(Un)trusted Intermediaries in CoAP

Thomas Fossati[*], Angelo P. Castellani[†], Salvatore Loreto[‡]

February 16, 2012

1 Problem Statement

End-to-end secure communication as specified in CoAP, either DTLS or IPsec, is feasible only if direct communication is possible with the peer endpoint. If one or more intermediaries are involved in the process, either upper-layer object security must be adopted or *total* trust on the intermediate proxies is required to preserve the security of the communication.

Based on the current specification [1], the only feasible option to build a secure tunnel between the endpoints is to create a chain of independent trust links equal to the number of traversed proxies, hence achieving the end-to-end trust transitively. This is equivalent to stating that the end node has to eventually trust all the involved intermediaries in between the endpoints.

Moreover, different trust links along the path from end to end may show different security modes. In fact, even if a client endpoint uses DTLS with the intermediary, the intermediary itself can use IPsec or even NoSec mode on the following link. This may be typical in case the intermediary wants to enable secure communication features to constrained devices that natively do not support them, e.g. very limited class-1 devices missing the hardware resources required. On the other hand, this approach may introduce vulnerabilities to the whole path in case the security policy enforced on a given link is not adequate.

It is also worth noting that [1] is not clear about how a `coaps` request to a proxy works (i.e. whether to use the client or the proxy credentials to create the DTLS connection), especially in case the proxy forwards the request to another proxy instead of sending it to the server specified by the absolute-URI.

Which mechanisms can be envisaged in order to allow secure and transparent end-to-end communication using the CoAP protocol ? Is it possible to remove, or at least reduce, the amount of trust that end nodes in a CoAP network have to put in the proxy nodes ? How and at what price can this be attained ?

This short paper tries to provoke some thinking about this seemingly open question, and to provide a tentative answer in terms of what is currently available or missing in current CoAP specification, what could be added to circumvent the issue, and what the engendered ramifications are.

[*]KoanLogic, tho@koanlogic.com.
[†]Consorzio Ferrara Ricerche (CFR), DEI – University of Padova, castellani@dei.unipd.it.
[‡]Ericsson, salvatore.loreto@ericsson.com.

1.1 Secure HTTP-CoAP mapping

The problem stated in Section 1 applies especially if one of the endpoints is using HTTP, and thus an HC proxy [2] is involved in the path, since TLS and DTLS can't be seamlessly bridged.

The HC proxy must have a security policy mapping in place to translate from the TLS security mode of operation used in HTTPS to any security mode available on the path to the requested CoAP resource. Depending on the deployment, security on the CoAP path might not be required, or could be guaranteed by means of other mechanisms, e.g. layer 2 security; in this context CoAP communication could even be implemented using the NoSec security mode.

2 Introducing the Untrusted Proxy

Though at least one documented MitM attack on (D)TLS exist [3], it is believed that current (D)TLS specification and its most widespread implementations (i.e. at least OpenSSL and GNUTLS) are immune to such risk, which would abrogate – *by definition* – any chance to reduce the amount of trust that depending nodes must put onto intermediaries.

Given this essential premise, the only possibility currently left to the constrained node to do end-to-end security in the depicted setting, is to use some kind of so called *object security*, i.e. protect the messaging contents, and communicating endpoints identities, by means of a tamper proof envelope such as [4] or [5]. This could result in more chubby and complex implementation, and be easily vulnerable to replay attacks unless well defined.

At present, what seems to be missing in CoAP to attain end-to-end security while minimizing the trust on proxy nodes, is some sort of *bypass* facility by which end nodes could communicate traversing the chain of intermediaries transparently, e.g. one that mimics the semantics of the `CONNECT` method in HTTP.

Provisioning CoAP with such facility would allow direct communication with trusted nodes deployed on an untrusted network even without direct connectivity to them, by relying on a CoAP intermediary to provide such "restrict" communication facility, blindly forwarding packets until the associated security context is drained.

2.1 Ramifications

The implications of such increase in power are rather interesting, and need to be weighted carefully when practically taking into consideration the introduction of a `CONNECT`-like method in CoAP.

Security Policy Bypass. By allowing an end node to use a `CONNECT`-like facility, a security exception is introduced natively into the architecture, which quite reverses the established trust relationships. Now the proxy has to be confident in the end node, which is permitted, in principle, to bypass the security policy implicitly set by the proxy role and topological position. Hence, the availability of such facility definitely needs to be constrained by white/black-listing of intended sources and targets.

Bootleg PCP. Lacking an access control mechanism based on *strong* authentication primitives (e.g. using one based on simple network locators), the added functionality could be trivially abused in more than one disruptive way, e.g. to build a fraudulent Port Control Protocol. Note that devices willing to do end-to-end security already have the needed computational resources to exchange strong identities with the proxy node, so this should be affordable at no additional cost.

Cache Elusion. Another side-effect is the elusion of the caching function. Note that since resources are requested from the `coaps` scheme, the cache is weakened in any case since "secure" resources are not (publicly) cacheable by their own nature. But this is stronger since, in this case, the cache is *fully* eluded, blowing away one of the fundamental functions of the proxy.

References

[1] *"Constrained Application Protocol (CoAP)"*, Z. Shelby, K. Hartke, C. Bormann, B. Frank, November 2011.

[2] *"Best practices for HTTP-CoAP mapping implementation"*, Angelo P. Castellani, Salvatore Loreto, Akbar Rahman, Thomas Fossati, and Esko Dijk, October 2011.

[3] *"Renegotiating TLS"*, Marsh Ray, Steve Dispensa, November 2009.

[4] *"Cryptographic Message Syntax"*, R. Housley, STD 70, RFC 5652, September 2009.

[5] *"Javascript Object Signing and Encryption Charter"*