



ESPILON
SECURITY RESEARCH

SECURITY ADVISORY

Denial of service from an MQTT v5.0 deserialization fault in coreMQTT

CVE-2026-8686

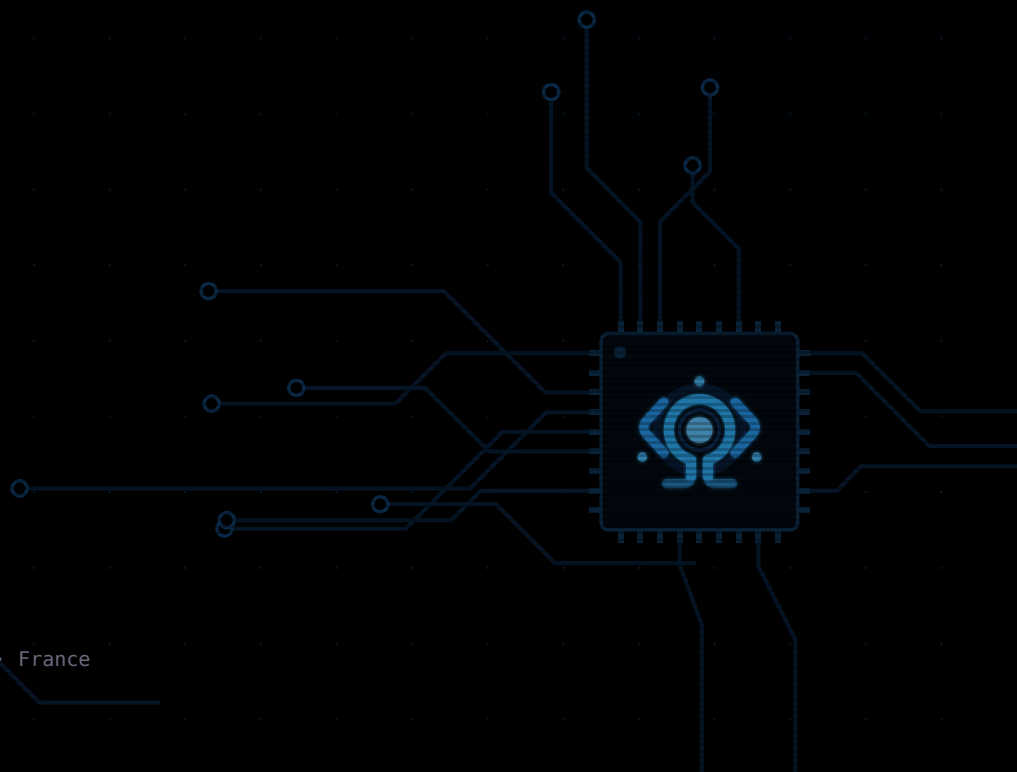
CVSS 7.5 · HIGH

AWS · coreMQTT

CVE PUBLISHED 15 May 2026

Marouane El Moufid (Eun0us)

espilon.net · Association loi 1901 · France



01 Vulnerability

7.5

HIGH

CVSS 3.1 BASE SCORE · AMAZON (CNA) · GHSA-6qh9-r6jp-2wxc

AV:N / AC:L / PR:N / UI:N / S:U / C:N / I:N / A:H

The CVE record also carries a CVSS 4.0 score of 8.7 (VA:H).

CVE	CVE-2026-8686 (CNA: Amazon)
Product	FreeRTOS coreMQTT, MQTT v5.0 client library in C
Component	deserializeSubUnsubAckProperties() in source/core_mqtt_serializer.c
Affected	v5.0.0 (released 2026-03-11), the only version listed by the advisory and the CVE record
Fixed	v5.0.1 (released 2026-03-27)
Type	Out-of-bounds read (CWE-125, as recorded by the CNA)
Attacker	The MQTT broker the client is connected to; network, no authentication
Credit	Epsilon · Marouane El Moufid (Eun0us), reporter

Summary

coreMQTT v5.0.0 is the first coreMQTT release that speaks MQTT v5.0. When it receives a SUBACK or UNSUBACK, it decodes the packet's property list in `deserializeSubUnsubAckProperties()`. The property length is a variable-length integer read from the packet, and v5.0.0 never compares it with the packet's remaining length. A broker that declares more property bytes than the packet holds makes the client walk the property list past the end of the received data. The vendor advisory describes the result as a heap out-of-bounds read, reachable over the network without authentication; the CNA scores it as a denial of service. v5.0.1 adds the missing bound.

MQTT v5.0 is not an option in v5.0.0: its README directs MQTT 3.1.1 users to v2.3.1, so every v5.0.0 deployment runs this parser. The v2.x line does not contain `deserializeSubUnsubAckProperties()` and is not listed as affected.

02 Technical details

Vulnerable code

`deserializeSubUnsubAck()` requires a remaining length of at least 4 bytes and a non-zero packet identifier, passes the rest to the property decoder, then derives the number of reason-code bytes from what the properties claimed to use:

```
• source/core_mqtt_serializer.c, deserializeSubUnsubAck() (coreMQTT v5.0.0, abridged)

1900     if( ( status == MQTTSuccess ) && ( incomingPacket->remainingLength > 4U ) )
1901         status = deserializeSubUnsubAckProperties( pPropBuffer,
1902         ....
1914         size_t totalPropertiesLength = ( size_t ) ( propertyLengthU32 + variableLengthEncodedSize( propertyLengthU32 ) );
1917         statusTotalBytes = remainingLength - sizeof( uint16_t ) - totalPropertiesLength;
1920         pStatusStart = &pIndex[ totalPropertiesLength ];
1921         status = readSubbackStatus( statusTotalBytes, pStatusStart, pReasonCodes );
```

The property decoder takes the property length from the wire and uses it, unchecked, as the bound of its walk:

```
• source/core_mqtt_serializer.c, deserializeSubUnsubAckProperties() (coreMQTT v5.0.0, abridged)
```

```
2606     status = decodeVariableLength( pLocalIndex, ( size_t ) ( remainingLength - 2U ), &propertyLength );
....
2626     pLocalIndex = &pLocalIndex[ variableLengthEncodedSize( propertyLength ) ];
....
2629     if( pPropBuffer != NULL )
2630     {
2631         pPropBuffer->bufferLength = propertyLength;
2632         pPropBuffer->pBuffer = pLocalIndex;
2633     }
2634
2635     while( ( propertyLength > 0U ) && ( status == MQTTSuccess ) )
2636     {
2637         uint8_t propertyId = *pLocalIndex;
2638         pLocalIndex = &pLocalIndex[ 1 ];
2639         propertyLength -= 1U;
2640
2641         ....
2643         case MQTT_REASON_STRING_ID:
2644             status = decodeUtf8( &pReasonString, &reasonStringLength, &propertyLength,
2645                                 &reasonString, &pLocalIndex );
2646
2647         ....
2664             status = decodeUserProp( &propertyKey, &propertyKeyLen, &propertyValue,
2665                                     &propertyValueLen, &propertyLength, &pLocalIndex );
```

Root cause

The only quantity the loop is checked against is `propertyLength`, the attacker's own number. The helpers do the same: `decodeUtf8()` (`core_mqtt_serializer_private.c:287`) rejects a string only if its 2-byte length exceeds the remaining `propertyLength`, never the remaining packet. Nothing between line 2606 and the loop compares the decoded value with `remainingLength`, so every property ID and string length read beyond the packet's last byte is read from whatever memory follows it.

If the walk completes, a second effect follows. Line 1917 is evaluated in `size_t`; with a property length larger than the packet, the subtraction wraps to a value close to `SIZE_MAX`, and `readSubackStatus()` then reads reason-code bytes beyond the packet until it meets one that is not a valid reason code. The `MQTTPropBuilder_t` handed to the application callback also carries the unchecked length (lines 2631 to 2632).

Reachability

SUBACK and UNSUBACK are handled from the receive loop: `handleIncomingAck()` (`core_mqtt.c:2262`) calls `handleSubUnsubAck()`, which calls `MQTT_DeserializeAck()` (`core_mqtt.c:4017`), which dispatches both types to `deserializeSubUnsubAck()` (`core_mqtt_serializer.c:4806`). Beyond parameter checks and the negotiated maximum packet size, nothing else is required; the client does not check that it has a SUBSCRIBE or UNSUBSCRIBE outstanding before decoding. The attacker is the broker the device is connected to, or any party able to stand in for it on the connection. No credentials are needed on the attacker's side and no action from the device user.

03 Impact and proof

Impact

The CNA rates the issue C:N/I:N/A:H: a crafted SUBACK or UNSUBACK makes the client read outside the received packet, and the consequence scored is a crash of the client. The vendor advisory text adds that adjacent heap memory may be read; the CNA vector does not score confidentiality, and this report does not claim a disclosure channel back to the broker. There is no write primitive.

Proof

Our report to AWS on 2026-03-24 carried a 6-byte UNSUBACK found by fuzzing. We replayed it on 2026-10-28 against the v5.0.0 tag (0dd6943), with the PoC sent in that report and AddressSanitizer. The packet bytes after the fixed header are copied into a heap block of exactly their length, as a receive buffer would hold them. After the packet identifier 06 0b, the three bytes 8c 98 2e encode a property length of 756,748, and the first property identifier read at line 2637 is the byte just past the packet.

- AddressSanitizer, coreMQTT v5.0.0, packet b0 | 06 0b 8c 98 2e (replayed 2026-10-28, addresses as printed)

```
==2759732==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x602000000015
READ of size 1 at 0x602000000015 thread T0
#0 in deserializeSubUnsubAckProperties source/core_mqtt_serializer.c:2637:30
#1 in deserializeSubUnsubAck          source/core_mqtt_serializer.c:1902:18
#2 in MQTT_DeserializeAck            source/core_mqtt_serializer.c:4806:26
#3 in main                           poc.c:67:27
0x602000000015 is located 0 bytes to the right of 5-byte region [0x602000000010,0x602000000015)
SUMMARY: AddressSanitizer: heap-buffer-overflow core_mqtt_serializer.c:2637:30 in deserializeSubUnsubAckProperties
```

The vendor's regression test for the fix (test/unit-test/core_mqtt_oob_ustest.c:44 in v5.0.1) uses a SUBACK of 23 bytes whose remaining length is 21 but whose property length is declared as 28. v5.0.0 accepts it and reads the key and value of the last user property, lengths and string bytes, from the ten bytes after the packet. The test asserts that v5.0.1 returns MQTTBadResponse.

04 Fix and recommendation

Upstream fix

PR #367 (86c684e, a91f6b1, 07a1be3, merged 2026-03-26) is the whole serializer change in v5.0.1:

- git diff v5.0.0 v5.0.1 -- source/core_mqtt_serializer.c (abridged: unchanged context lines omitted)

```
@@ -1897,7 +1897,7 @@ static MQTTStatus_t deserializeSubUnsubAck( const MQTTPacketInfo_t * incomingPac
-   if( ( status == MQTTSuccess ) && ( incomingPacket->remainingLength > 4U ) )
+   if( ( status == MQTTSuccess ) && ( incomingPacket->remainingLength >= 4U ) )
@@ -2600,6 +2600,7 @@ static MQTTStatus_t deserializeSubUnsubAckProperties( MQTTPropBuilder_t * pPropB
assert( !CHECK_U32T_OVERFLOW_SIZE_T( remainingLength ) );
+   assert( remainingLength >= 2U );
@@ -2623,13 +2624,26 @@ static MQTTStatus_t deserializeSubUnsubAckProperties( MQTTPropBuilder_t * pPropB
if( status == MQTTSuccess )
{
-   pLocalIndex = &pLocalIndex[ variableLengthEncodedSize( propertyLength ) ];
-   }
+   uint32_t bytesForPropLen = variableLengthEncodedSize( propertyLength );
+
+   if( pPropBuffer != NULL )
+   {
+       pPropBuffer->bufferLength = propertyLength;
+       pPropBuffer->pBuffer = pLocalIndex;
+       pLocalIndex = &pLocalIndex[ ( size_t ) bytesForPropLen ];
+
+       /* Remaining length should be at least big enough to hold the packet ID, the properties
+        * and the encoded property length. */
+       if( ( propertyLength + bytesForPropLen + 2U ) > remainingLength )
+       {
+           LogError( ( "Invalid property length. Goes beyond the packet bounds." ) );
+           status = MQTTBadResponse;
+       }
+       else if( pPropBuffer != NULL )
+       {
+           pPropBuffer->bufferLength = propertyLength;
+           pPropBuffer->pBuffer = pLocalIndex;
+       }
+       else
+       {
+           /* Nothing to do. No property buffer provided and no error. */
+       }
+   }
}
```

The new check rejects any property length that, with its own encoding and the 2-byte packet identifier, does not fit in the remaining length. That bounds the walk to the packet and keeps line 1917 from wrapping; the property buffer handed to the application is filled only after the length is validated.

Recommendation

Upgrade to coreMQTT v5.0.1 or later; the advisory states that no workaround exists. Products that vendor or fork the v5.0.0 serializer should apply PR #367 and its regression test. TLS with broker authentication narrows the attacker to the broker itself but does not fix the parser.

05 Disclosure

Timeline

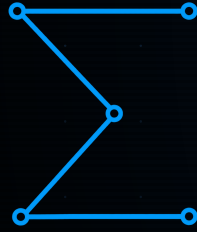
DATE	EVENT
2026-03-11	coreMQTT v5.0.0 released, with MQTT v5.0 support and the vulnerable code
2026-03-24	Reported by Espilon to AWS Security (aws-security@amazon.com), with a PoC and the crashing input
2026-03-26	Fix merged upstream (PR #367)
2026-03-27	coreMQTT v5.0.1 released
2026-05-15	CVE-2026-8686 published by Amazon (CNA); advisory GHSA-6qh9-r6jp-2wxc published

Credit

Reported by Marouane El Moufid (EunOus) of Espilon. The vendor advisory thanks “EunOus of Espilon for collaborating on this issue through the coordinated vulnerability disclosure process” and lists EunOus as reporter.

References

Advisory	github.com/FreeRTOS/coreMQTT/security/advisories/GHSA-6qh9-r6jp-2wxc
CVE record	cve.org/CVERecord?id=CVE-2026-8686
AWS bulletin	aws.amazon.com/security/security-bulletins/2026-032-aws/
Fix	github.com/FreeRTOS/coreMQTT/pull/367
Fix commit	github.com/FreeRTOS/coreMQTT/commit/86c684ea5cde0ca2ab9fcfec1d8b3ff59220efdd
Release	github.com/FreeRTOS/coreMQTT/releases/tag/v5.0.1



ESPILON

SECURITY RESEARCH

Embedded, IoT and OT security research

espilon.net

eun0us@espilon.net

Association loi 1901 • France