscating parameters transported in plain text, aligning message flow and traffic between the different cases, adding padding so different messages become indistinguishable, etc.

3.1.2. Publisher-side

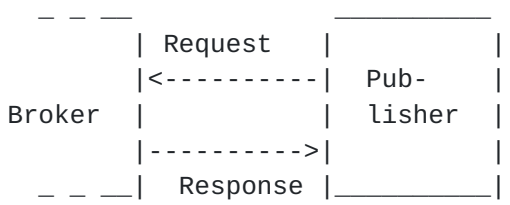

Figure 12: The Publisher End

The publisher sends a publication request to the broker and waits for a response.

The threat of the broker eavesdropping on the data in the publication request is REQUIRED to be mitigated by the security solution: publishers MUST confidentiality protect the data in the requests they send. This excludes parts that the broker needs to read to perform its job, e.g., the topic.

The threat of the broker measuring the size, frequency or distribution of publication requests is NOT REQUIRED to be mitigated by the security solution; see Section 3.1.1.6.

The broker is in full control of the response and may therefore arbitrarily spoof, delay, or withhold it. This threat is NOT REQUIRED to be mitigated. For example, a proof that the broker has notified all subscribers is NOT REQUIRED.

3.2. Solutions

3.2.1. Brokering

In this case we study brokering: how a broker may serve the same publication to multiple subscribers observing the same topic.

The brokering functionality protects communication-constrained publishers from repeated requests for the same resources, possibly originating from different subscribers. This saves system resources, bandwidth, and round-trip time.

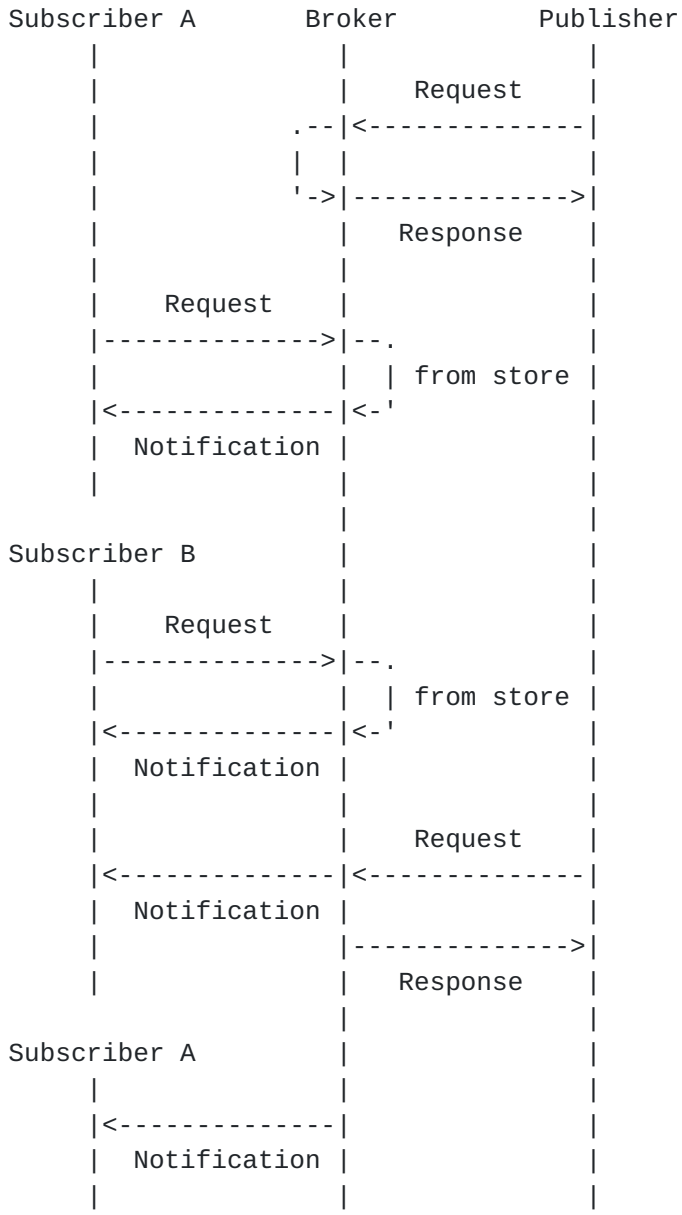

Figure 13: Message Flow for Publish Subscribe

Example

In Figure 13, the publisher publishes to a topic (e.g., the current reading of a temperature sensor). Multiple subscribers are interested in the current state of the topic and observe the topic as specified in [I-D.ietf-core-coap-pubsub]. The goal is to keep the state observed by the subscribers closely in sync with the actual state of the resource at the publisher. Another goal is to minimize the burden on the publisher by moving the task to fan out notifications to multiple subscribers from the publisher to the broker.

3.2.1.1. Functional Requirements

The security solution SHOULD protect subscription and publication requests in a way that a broker can perform the following tasks:

FR3.1 Storing publications. This requires that the broker is able to read the topic of the request.

FR3.2 Returning a stored publication without contacting the publisher.

3.2.1.2. Processing Rules

The broker complies with the following rules:

PR3.1 If the broker receives a request where the topic matches that of a cached publication, then the broker responds with that publication.

PR3.2 The broker caches and forwards publication notifications.

3.2.1.3. Authenticity

A publication is considered authentic by the subscriber if the subscriber can obtain proof for all all of the following items:

A3.1 that the payload is associated to the topic;

A3.2 that the payload has not been altered since published;

A3.3 that the publication is in sequence.

3.2.1.4. Confidentiality

The payload of a publication request is confidentiality protected.

4. Security Considerations

This document is about security; as such, there are no additional security considerations.

5. IANA Considerations

This document includes no request to IANA.

6. References

6.1. 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>.

[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>.

6.2. Informative References

[I-D.ietf-core-coap-pubsub] Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-02 (work in progress), July 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.

[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>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <http://www.rfc-editor.org/info/rfc7228>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <http://www.rfc-editor.org/info/rfc8152>.

Acknowledgments

Thanks to Ari Keranen, John Mattsson, Jim Schaad and Ludwig Seitz for helpful comments and discussions that have shaped the document.

Authors' Addresses

Goeran Selander
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen 28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org