EUF-CMA for CMS SignedData

IETF 121 – LAMPS

The Problem

ForgedAttributes: An Existential Forgery Vulnerability of CMS and PKCS#7 Signatures, Falko Strenzke, https://eprint.iacr.org/2023/1801.pdf

- Different signing behaviour when SignedAttributes is present allows an existential forgery.

Attack

Presented by Falko

- signedAttrs:
  - SEQUENCE of attributes
  - one of them is the messageDigest attribute:
    - contains Hash(M)
  - $\text{signedAttr}_M^{\text{DER}} = \text{DER-encode}(\text{signedAttrs}(M))$
    - to indicate they contain Hash(M)

```
SignerInfo ::= SEQUENCE {
        version CMSVersion,
        sid SignerIdentifier,
        digestAlgorithm DigestAlgorithmIdentifier,
        signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,
        signatureAlgorithm SignatureAlgorithmIdentifier,
        signature SignatureValue,
        unsignedAttrs [1] IMPLICIT UnsignedAttributes OPTIONAL }

SignedAttributes ::= SET SIZE (1..MAX) OF Attribute


Attribute ::= SEQUENCE {
        attrType OBJECT IDENTIFIER,
        attrValues SET OF AttributeValue }

AttributeValue ::= ANY
```

Attack variant 1: Let the signer sign an attacker-chosen message of specific form w/o signedAttrs:

1: **procedure** CMS-Sign(secret key K_s, message M)
2: **if** signedAttrs are absent **then**
3: $D = \mathrm{HASH}(M)$
4: **else**
5: $D = \mathrm{HASH}(\mathrm{signedAttr}_M^{\mathrm{DER}})$
6: **end if**
7: return sign(K_s, D)
8: **end procedure**

Attack variant 1: Let the signer sign an attacker-chosen message of specific form w/o signedAttrs:

1: **procedure** CMS-Sign(secret key K_s, message M)
2: **if** signedAttrs are absent **then**
3: $D = \mathrm{HASH}(M)$ // $M = \mathrm{signedAttr}_M^{\mathrm{DER}} \leftarrow$ 👻
4: **else**
5: $D = \mathrm{HASH}(\mathrm{signedAttr}_M^{\mathrm{DER}})$
6: **end if**
7: return sign(K_s, D)
8: **end procedure**

Attack variant 1: Let the signer sign an attacker-chosen message of specific form w/o signedAttrs:

1: **procedure** CMS-Sign(secret key K_s, message M)
2: **if** signedAttrs are absent **then**
3: $D = \text{HASH}(M)$ // $M = \text{signedAttr}_M^{\text{DER}} \leftarrow$ 👻
4: **else**
5: $D = \text{HASH}(\text{signedAttr}_M^{\text{DER}})$
6: **end if**
7: return sign(K_s, D)
8: **end procedure**

Attack variant 1: Let the signer sign an attacker-chosen message of specific form w/o signedAttrs:

1: **procedure** CMS-Sign(secret key K_s, message M)
2: **if** signedAttrs are absent **then**
3: $D = \mathrm{HASH}(M)$ // $M = \mathrm{signedAttr}^{\mathrm{DER}}_{M'} \leftarrow$ 👻
4: **else**
5: $D = \mathrm{HASH}($ signedAttr$^{\mathrm{DER}}$) // ←↑ cannot distinguish, signature valid for M'
6: **end if** // 👻(adds signedAttrs to IS)
7: return sign(K_s, D)
8: **end procedure**

Attack variant 1: Let the signer sign an attacker-chosen message of specific form w/o signedAttrs:

1: **procedure** CMS-Sign(secret key K_s, message M)
2: **if** signedAttrs are absent **then**
3: $D = \mathrm{HASH}(M)$ // $M = \mathrm{signedAttr}_M^{\mathrm{DER}} \leftarrow$ 👻
4: **else**
5: $D = \mathrm{HASH}($ signedA ttr$^{\mathrm{DER}}$) // ←↑ cannot distinguish, signature valid for M'
6: **end if** // 👻(adds signedAttrs to IS)
7: return sign(K_s, D)
8: **end procedure**

→ Can forge signatures for arbitrary attacker-chosen message

Attack variant 2: Let the signer sign any message with signedAttrs:

1: **procedure** CMS-Sign(secret key K_s, message M)
2: **if** signedAttrs are absent **then**
3: $D = \mathrm{HASH}(M)$
4: **else**
5: $D = \mathrm{HASH}(\mathrm{signedAttr}_M^{\mathrm{DER}})$
6: **end if**
7: return sign(K_s, D)
8: **end procedure**

Attack variant 2: Let the signer sign any message with signedAttrs:

1: **procedure** CMS-Sign(secret key K_s, message M)
2: **if** signedAttrs are absent **then**
3: $D = \mathrm{HASH}(M)$
4: **else**
5: $D = \mathrm{HASH}(\mathrm{signedAttr}_M^{\mathrm{DER}})$ // $M' = \mathrm{signedAttr}_M^{\mathrm{DER}} \leftarrow$ 👻
6: **end if**
7: return sign(K_s, D)
8: **end procedure**

Attack variant 2: Let the signer sign any message with signedAttrs:

1: **procedure** CMS-Sign(secret key K_s, message M)
2: **if** signedAttrs are absent **then**
3: $D = \mathrm{HASH}(M')$ // ← cannot be distinguished from this case (remove signedAttrs)
4: **else**
5: $D = \mathrm{HASH}(\mathrm{signedAttr}_M^{\mathrm{DER}})$ // $M' = \mathrm{signedAttr}_M^{\mathrm{DER}} \leftarrow$ 👻
6: **end if**
7: return sign(K_s, D)
8: **end procedure**

Attack variant 2: Let the signer sign any message with signedAttrs:

1: **procedure** CMS-Sign(secret key K_s, message M)
2: **if** signedAttrs are absent **then**
3: $D = \mathrm{HASH}(M')$ // ← cannot be distinguished from this case (remove signedAttrs)
4: **else**
5: $D = \mathrm{HASH}(\mathrm{signedAttr}_M^{\mathrm{DER}})$ // $M' = \mathrm{signedAttr}_M^{\mathrm{DER}}$ ←
6: **end if**
7: return sign(K_s, D)
8: **end procedure**

→ Can forge signatures for message of form $\mathrm{signedAttr}_M^{\mathrm{DER}}$

Unaffected Systems

- Where SignedAttributes is mandatory:
  - SCEP
  - Certificate Transparency
  - RFC 4108 firmware update
  - German Smart Metering CMS data format
- When the message is signed then encrypted.

Conceivably Vulnerable Systems

- Must allow absence of SignedAttributes
- Unencrypted firmware update denial of service
- Dense message space
- Signing unstructured data
- External signatures over unstructured data
- Systems with permissive parsers

Mitigation

Presented by Daniel

Application-level Mitigation

- (Within CMS) Fail signature generation and verification if the message is a valid DER-encoded SignedAttributes.
- Require SignedAttributes.
- Forbid SignedAttributes
- (Within application) More robust parsing, discard

CMS Mitigation

We sketch three options:

- Quick and Dirty
- More Flexible
- Both

CMS Mitigation: Quick and Dirty

- Update *cms-sphincs-plus*, *cms-ml-dsa*, and *pq-composite-sigs* to set signature context: “CMS-with-SignedAttrs” vs “CMS-without-SignedAttrs”
- (?)RFC to specify this behaviour for future signature algorithms.
- (?)Future signature algorithms refer to this RFC.
- Doesn’t address RSA, ECDSA, EdDSA
- Pro: forces implementations to support context now, universal support
- Con: forces implementations to support context now

CMS Mitigation: More Flexible

- New unsigned attribute: sign-with-context
- Attribute contains the context string:
  “<keyword_1>[=value];…; <keyword_n>[=value]”
- Keywords are ordered alphanumerically
- Sign(K, M, ctx=“IETF/CMS:” + context_string)
  - or Sign(K, M, ctx=“IETF/CMS:” + HASH(context_string)) to allow a longer context string.
- keyword_1: “signedattrs” for when signed attributes are used.
- keyword/value 2?: “application_ctx=<value>”, e.g. “S/MIME”.

CMS Mitigation: More Flexible

- *cms-sphincs-plus*, *cms-ml-dsa*, and *pq-composite-sigs* progress with default context = “”, implementations indicate support with new attribute.
  - Requires signer to know that the verifier supports the attribute.
- Could create EdDSActx if anyone cared.
- Doesn’t address RSA, ECDSA
- Pro: allows current drafts to progress with no changes, gives implementations time to support context
- Con: may never be universally supported

CMS Mitigation: Both

- Update *cms-sphincs-plus*, *cms-ml-dsa*, and *pq-composite-sigs* to set signature context: “CMS-with-SignedAttrs” vs “CMS-without-SignedAttrs” unless overridden by some other advertised values.
- New unsigned attribute: sign-with-context
  - If the attribute it used, it replaces and “CMS-with-SignedAttrs” vs “CMS-without-SignedAttrs”

Next Steps?

- Should the WG address this issue?
  - yes/maybe -> draft
- With which mitigation?