Workgroup: Network Working Group
Internet-Draft:
draft-mattsson-core-coap-attacks-latest
Published: 4 February 2022
Intended Status: Informational
Expires: 8 August 2022
Authors: J. Preuß Mattsson J. Fornehed G. Selander
Ericsson Ericsson Ericsson
F. Palombini C. Amsüss
Ericsson Energy Harvesting Solutions

Attacks on the Constrained Application Protocol (CoAP)

Abstract

Being able to securely read information from sensors, to securely control actuators, and to not enable distributed denial-of-service attacks are essential in a world of connected and networking things interacting with the physical world. This document summarizes a number of known attacks on CoAP and show that just using CoAP with a security protocol like DTLS, TLS, or OSCORE is not enough for secure operation. Several of the discussed attacks can be mitigated with the solutions in draft-ietf-core-echo-request-tag.

Table of Contents

1. Introduction

Being able to securely read information from sensors and to securely control actuators are essential in a world of connected and networking things interacting with the physical world. One protocol used to interact with sensors and actuators is the Constrained Application Protocol (CoAP) [RFC7252]. Any Internet-of-Things (IoT) deployment valuing security and privacy would use a security protocol such as DTLS [I-D.ietf-tls-dtls13], TLS [RFC8446], or OSCORE [RFC8613] to protect CoAP, where the choice of security protocol depends on the transport protocol and the presence of intermediaries. The use of CoAP over UDP and DTLS is specified in [RFC7252] and the use of CoAP over TCP and TLS is specified in [RFC8323]. OSCORE protects CoAP end-to-end with the use of COSE [RFC8152] and the CoAP Object-Security option [RFC8613], and can therefore be used over any transport.

The Constrained Application Protocol (CoAP) [RFC7252] was designed with the assumption that security could be provided on a separate layer, in particular by using DTLS [RFC6347]. The four properties traditionally provided by security protocols are:

*Data confidentiality

*Data origin authentication

*Data integrity checking

*Replay protection

In this document we show that protecting CoAP with a security protocol on another layer is not nearly enough to securely control actuators (and in many cases sensors) and that secure operation often demands far more than the four properties traditionally provided by security protocols. We describe several serious attacks any on-path attacker (i.e., not only "trusted intermediaries") can do and discusses tougher requirements and mechanisms to mitigate the attacks. In general, secure operation of actuators also requires the three properties:

*Data-to-data binding

*Data-to-space binding

*Data-to-time binding

"Data-to-data binding" is e.g., binding of responses to a request or binding of data fragments to each other. "Data-to-space binding" is the binding of data to an absolute or relative point in space (i.e., a location) and may in the relative case be referred to as proximity. "Data-to-time binding" is the binding of data to an absolute or relative point in time and may in the relative case be referred to as freshness. The two last properties may be bundled together as "Data-to-spacetime binding".

Freshness is a measure of when a message was sent on a timescale of the recipient. A client or server that receives a message can either verify that the message is fresh or determine that it cannot be verified that the message is fresh. What is considered fresh is application dependent. Freshness is completely different from replay protection, but most replay protection mechanism use a sequence number. Assuming the client is well-behaving, such a sequence number that can be used by the server as a relative measure of when a message was sent on a timescale of the sender. Replay protection is mandatory in TLS and OSCORE and optional in DTLS. DTLS and TLS use sequence numbers for both requests and responses. In TLS the sequence numbers are implicit and not sent in the record. OSCORE use sequence numbers for requests and some responses. Most OSCORE responses are bound to the request and therefore, enable the client to determine if the response is fresh or not.

The request delay attack (valid for DTLS, TLS, and OSCORE and described in Section 2.2) lets an attacker control an actuator at a much later time than the client anticipated. The response delay and mismatch attack (valid for DTLS and TLS and described in Section 2.3) lets an attacker respond to a client with a response meant for

an older request. The request fragment rearrangement attack (valid for DTLS, TLS, and OSCORE and described in Section 2.4) lets an attacker cause unauthorized operations to be performed on the server, and responses to unauthorized operations to be mistaken for responses to authorized operations.

The goal with this document is motivating generic and protocol-specific recommendations on the usage of CoAP. Mechanisms mitigating some of the attacks discussed in this document can be found in [I-D.ietf-core-echo-request-tag]. This document is a companion document to [I-D.ietf-core-echo-request-tag] giving more information on the attacks motivating the mechanisms.

2. Attacks on CoAP

Internet-of-Things (IoT) deployments valuing security and privacy, need to use a security protocol such as DTLS, TLS, or OSCORE to protect CoAP. This is especially true for deployments of actuators where attacks often (but not always) have serious consequences. The attacks described in this section are made under the assumption that CoAP is already protected with a security protocol such as DTLS, TLS, or OSCORE, as an attacker otherwise can easily forge false requests and responses.

2.1. The Block Attack

An on-path attacker can block the delivery of any number of requests or responses. The attack can also be performed by an attacker jamming the lower layer radio protocol. This is true even if a security protocol like DTLS, TLS, or OSCORE is used. Encryption makes selective blocking of messages harder, but not impossible or even infeasible. With DTLS and TLS, proxies can read the complete CoAP message, and with OSCORE, the CoAP header and several CoAP options are not encrypted. In all three security protocols, the IP-addresses, ports, and CoAP message lengths are available to all on-path attackers, which may be enough to determine the server, resource, and command. The block attack is illustrated in Figures 1 and 2.

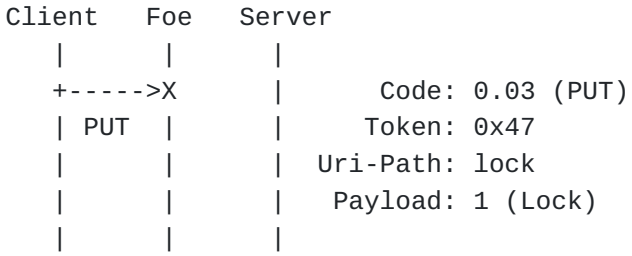

Figure 1: Blocking a request

Where 'X' means the attacker is blocking delivery of the message.

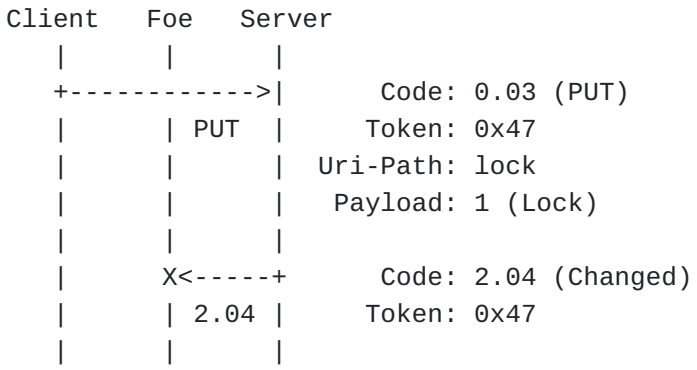

Figure 2: Blocking a response

While blocking requests to, or responses from, a sensor is just a denial-of-service attack, blocking a request to, or a response from, an actuator results in the client losing information about the server's status. If the actuator e.g., is a lock (door, car, etc.), the attack results in the client not knowing (except by using out-of-band information) whether the lock is unlocked or locked, just like the observer in the famous Schrödinger’s cat thought experiment. Due to the nature of the attack, the client cannot distinguish the attack from connectivity problems, offline servers, or unexpected behavior from middle boxes such as NATs and firewalls.

Remedy: Any IoT deployment of actuators where synchronized state is important need to use confirmable messages and the client need to take appropriate actions when a response is not received and it therefore loses information about the server's status.

2.2. The Request Delay Attack

An on-path attacker may not only block packets, but can also delay the delivery of any packet (request or response) by a chosen amount of time. If CoAP is used over a reliable and ordered transport such as TCP with TLS or OSCORE (with TLS-like sequence number handling), no messages can be delivered before the delayed message. If CoAP is used over an unreliable and unordered transport such as UDP with DTLS or OSCORE, other messages can be delivered before the delayed message as long as the delayed packet is delivered inside the replay window. When CoAP is used over UDP, both DTLS and OSCORE allow out-of-order delivery and uses sequence numbers together with a replay window to protect against replay attacks against requests. The replay window has a default length of 64 in DTLS and 32 in OSCORE. The attacker can influence the replay window state by blocking and delaying packets. By first delaying a request, and then later, after delivery, blocking the response to the request, the client is not

made aware of the delayed delivery except by the missing response. In general, the server has no way of knowing that the request was delayed and will therefore happily process the request. Note that delays can also happen for other reasons than a malicious attacker.

If some wireless low-level protocol is used, the attack can also be performed by the attacker simultaneously recording what the client transmits while at the same time jamming the server. The request delay attack is illustrated in Figure 3.

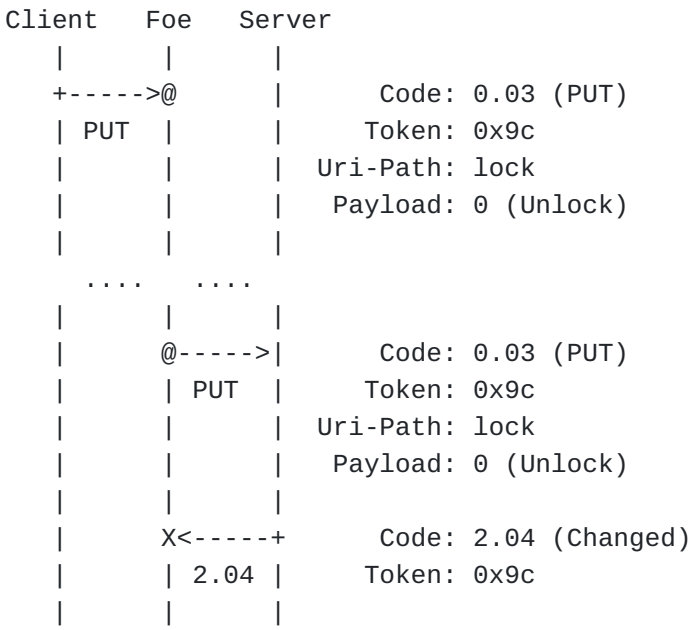

Figure 3: Delaying a request

Where '@' means the attacker is storing and later forwarding the message (@ may alternatively be seen as a wormhole connecting two points in time).

While an attacker delaying a request to a sensor is often not a security problem, an attacker delaying a request to an actuator performing an action is often a serious problem. A request to an actuator (for example a request to unlock a lock) is often only meant to be valid for a short time frame, and if the request does not reach the actuator during this short timeframe, the request should not be fulfilled. In the unlock example, if the client does not get any response and does not physically see the lock opening, the user is likely to walk away, calling the locksmith (or the IT-support).

If a non-zero replay window is used (the default when CoAP is used over UDP), the attacker can let the client interact with the actuator before delivering the delayed request to the server

(illustrated in Figure 4). In the lock example, the attacker may store the first "unlock" request for later use. The client will likely resend the request with the same token. If DTLS is used, the resent packet will have a different sequence number and the attacker can forward it. If OSCORE is used, resent packets will have the same sequence number and the attacker must block them all until the client sends a new message with a new sequence number (not shown in Figure 4). After a while when the client has locked the door again, the attacker can deliver the delayed "unlock" message to the door, a very serious attack.

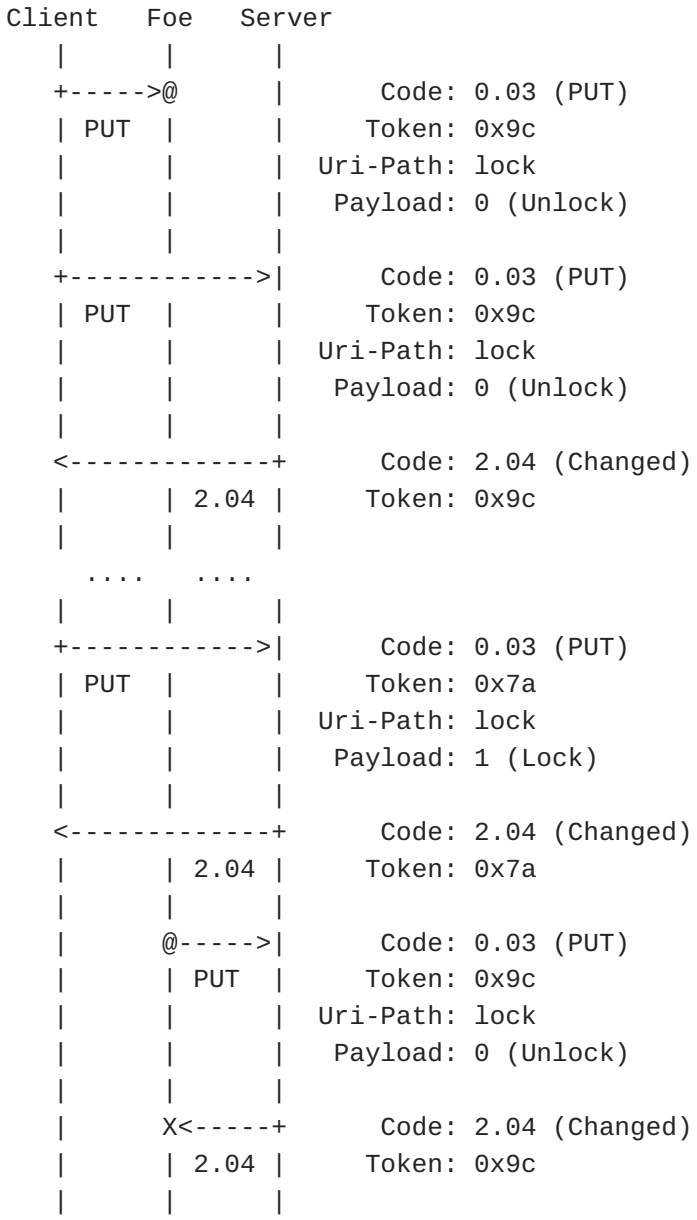

Figure 4: Delaying request with reordering

While the second attack (Figure 4) can be mitigated by using a replay window of length zero, the first attack (Figure 3) cannot. A solution must enable the server to verify that the request was received within a certain time frame after it was sent or enable the server to securely determine an absolute point in time when the request is to be executed. This can be accomplished with either a challenge-response pattern, by exchanging timestamps between client and server, or by only allowing requests a short period after client authentication.

Requiring a fresh client authentication (such as a new TLS/DTLS handshake or an EDHOC key exchange [I-D.ietf-lake-edhoc]) mitigates the problem, but requires larger messages and more processing than a dedicated solution. Security solutions based on exchanging timestamps require exactly synchronized time between client and server, and this may be hard to control with complications such as time zones and daylight saving. Wall clock time is not monotonic, may reveal that the endpoints will accept expired certificates, or reveal the endpoint's location. Use of non-monotonic clocks is problematic as the server will accept requests if the clock is moved backward and reject requests if the clock is moved forward. Even if the clocks are synchronized at one point in time, they may easily get out-of-sync and an attacker may even be able to affect the client or the server time in various ways such as setting up a fake NTP server, broadcasting false time signals to radio-controlled clocks, or exposing one of them to a strong gravity field. As soon as client falsely believes it is time synchronized with the server, delay attacks are possible. A challenge response mechanism where the server does not need to synchronize its time with the client is easier to analyze but require more roundtrips. The challenges, responses, and timestamps may be sent in a CoAP option or in the CoAP payload.

Remedy: Any IoT deployment of actuators where freshness is important should use the mechanisms specified in [I-D.ietf-core-echo-request-tag] unless another application specific challenge-response or timestamp mechanism is used.

2.3. The Response Delay and Mismatch Attack

The following attack can be performed if CoAP is protected by a security protocol where the response is not bound to the request in any way except by the CoAP token. This would include most general security protocols, such as DTLS, TLS, and IPsec, but not OSCORE. CoAP [RFC7252] uses a client generated token that the server echoes to match responses to request, but does not give any guidelines for the use of token with DTLS and TLS, except that the tokens currently "in use" SHOULD (not SHALL) be unique. In HTTPS, this type of binding is always assured by the ordered and reliable delivery, as

well as mandating that the server sends responses in the same order that the requests were received.

The attacker performs the attack by delaying delivery of a response until the client sends a request with the same token, the response will be accepted by the client as a valid response to the later request. If CoAP is used over a reliable and ordered transport such as TCP with TLS, no messages can be delivered before the delayed message. If CoAP is used over an unreliable and unordered transport such as UDP with DTLS, other messages can be delivered before the delayed message as long as the delayed packet is delivered inside the replay window. Note that mismatches can also happen for other reasons than a malicious attacker, e.g., delayed delivery or a server sending notifications to an uninterested client.

The attack can be performed by an attacker on the wire, or an attacker simultaneously recording what the server transmits while at the same time jamming the client. As (D)TLS encrypts the Token, the attacker needs to predict when the Token is resused. How hard that is depends on the CoAP library, but some implementations are known to omit the Token as much as possible and others lets the application chose the Token. If the response is a "piggybacked response", the client may additionally check the Message ID and drop it on mismatch. That doesn't make the attack impossible, but lowers the probability.

The response delay and mismatch attack is illustrated in Figure 5.

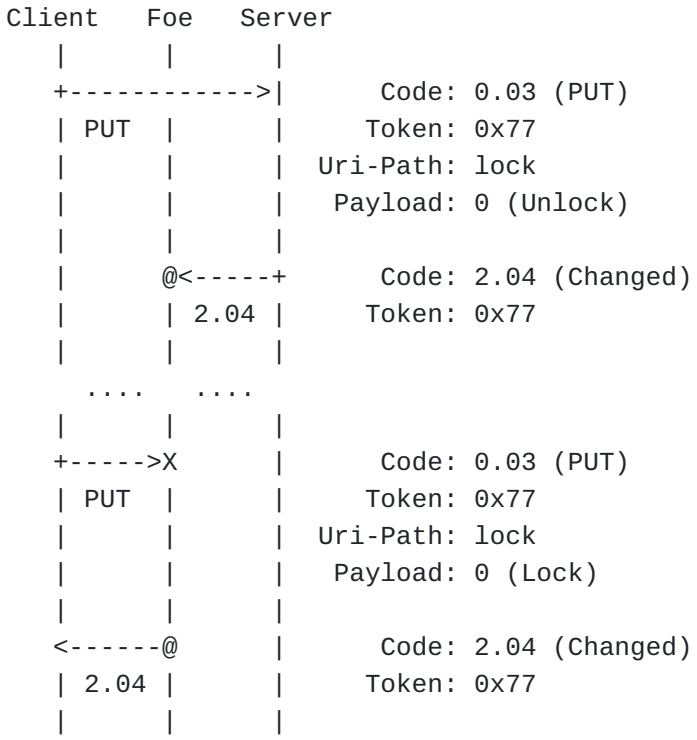

Figure 5: Delaying and mismatching response to PUT

If we once again take a lock as an example, the security consequences may be severe as the client receives a response message likely to be interpreted as confirmation of a locked door, while the received response message is in fact confirming an earlier unlock of the door. As the client is likely to leave the (believed to be locked) door unattended, the attacker may enter the home, enterprise, or car protected by the lock.

The same attack may be performed on sensors. As illustrated in Figure 6, an attacker may convince the client that the lock is locked, when it in fact is not. The "Unlock" request may be also be sent by another client authorized to control the lock.

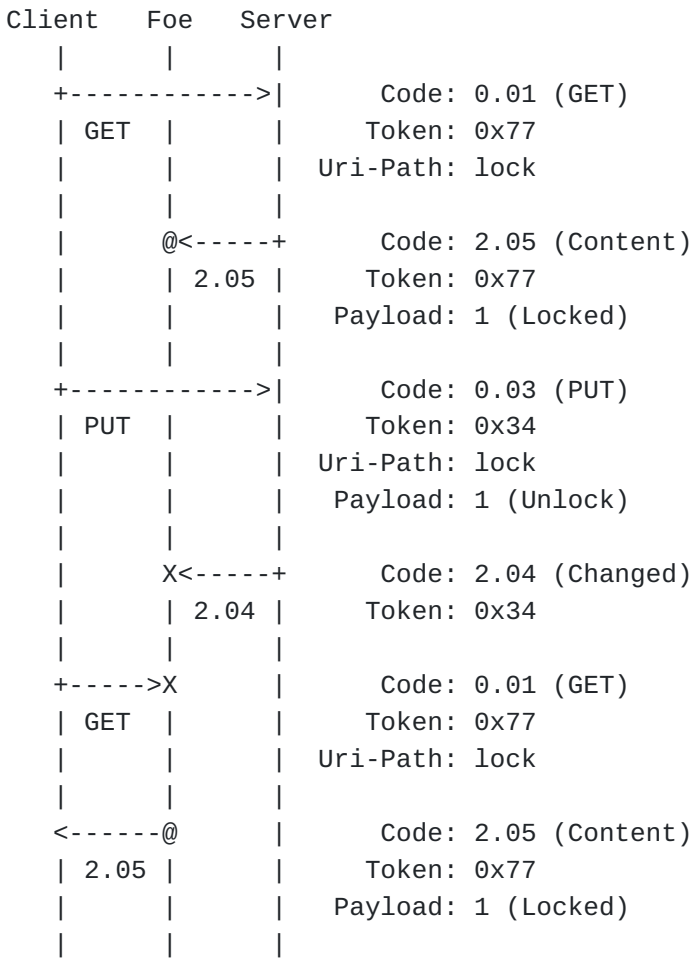

Figure 6: Delaying and mismatching response to GET

As illustrated in Figure 7, an attacker may even mix responses from different resources as long as the two resources share the same (D)TLS connection on some part of the path towards the client. This can happen if the resources are located behind a common gateway, or

are served by the same CoAP proxy. An on-path attacker (not necessarily a (D)TLS endpoint such as a proxy) may e.g., deceive a client that the living room is on fire by responding with an earlier delayed response from the oven (temperatures in degree Celsius).

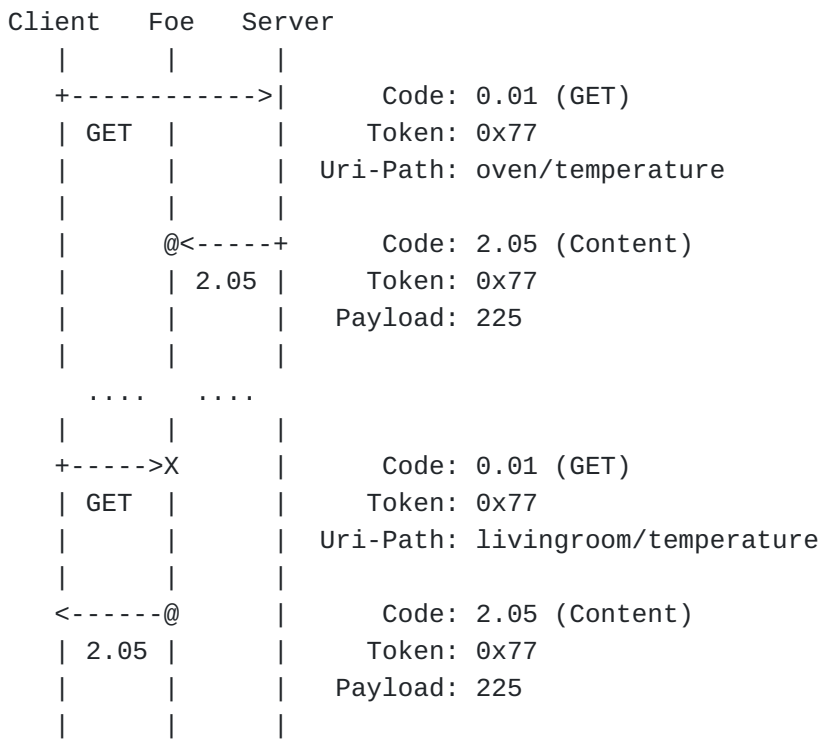

Figure 7: Delaying and mismatching response from other resource

Remedy: Section 4.2 of [I-D.ietf-core-echo-request-tag] formally updates the client token processing for CoAP [RFC7252]. Following this updated processing mitigates the attack.

2.4. The Request Fragment Rearrangement Attack

These attack scenarios show that the Request Delay and Block Attacks can be used against block-wise transfers to cause unauthorized operations to be performed on the server, and responses to unauthorized operations to be mistaken for responses to authorized operations. The combination of these attacks is described as a separate attack because it makes the Request Delay Attack relevant to systems that are otherwise not time-dependent, which means that they could disregard the Request Delay Attack.

This attack works even if the individual request/response pairs are encrypted, authenticated and protected against the Response Delay and Mismatch Attack, provided the attacker is on the network path and can correctly guess which operations the respective packages belong to.

The attacks can be performed on any security protocol where the attacker can delay the delivery of a message unnoticed. This incluses DTLS, IPsec, and most OSCORE configurations. The attacks does not work on TCP with TLS or OSCORE (with TLS-like sequence number handling) as in these cases no messages can be delivered before the delayed message.

2.4.1. Completing an Operation with an Earlier Final Block

In this scenario (illustrated in Figure 8), blocks from two operations on a POST-accepting resource are combined to make the server execute an action that was not intended by the authorized client. This works only if the client attempts a second operation after the first operation failed (due to what the attacker made appear like a network outage) within the replay window. The client does not receive a confirmation on the second operation either, but, by the time the client acts on it, the server has already executed the unauthorized action.

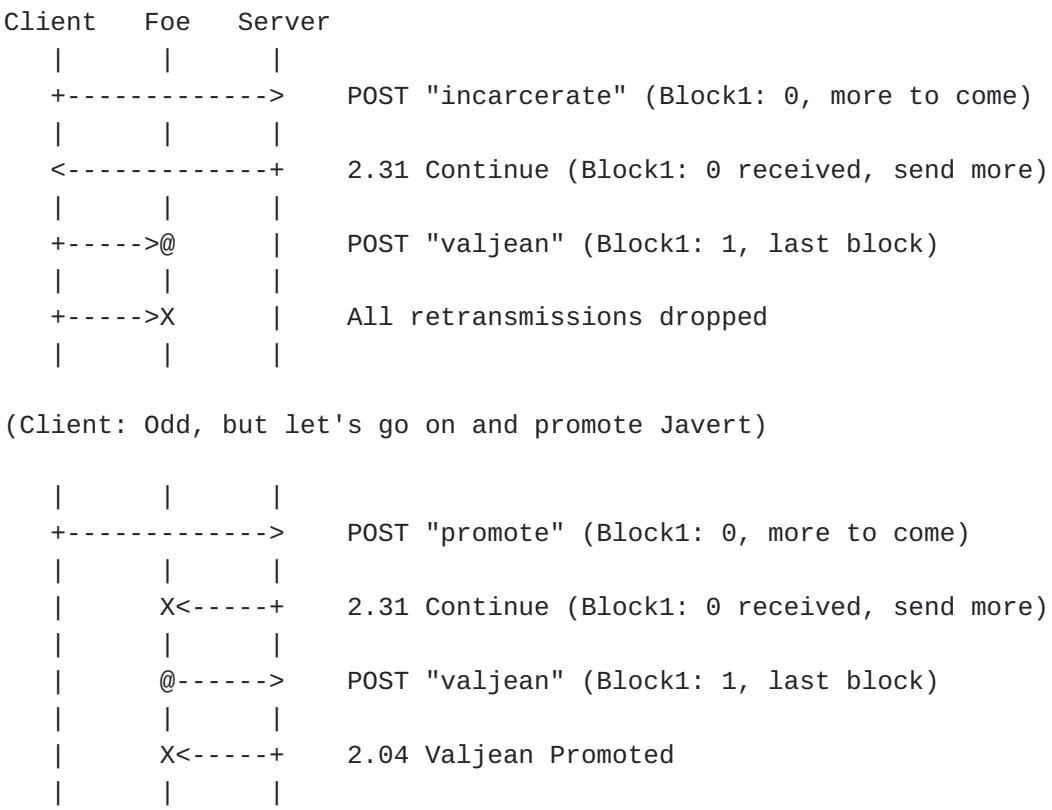

Figure 8: Completing an operation with an earlier final block

Remedy: If a client starts new block-wise operations on a security context that has lost packets, it needs to label the fragments in such a way that the server will not mix them up.

A mechanism to that effect is described as Request-Tag [I-D.ietf-core-echo-request-tag]. Had it been in place in the example and used for body integrity protection, the client would have set the Request-Tag option in the "promote" request. Depending on the server's capabilities and setup, either of four outcomes could have occurred:

1. The server could have processed the reinjected POST "valjean" as belonging to the original "incarcerate" block; that's the expected case when the server can handle simultaneous block transfers.

2. The server could respond 5.03 Service Unavailable, including a Max-Age option indicating how long it prefers not to take any requests that force it to overwrite the state kept for the "incarcerate" request.

3. The server could decide to drop the state kept for the "incarcerate" request's state, and process the "promote" request. The reinjected POST "valjean" will then fail with 4.08 Request Entity incomplete, indicating that the server does not have the start of the operation any more.

2.4.2. Injecting a Withheld First Block

If the first block of a request is withheld by the attacker for later use, it can be used to have the server process a different request body than intended by the client. Unlike in the previous scenario, it will return a response based on that body to the client.

Again, a first operation (that would go like "Girl stole apple. What shall we do with her?" - "Set her free.") is aborted by the proxy, and a part of that operation is later used in a different operation to prime the server for responding leniently to another operation that would originally have been "Evil Queen poisoned apple. What shall we do with her?" - "Lock her up.". The attack is illustrated in Figure 9.

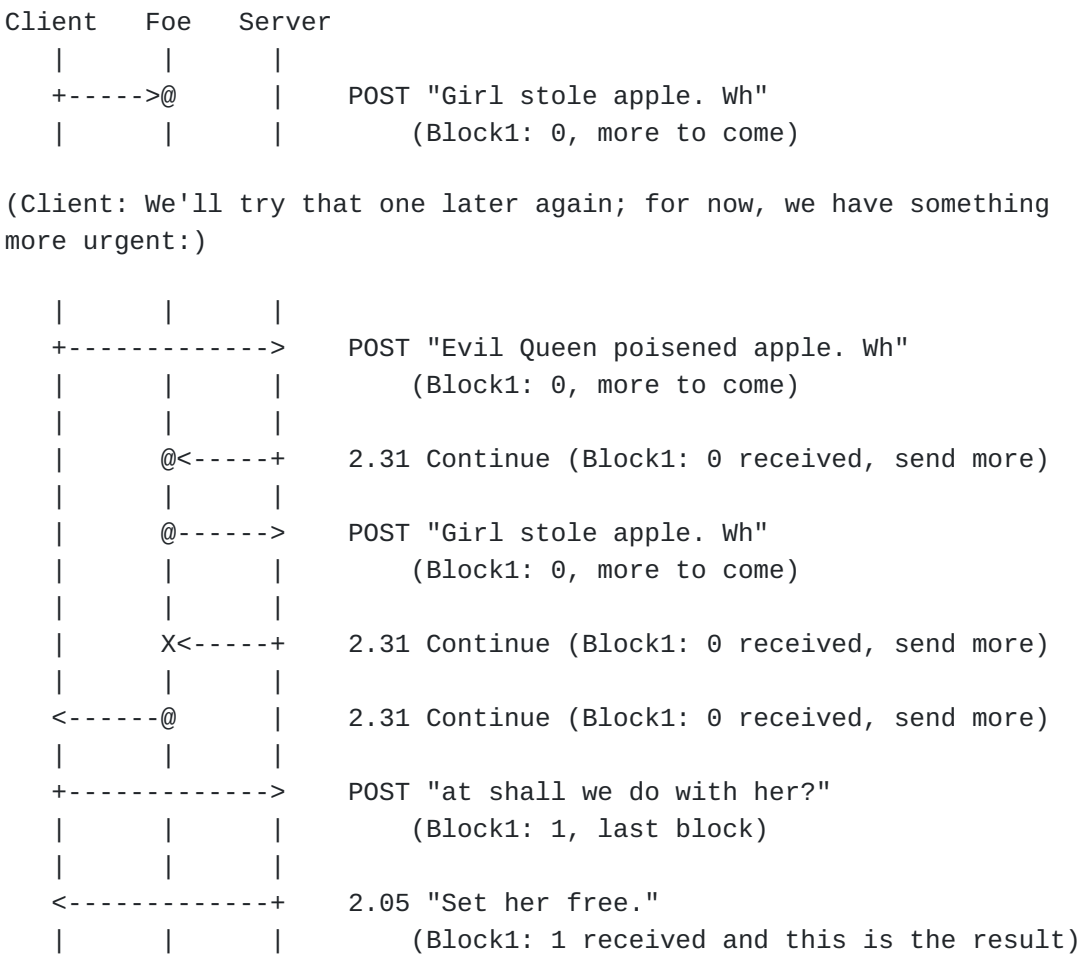

Figure 9: Injecting a withheld first block

The remedy described in Section 2.4.1 works also for this case. Note that merely requiring that blocks of an operation should have incrementing sequence numbers would be insufficient to remedy this attack.

2.4.3. Attack difficulty

The success of any fragment rearrangement attack has multiple prerequisites:

* A client sends different block-wise requests that are only distinguished by their content.

  This is generally rare in typical CoRE applications, but can happen when the bodies of FETCH requests exceed the fragmentation threshold, or when SOAP patterns are emulated.

* A client starts later block-wise operations after an earlier one has failed.

This happens regularly as a consequence of operating in a low-power and lossy network: Losses can cause failed operation (especially when the network is unavailable for time exceeding the "few expected round-trips" they may be limited to per [RFC7959]), and the cost of reestablishing a security context.

* The attacker needs to be able to determine which packets contain which fragments.

  This can be achieved by an on-path attacker by observing request timing, or simply by observing request sizes in the case when a body is split into precisely two blocks.

It is *not* a prerequisite that the resulting misassembled request body is syntactically correct: As the server erroneously expects the body to be integrity protected from an authorized source, it might be using a parser not suitable for untrusted input. Such a parser might crash the server in extreme cases, but might also produce a valid but incorrect response to the request the client associates the response with. Note that many constrained applications aim to minimize traffic and thus employ compact data formats; that compactness leaves little room for syntactically invalid messages.

The attack is easier if the attacker has control over the request bodies (which would be the case when a trusted proxy validates the attacker's authorization to perform two given requests, and an attack on the path between the proxy and the server recombines the blocks to a semantically different request). Attacks of that shape can easily result in reassembled bodies chosen by the attacker, but no services are currently known that operate in this way.

Summarizing, it is unlikely that an attacker can perform any of the fragment rearrangement attacks on any given system - but given the diversity of applications built on CoAP, it is easily to imagine that single applications would be vulnerable. As block-wise transfer is a basic feature of CoAP and its details are sometimes hidden behind abstractions or proxies, application authors can not be expected to design their applications with these attacks in mind, and mitigation on the protocol level is prudent.

2.5. The Relay Attack

Yet another type of attack can be performed in deployments where actuator actions are triggered automatically based on proximity and without any user interaction, e.g., a car (the client) constantly polling for the car key (the server) and unlocking both doors and engine as soon as the car key responds. An attacker (or pair of attackers) may simply relay the CoAP messages out-of-band, using for examples some other radio technology. By doing this, the actuator

(i.e., the car) believes that the client is close by and performs actions based on that false assumption. The attack is illustrated in Figure 10. In this example the car is using an application specific challenge-response mechanism transferred as CoAP payloads.

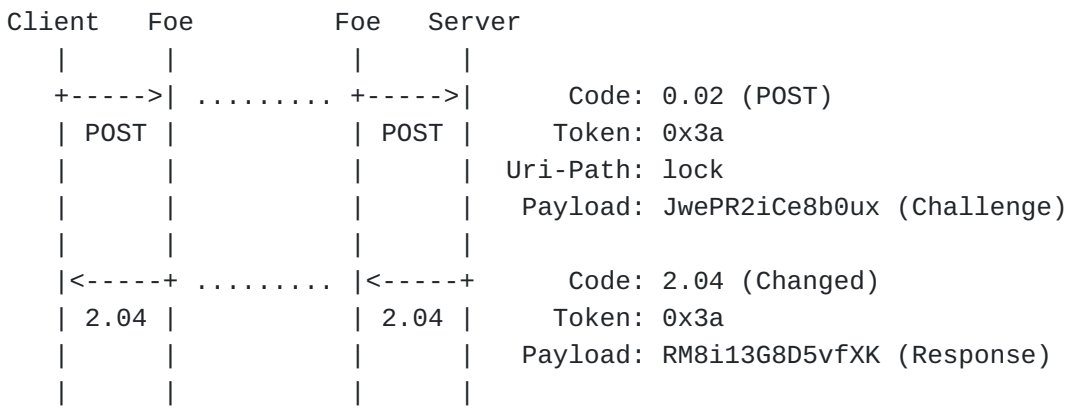

Figure 10: Relay attack (the client is the actuator)

The consequences may be severe, and in the case of a car, lead to the attacker unlocking and driving away with the car, an attack that unfortunately is happening in practice.

Remedy: Getting a response over a short-range radio cannot be taken as proof of proximity and can therefore not be used to take actions based on such proximity. Any automatically triggered mechanisms relying on proximity need to use other stronger mechanisms to establish proximity. Mechanisms that can be used are: measuring the round-trip time and calculating the maximum possible distance based on the speed of light, or using radio with an extremely short range like NFC (centimeters instead of meters). Another option is to include geographical coordinates (from e.g., GPS) in the messages and calculate proximity based on these, but in this case the location measurements need to be very precise and the system need to make sure that an attacker cannot influence the location estimation. Some types of global navigation satellite systems (GNSS) receivers are vulnerable to spoofing attacks.

3. Security Considerations

The whole document can be seen as security considerations for CoAP.

4. IANA Considerations

This document has no actions for IANA.

5. Informative References

[I-D.ietf-core-echo-request-tag]

Amsüss, C., Mattsson, J. P., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", Work in Progress, Internet-Draft, draft-ietf-core-echo-request-tag-14, 4 October 2021, <https://www.ietf.org/archive/id/draft-ietf-core-echo-request-tag-14.txt>.

[I-D.ietf-lake-edhoc] Selander, G., Mattsson, J. P., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", Work in Progress, Internet-Draft, draft-ietf-lake-edhoc-12, 20 October 2021, <https://www.ietf.org/archive/id/draft-ietf-lake-edhoc-12.txt>.

[I-D.ietf-tls-dtls13] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <https://www.ietf.org/archive/id/draft-ietf-tls-dtls13-43.txt>.

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

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://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, <https://www.rfc-editor.org/info/rfc7959>.

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

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <https://www.rfc-editor.org/info/rfc8323>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

Acknowledgements

The authors would like to thank Carsten Bormann, Klaus Hartke, Jaime Jiménez, Ari Keränen, Matthias Kovatsch, Achim Kraus, Sandeep Kumar, and András Méhes for their valuable comments and feedback.

Authors' Addresses

John Preuß Mattsson
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.mattsson@ericsson.com

John Fornehed
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.fornehed@ericsson.com

Göran 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

Christian Amsüss
Energy Harvesting Solutions

Email: c.amsuess@energyharvesting.at