MONROE ELECTRONICS

Industry Advisory

DIGITAL SIGNATURES AND C14N CROSS PLATFORM COMPATIBILITY ISSUES:

RECOMMENDATIONS FOR FEMA IPAWS AND OTHER CAP SYSTEMS.

CONTENTS

Overview and Recommendations

This advisory identifies and details a cross platform compatibility issue when XML Signatures are signed with Inclusive Canonicalization (also known as C14N). Specifically, an XML signature produced on a Windows .NET system cannot be verified using the XMLSec (both on Linux and Windows). We have found that the reason for the incompatibility is that the Inclusive C14N algorithm used on .NET systems does not match the W3C spec, while the algorithm implemented on XMLSec does, resulting in potentially different "canonicalized" SignedInfo bytestreams.

This cross platform compatibility issue is likely to pose significant issues in the IPAWS CAP environment, as it appears that Inclusive C14N digitally signed CAP messages have been appearing in the IPAWS operational and test environments, contrary to IPAWS design guidance.

The introduction of Inclusive C14N signatures was not anticipated by the EAS CAP manufacturing community, which has by and large been adhering closely to IPAWS design guidance and conformity requirements (including reliance on *Exclusive* C14N). Further, it is not immediately clear whether all EAS CAP manufacturers and products can adapt to a broader requirement to accept both Inclusive C14N and Exclusive C14N signatures.

Electrostatic Measurement
Emergency Alert Systems
CATV Switching and Control

Revision: 001 Publication: 7/19/12

585-765-2254 fax 585-765-9330
100 Housel Ave. | Lyndonville | NY | 14098
www.monroe-electronics.com

As detailed below, due the cross platform compatibility issues inherent in Inclusive C14N, we recommend that the IPAWS aggregator should, at the earliest opportunity, be enabled to to detect and reject CAP alerts that include <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/> in their Signature blocks.

Background: IPAWS and Exclusive Canonicalization

In the **IPAWS 3.01_02 Designers Guide** documentation, Exclusive C14N is listed as the method that is supposed to be used for signing alerts (see chart on section 4, page 13, regarding signing SOAP enveloped CAP messages submitted to the aggregator). This is further clarified in section 5, page 17, which states: "***NOTE: To ensure successful verification of digitally signed CAP messages it is important that Canonicalization (Exclusive) form be utilized."***

This indicates that IPAWS has determined that Exc-C14N is to be required. However, we have already observed numerous signed CAP messages posted in both the IPAWS operational and IPAWS TDL environments that are not using Exc-C14N. These messages are using Inclusive C14N (Inc-C14N), which poses specific cross platform compatibility issues.

For whatever reasons, some developers do not appear to be adhering to IPAWS design guidance for their respective CAP origination systems. Further, it does not appear to be the case that the IPAWS aggregator is enforcing the Exc-C14N restriction.

Based on the analysis below, we strongly recommend that the FEMA IPAWS aggregator enforce adherence to Exc-C14N, and reject any alerts that specify Inclusive C14N as having an invalid signature.

We do not, at this point, recommend that CAP EAS equipment manufacturers should accommodate alerts that specify inclusive C14N, as this appears to be contrary to IPAWS developers' guidance). While we have already prepared an option to accommodate Inclusive C14N, we would much prefer that the FEMA IPAWS aggregator enforce Exc-C14N to mitigate likely cross platform compatibility issues in a multi-vendor environment.

Background: State and Local CAP alerting systems

As increasing numbers of CAP alert origination platforms are appearing, we are likewise noting the appearance of Inclusive C14N in several of these platforms. On the one hand, there are no specific design guidelines for individual CAP message origination systems, per se. On the other hand, importantly, a CAP message origination system intending to interoperate with FEMA IPAWS is required to adhere to the IPAWS developers' guidance.

Our key recommendation for state and local CAP alerting systems is that they must likewise utilize Exclusive C14N if they are utilizing digital signatures. Because of the strong likelihood of cross-platform incompatibility, reliance on Inclusive C14N should be discouraged whether or not a CAP origination system intends on interoperating with IPAWS.

Analysis: Inclusive C14N Cross Platform Compatibility Issues

We have characterized a problem wherein an XML signature produced on a Windows .NET system cannot be verified using the XMLSec (both on Linux and Windows).

The reverse also occurs, where the same XML signed under XMLSec fails to verify on the Windows .NET system. The actual issue lies with the C14N software, and which technically, on the XMLSec side, is a part of the libXML2 code and not in the core XMLSec code.

This same behavior was also found using the Linux Apache XML Signature software.

XML Signature with Inclusive C14N http://www.w3.org/TR/2001/REC-xml-c14n-20010315	.NET sign	XMLSec or Apache XML Signature sign
.NET verify	OK	Failure
XMLSec or Apache XML Signature verify	Failure	OK

The root of the problem is actually very simple and only occurs when the Signature CanonicalizationMethod is Inclusive C14N (http://www.w3.org/TR/2001/REC-xml-c14n-20010315).

If the C14N method is Exclusive C14N ("http://www.w3.org/2001/10/xml-exc-c14n#") then there is no cross platform incompatibility.

XML Signature with Exclusive C14N http://www.w3.org/2001/10/xml-exc-c14n#	.NET sign	XMLSec or Apache XML Signature sign
.NET verify	OK	OK
XMLSec or Apache XML Signature verify	OK	OK

As background, it is useful to look at the two essential parts to XML signature verification.

1. **Reference Validation.** This is achieved by recreating the DigestValue from the Reference URI. If the Digests match, then you know that the XML data (in our case the CAP alert info) has not been changed from the original. At this point verification can continue in the attempt to validate the Signature value.
2. **Signature Validation.** We will not go into the full details here, as it involves several elements of the Signature block, such as the SignatureValue computed when the document was signed, hashing, decryption with a X509Certificate Public Key, and the canonicalized form of the full <SignedInfo> block. However, the important issuet is that - during the signing and verification process - the entire <SignedInfo> block element, including xmlns namespace attributes inside the tags (*this is important*) is serialized into a reproducible "canonicalized" bytestream form, using the canonicalization (C14N) method specified in the node `<CanonicalizationMethod>`.

This "canonicalized" form is input data for both the signing and verification calculations.

During verification, the C14N transformation of the <SignedInfo> block must match the <SignedInfo> form used when the document was signed. Any deviation will invalidate the Signature verification computation. In the end, if the given SignatureValue can be proved to

match the expected SignatureValue computed from the X509Certificate Public Key and the <SignedInfo> block "canonicalized" bytestream, then the Signature is verified.

Note, that DigestValue, which is a hash of the Referenced XML file (in our case the CAP alert), is included inside the <SignedInfo> block, so therefore, the Signature includes the essential "fingerprint" of the Referenced XML data.

Since Signatures contain the DigestValue, and are created using secure certificates, and the certificate can be traced back to a trusted source, a verified signature means that the data in the XML file has not only been preserved intact from its creation, but also, the author has been verified and can also be verified against a trusted reference.

A simple way to describe this is that the canonicalized <SignedInfo> block is used to create the signature as well as to verify the signature. The specified `<CanonicalizationMethod>` inside the <SignedInfo> says how this was done when the Signature was produced, and how this must be done when the Signature is verified.

Any differences between the signing instance and the verification instance of the canonicalized <SignedInfo> will cause a signature verification to fail.

This explains the incompatibility. We have found that the reason for the incompatibility is that the Inclusive C14N algorithm used on .NET systems does not match the W3C spec, while the algorithm implemented on XMLSec does, resulting in potentially different "canonicalized" SignedInfo bytestreams.

The different implementations create a strong likelihood that a cross platform incompatibility will occur when Inclusive C14N is used. To be exact, it depends on if more than one xmlns namespace attributes are included in the <Signature> tag, because these get propagated differently into the <SignedInfo> node, depending on the platform running the Inclusive C14N method.

Examples

Below are the relevant parts of an example <Signature> block from a CAP alert, specified as using Inclusive C14N. Examine the <Signature> node xmlns namespace attribute definitions. During Signature verification C14N operations will propagate one or more of these into attributes into the <SignedInfo> node.

Note that this occurs as a hidden, intermediate step during verification, so special tools or debugging modes must be used to see the effects of C14N.

```
<Signature:Signature
xmlns="http://www.w3.org/2000/09/xmldsig#"  xmlns:u="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd"  xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"  xmlns:Signature="http://www.w3.
org/2000/09/xmldsig#" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SignedInfo>
<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
<SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-
sha256"/>
<Reference URI="">
<Transforms><Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/></Transforms>
<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
<DigestValue>PtRfa3EJKTr6yeeuokpGu4KvHnGPAcd1YqhZtts+qOs=</DigestValue>
</Reference>
</SignedInfo>

<SignatureValue>T3NO+cj2tBIhDI4w/MnauJhzusMkUppq94MZWCKm....</SignatureValue>
...
...
</Signature>
```

During Windows .NET Inclusive C14N transformation, the <SignedInfo> node is canonicalized to this bytestream (this can be produced from a Windows C14N C-Sharp application, see http://pages.infinit.net/ctech/20040316-0936.html):

```
<SignedInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"></CanonicalizationMethod>
<SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"></SignatureMethod>
<Reference URI="">
<Transforms><Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"></Transform></Transforms>
<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"></DigestMethod>
<DigestValue>PtRfa3EJKTr6yeeuokpGu4KvHnGPAcd1YqhZtts+qOs=</DigestValue>
</Reference>
</SignedInfo>
```

But using XMLSec (using libXML2 underneath) C14N, the <SignedInfo> node is canonicalized to a different bytestream (this can be produced from the XMLSec application using the --store-signatures parameter):

```
<SignedInfo xmlns="http://www.w3.org/2000/09/xmldsig#"
xmlns:Signature="http://www.w3.org/2000/09/xmldsig#"
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:u="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"></CanonicalizationMethod>
<SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"></SignatureMethod>
<Reference URI="">
<Transforms><Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"></Transform></Transforms>
<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"></DigestMethod>
<DigestValue>PtRfa3EJKTr6yeeuokpGu4KvHnGPAcd1YqhZtts+qOs=</DigestValue>
</Reference>
</SignedInfo> xmlns:u= xmlns:xsi=
```

Notice the simple difference between the **xmlns attrs** in **<SignedInfo>** on the two systems.

This difference is all that is needed to ruin the Signature portability across these systems. Also remember that Inclusive C14N signatures will verify within the same platform, just not cross platform. Therefore, the problem is only discovered later if the CAP alert ends up in a different system for verification.

It is interesting and useful to point out that if Exclusive C14N is used, both systems canonicalize the <SignedInfo> block to the same bytestream. That bytestream pruduces <SignedInfo xmlns="http://www.w3.org/2000/09/xmldsig#">, with no other changes, and so no problem occurs with cross platform verification. This makes Exclusive C14N the safe choice.

It is also perhaps even more interesting to note that the .NET Inclusive C14N basically replicates what Exclusive C14N does to the namespace propagation into <SignedInfo>. It produces the same result as the Exclusive C14N method. So if the Signature validation software can ignore the requested method written into the <CanonicalizationMethod> and just tries to use Exclusive C14N, the .NET originated signature will always verify!

Because of this, a "modified" XMLSec application can verify a signature from the .NET Windows system if in midstream it substitutes the Exclusive C14N transform instead of using the requested Inclusive C14N (requested from the XML Signature). We have confirmed this. If Exclusive C14N is switched on at the right moment, suddenly XML signatures from the "other" system verify, when previously, using the correct libXML2 Inclusive C14N method, they would not.

We contacted the **author of XMLSec and the C14N code in libXML2,** a published expert on XML signatures. He had a surprising statement. He stated that this **incompatibility has actually been known for around 10 years.** The Inclusive C14N implementation in **libXML2 follows the W3C spec,** while **.NET does not.** He states that Microsoft has not modified the code in order to maintain backward compatibility.

From our careful reading of the W3C Canonical XML spec, we concur that the libXML2 result is correct, wherein all namespace attributes from <Signature> should be sorted and propagated into <SignedInfo> under Inclusive C14N, as in

```
<SignedInfo xmlns="http://www.w3.org/2000/09/xmldsig#"
xmlns:Signature="http://www.w3.org/2000/09/xmldsig#"
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
```

```
xmlns:u="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

Thus the compatibility problem is strictly caused by the difference in the SignedInfo node xmlns namespace attribute handling during the Inclusive Canonicalization step.

Conclusion and Recommendations

Since the IPAWS documentation already specifies that Exclusive C14N is the correct method to use for CAP alerts sent to the IPAWS system, it follows CAP alerts signed using Inclusive C14N should be considered an error in order to avoid this pitfall.

We feel that It would be appropriate for the IPAWS aggregator to detect and reject CAP alerts that include <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/> in their Signature blocks.

We further strongly recommend that CAP message origination platforms rely upon the Exclusive C14N method if their CAP messages are intended for consumption by CAP EAS equipment, whether via IPAWS or other dissemination means.