How To Break XML Signature and XML Encryption

Juraj Somorovsky
Horst-Görtz Institute
Ruhr-University of Bochum
Juraj.somorovsky@rub.de

OWASP 17.11.2011

Copyright © The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document under the terms of the OWASP License.
Motivation – Web Services

- Method for machine-to-machine communication over networks
- Used in commerce, finance, government, military, ...
- XML-based message format
Motivation – XML Security

• SSL / TLS: transport-level security

• Messages secured only during transport!
Motivation – XML Security

• Message level security

• Security applied directly on the messages
• No need for SSL / TLS
• Realized using XML Signature, XML Encryption
Motivation – XML Security

• Another example: Browser-based Single Sign-On

• Messages secured only during transport!
Motivation – XML Security

• Does SSL / TLS help?

• Need for message level security!
Motivation – XML Security

• Another example: Browser-based Single Sign-On

• Could be realized using XML Signature and XML Encryption
Motivation – XML Security

• W3C Standards: XML Signature and XML Encryption
• Describe various methods for applying cryptographic algorithms to XML documents

```xml
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```
Overview

1. Breaking XML Signature
   • Cloud Computing Management Interfaces
   • Amazon EC2 SOAP Interface
   • XML Signature Wrapping on Eucalyptus and Amazon
   • Countermeasures and Conclusion

2. Breaking XML Encryption
   • Attack Scenario
   • Decrypting by checking plaintext validity
   • Application to CBC mode of operation in XML Encryption
   • Countermeasures and Conclusion

3. Conclusion


Cloud Computing:
- Amazon Web Services (2006):
  - Public Cloud
- Eucalyptus Cloud (2009):
  - Reimplements Interfaces
  - Private Cloud
Cloud Computing Management Interfaces

- Controlling of the cloud using different interfaces
Overview

1. **Breaking XML Signature**
   - Cloud Computing Management Interfaces
   - Amazon EC2 SOAP Interface
   - XML Signature Wrapping on Eucalyptus and Amazon
   - Countermeasures and Conclusion

2. **Breaking XML Encryption**
   - Attack Scenario
   - Decrypting by checking plaintext validity
   - Application to CBC mode of operation in XML Encryption
   - Countermeasures and Conclusion

3. Conclusion


Amazon EC2 SOAP Interface

Monitoring of running instances:

```xml
<Envelope>
  <Header />
  <Body>
    <MonitorInstances>
      <InstanceId> instance </InstanceId>
    </MonitorInstances>
  </Body>
</Envelope>
```
Amazon EC2 SOAP Interface - XML Signature
Amazon EC2 SOAP Interface - XML Signature

<Envelope>
  <Header/>
  <Body>
    <MonitorInstances/>
  </Body>
</Envelope>

<Envelope>
  <Header/>
  <Body>
    <Instances/>
  </Body>
</Envelope>
Overview

1. Breaking XML Signature
   - Cloud Computing Management Interfaces
   - Amazon EC2 SOAP Interface
   - XML Signature Wrapping on Eucalyptus and Amazon
   - Countermeasures and Conclusion

2. Breaking XML Encryption
   - Attack Scenario
   - Decrypting by checking plaintext validity
   - Application to CBC mode of operation in XML Encryption
   - Countermeasures and Conclusion

3. Conclusion


XML Signature Wrapping

McIntosh and Austel. XML Signature Element Wrapping attacks, 2005

The same attack on the Timestamp!
XML Signature Wrapping on Eucalyptus

- Attack by McIntosh and Austel directly applicable
- Eucalyptus: Open Source
- Reason: Problem in Apache Axis Web Services Framework
- CVE-2011-0730
XML Signature Wrapping on Amazon

- Attack from McIntosh & Austel does not work
- Amazon checks, if the Id of the signed element equals to the Id of the processed element
- But what happens if we use two elements with the same Id?

- Which element is used for signature validation?

- Which for function execution?
XML Signature Wrapping on Amazon

Duplicate the Body element:
• Function invocation from the first Body element
• Signature verification over the second Body element

The same attack on the Timestamp!
XML Signature Wrapping on Amazon - Analysis

- Amazon: No Open Source
- Analysis using the SOAP error messages
  - The timestamp has expired
  - The timestamp or body was not signed
  - The certificate holder could not be authorized
  - The signature was invalid

- SOAP error messages = really good source of information

- We sent different hand-crafted messages to the Amazon EC2 interface
XML Signature Wrapping on Amazon - Analysis

We can really check if the user’s Public Key is valid!

If there is no signature, the signature validation phase is skipped!
XML Signature Wrapping on Amazon - Analysis

- Works only for the Amazon EC2 / Eucalyptus SOAP interface
- One valid SOAP message is enough to:
  - Start and stop cloud instances
  - Download and upload virtual images
  - ...get full control over the victim’s cloud!
Overview

1. Breaking XML Signature
   - Cloud Computing Management Interfaces
   - Amazon EC2 SOAP Interface
   - XML Signature Wrapping on Eucalyptus and Amazon
   - Countermeasures and Conclusion

2. Breaking XML Encryption
   - Attack Scenario
   - Decrypting by checking plaintext validity
   - Application to CBC mode of operation in XML Encryption
   - Countermeasures and Conclusion

3. Conclusion


XML Signature Wrapping - Conclusion

- We showed practical critical Signature Wrapping attacks on Amazon and Eucalyptus Cloud Interfaces
- All the vulnerabilities have been fixed
- XML Signature Wrapping attacks are known since 2005, but:
  - Are not in focus of research community
  - Nearly all implementations are vulnerable
- Please be aware of Signature Wrapping when applying XML Signatures
  - In Web Services
  - SAML (Single Sign-On)
  - Custom applications
- There are more attacks coming soon
Overview

1. Breaking XML Signature
   - Cloud Computing Management Interfaces
   - Amazon EC2 SOAP Interface
   - XML Signature Wrapping on Eucalyptus and Amazon
   - Countermeasures and Conclusion

2. Breaking XML Encryption
   - Attack Scenario
   - Decrypting by checking plaintext validity
   - Application to CBC mode of operation in XML Encryption
   - Countermeasures and Conclusion

3. Conclusion


XML Encryption

- W3C standard for encrypting XML data (published in 2002)
- Describes various methods for applying
  - Symmetric ciphers (AES-CBC, 3DES-CBC)
  - Public-key encryption (e.g. RSA-PKCS#1)

```xml
<PaymentInfo>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000'>
    <Number>4019 ...5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

```xml
<PaymentInfo>
  <EncryptedData Id="EncData">
    <EncryptionMethod Algorithm="...xmlenc#aes128-cbc"/>
    …<CipherValue>3bP...Zx0=</CipherValue>…
  </EncryptedData>
</PaymentInfo>
```
XML Encryption

• Attack on XML Encryption
• All major Web Services frameworks vulnerable
  • Apache Axis 2
  • RedHat JBoss
  • IBM WebSphere
  • Microsoft .NET
  • And more (recently discovered)
• Also applicable to XML-based Single Sign-On (recently discovered)
XML Encryption – Attack Scenario

What is a “valid” plaintext?
How to use Web Service as “plaintext validity oracle”?
How to use this oracle to decrypt C?
How to create useful chosen ciphertexts $C_1, C_2, ...$?
Overview

1. Breaking XML Signature
   - Cloud Computing Management Interfaces
   - Amazon EC2 SOAP Interface
   - XML Signature Wrapping on Eucalyptus and Amazon
   - Countermeasures and Conclusion

2. Breaking XML Encryption
   - Attack Scenario
   - Decrypting by checking plaintext validity
   - Application to CBC mode of operation in XML Encryption
   - Countermeasures and Conclusion

3. Conclusion


Deciphering by checking plaintext validity

- XML is a text-based data format
- Characters (usually) encoded in ASCII
  - Type A: “special” characters
    EOF, BEL, ACK, …, <, &,
  - Type B: other
    A,B,C, …, a,b,c, …, 1,2,3, …, !, %,
- This talk:
  “Valid” plaintext contains no Type-A character
Decryption by checking plaintext validity

- Using Web Services Server as plaintext validity oracle

- Invalid plaintext \rightarrow Parsing error
- Parsing error \rightarrow Fault message (or another side channel)
Consider ASCII character $M_1 = (0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$

<table>
<thead>
<tr>
<th>$0x00$</th>
<th>(Type A)</th>
<th>$0x20$</th>
<th></th>
<th>$0x40$</th>
<th>@</th>
<th>$0x60$</th>
<th>'</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0x01$</td>
<td>(Type A)</td>
<td>$0x21$</td>
<td>!</td>
<td>$0x41$</td>
<td>A</td>
<td>$0x61$</td>
<td>a</td>
</tr>
<tr>
<td>$0x02$</td>
<td>(Type A)</td>
<td>$0x22$</td>
<td>&quot;</td>
<td>$0x42$</td>
<td>B</td>
<td>$0x62$</td>
<td>b</td>
</tr>
<tr>
<td>$0x03$</td>
<td>(Type A)</td>
<td>$0x23$</td>
<td>#</td>
<td>$0x43$</td>
<td>C</td>
<td>$0x63$</td>
<td>c</td>
</tr>
<tr>
<td>$0x04$</td>
<td>(Type A)</td>
<td>$0x24$</td>
<td>$</td>
<td>$</td>
<td>$0x44$</td>
<td>D</td>
<td>$0x64$</td>
</tr>
<tr>
<td>$0x05$</td>
<td>(Type A)</td>
<td>$0x25$</td>
<td>%</td>
<td>$0x45$</td>
<td>E</td>
<td>$0x65$</td>
<td>e</td>
</tr>
<tr>
<td>$0x06$</td>
<td>(Type A)</td>
<td>$0x26$</td>
<td>&amp;</td>
<td>$0x46$</td>
<td>F</td>
<td>$0x66$</td>
<td>f</td>
</tr>
<tr>
<td>$0x07$</td>
<td>(Type A)</td>
<td>$0x27$</td>
<td>'</td>
<td>$0x47$</td>
<td>G</td>
<td>$0x67$</td>
<td>g</td>
</tr>
<tr>
<td>$0x08$</td>
<td>(Type A)</td>
<td>$0x28$</td>
<td>(</td>
<td>$0x48$</td>
<td>H</td>
<td>$0x68$</td>
<td>h</td>
</tr>
<tr>
<td>$0x09$</td>
<td>HT</td>
<td>$0x29$</td>
<td>)</td>
<td>$0x49$</td>
<td>I</td>
<td>$0x69$</td>
<td>i</td>
</tr>
<tr>
<td>$0x0A$</td>
<td>LF</td>
<td>$0x2A$</td>
<td>*</td>
<td>$0x4A$</td>
<td>J</td>
<td>$0x6A$</td>
<td>j</td>
</tr>
<tr>
<td>$0x0B$</td>
<td>(Type A)</td>
<td>$0x2B$</td>
<td>+</td>
<td>$0x4B$</td>
<td>K</td>
<td>$0x6B$</td>
<td>k</td>
</tr>
<tr>
<td>$0x0C$</td>
<td>(Type A)</td>
<td>$0x2C$</td>
<td>,</td>
<td>$0x4C$</td>
<td>L</td>
<td>$0x6C$</td>
<td>l</td>
</tr>
<tr>
<td>$0x0D$</td>
<td>CR</td>
<td>$0x2D$</td>
<td>-</td>
<td>$0x4D$</td>
<td>M</td>
<td>$0x6D$</td>
<td>m</td>
</tr>
<tr>
<td>$0x0E$</td>
<td>(Type A)</td>
<td>$0x2E$</td>
<td>.</td>
<td>$0x4E$</td>
<td>N</td>
<td>$0x6E$</td>
<td>n</td>
</tr>
<tr>
<td>$0x0F$</td>
<td>(Type A)</td>
<td>$0x2F$</td>
<td>/</td>
<td>$0x4F$</td>
<td>O</td>
<td>$0x6F$</td>
<td>o</td>
</tr>
<tr>
<td>$0x10$</td>
<td>(Type A)</td>
<td>$0x30$</td>
<td>0</td>
<td>$0x50$</td>
<td>P</td>
<td>$0x70$</td>
<td>p</td>
</tr>
<tr>
<td>$0x11$</td>
<td>(Type A)</td>
<td>$0x31$</td>
<td>1</td>
<td>$0x51$</td>
<td>Q</td>
<td>$0x71$</td>
<td>q</td>
</tr>
<tr>
<td>$0x12$</td>
<td>(Type A)</td>
<td>$0x32$</td>
<td>2</td>
<td>$0x52$</td>
<td>R</td>
<td>$0x72$</td>
<td>r</td>
</tr>
<tr>
<td>$0x13$</td>
<td>(Type A)</td>
<td>$0x33$</td>
<td>3</td>
<td>$0x53$</td>
<td>S</td>
<td>$0x73$</td>
<td>s</td>
</tr>
<tr>
<td>$0x14$</td>
<td>(Type A)</td>
<td>$0x34$</td>
<td>4</td>
<td>$0x54$</td>
<td>T</td>
<td>$0x74$</td>
<td>t</td>
</tr>
<tr>
<td>$0x15$</td>
<td>(Type A)</td>
<td>$0x35$</td>
<td>5</td>
<td>$0x55$</td>
<td>U</td>
<td>$0x75$</td>
<td>u</td>
</tr>
<tr>
<td>$0x16$</td>
<td>(Type A)</td>
<td>$0x36$</td>
<td>6</td>
<td>$0x56$</td>
<td>V</td>
<td>$0x76$</td>
<td>v</td>
</tr>
<tr>
<td>$0x17$</td>
<td>(Type A)</td>
<td>$0x37$</td>
<td>7</td>
<td>$0x57$</td>
<td>W</td>
<td>$0x77$</td>
<td>w</td>
</tr>
<tr>
<td>$0x18$</td>
<td>(Type A)</td>
<td>$0x38$</td>
<td>8</td>
<td>$0x58$</td>
<td>X</td>
<td>$0x78$</td>
<td>x</td>
</tr>
<tr>
<td>$0x19$</td>
<td>(Type A)</td>
<td>$0x39$</td>
<td>9</td>
<td>$0x59$</td>
<td>Y</td>
<td>$0x79$</td>
<td>y</td>
</tr>
<tr>
<td>$0x1A$</td>
<td>(Type A)</td>
<td>$0x3A$</td>
<td>:</td>
<td>$0x5A$</td>
<td>Z</td>
<td>$0x7A$</td>
<td>z</td>
</tr>
<tr>
<td>$0x1B$</td>
<td>(Type A)</td>
<td>$0x3B$</td>
<td>;</td>
<td>$0x5B$</td>
<td>[</td>
<td>$0x7B$</td>
<td>{</td>
</tr>
<tr>
<td>$0x1C$</td>
<td>(Type A)</td>
<td>$0x3C$</td>
<td>&lt;</td>
<td>$0x5C$</td>
<td>\</td>
<td>$0x7C$</td>
<td></td>
</tr>
<tr>
<td>$0x1D$</td>
<td>(Type A)</td>
<td>$0x3D$</td>
<td>=</td>
<td>$0x5D$</td>
<td>]</td>
<td>$0x7D$</td>
<td>}</td>
</tr>
<tr>
<td>$0x1E$</td>
<td>(Type A)</td>
<td>$0x3E$</td>
<td>&gt;</td>
<td>$0x5E$</td>
<td>^</td>
<td>$0x7E$</td>
<td>~</td>
</tr>
<tr>
<td>$0x1F$</td>
<td>(Type A)</td>
<td>$0x3F$</td>
<td>?</td>
<td>$0x5F$</td>
<td>_</td>
<td>$0x7F$</td>
<td>DEL</td>
</tr>
</tbody>
</table>
Overview

1. Breaking XML Signature
   - Cloud Computing Management Interfaces
   - Amazon EC2 SOAP Interface
   - XML Signature Wrapping on Eucalyptus and Amazon
   - Countermeasures and Conclusion

2. Breaking XML Encryption
   - Attack Scenario
   - Decrypting by checking plaintext validity
   - Application to CBC mode of operation in XML Encryption
   - Countermeasures and Conclusion

3. Conclusion


Decryption by checking plaintext validity

- ASCII exhibits nice pattern of Type A/B characters
- Suppose we can transform \(\text{Enc}(M)\) into \(\text{Enc}(M \oplus \text{msk})\) for any msk
  - **We can flip arbitrary plaintext bits, given only the ciphertext**
- Approach: Given \(C = \text{Enc}(M)\),
  1. Modify plaintext character-wise
  2. Query the “oracle” with each modified ciphertext
  3. Observe whether plaintext remains “valid” or not
Deciphering by checking plaintext validity

- **Example**
  - We have eavesdropped a ciphertext $C = \text{Enc("ACMCCS11")}$
  - We recover $M = "ACMCCS11"$ character-wise
  - How to determine $(b_1, b_2)$ of $M_1 = "A"$?
Consider ASCII character $M_1 = (0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$

<table>
<thead>
<tr>
<th>$0x00$ (Type A)</th>
<th>$0x20$</th>
<th>$0x40$</th>
<th>$0x60$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0x01$ (Type A)</td>
<td>🟢 $0x21$</td>
<td>🟢 $0x41$</td>
<td>🟢 $0x61$</td>
</tr>
<tr>
<td>$0x02$ (Type A)</td>
<td>🟢 $0x22$</td>
<td>🟢 $0x42$</td>
<td>🟢 $0x62$</td>
</tr>
<tr>
<td>$0x03$ (Type A)</td>
<td>🟢 $0x23$</td>
<td>🟢 $0x43$</td>
<td>🟢 $0x63$</td>
</tr>
<tr>
<td>$0x04$ (Type A)</td>
<td>🟢 $0x24$</td>
<td>🟢 $0x44$</td>
<td>🟢 $0x64$</td>
</tr>
<tr>
<td>$0x05$ (Type A)</td>
<td>🟢 $0x25$</td>
<td>🟢 $0x45$</td>
<td>🟢 $0x65$</td>
</tr>
<tr>
<td>$0x06$ (Type A)</td>
<td>🟢 $0x26$</td>
<td>🟢 $0x46$</td>
<td>🟢 $0x66$</td>
</tr>
<tr>
<td>$0x07$ (Type A)</td>
<td>🟢 $0x27$</td>
<td>🟢 $0x47$</td>
<td>🟢 $0x67$</td>
</tr>
<tr>
<td>$0x08$ (Type A)</td>
<td>🟢 $0x28$</td>
<td>🟢 $0x48$</td>
<td>🟢 $0x68$</td>
</tr>
<tr>
<td>$0x09$ (Type A)</td>
<td>🟢 $0x29$</td>
<td>🟢 $0x49$</td>
<td>🟢 $0x69$</td>
</tr>
<tr>
<td>$0x0A$ (Type A)</td>
<td>🟢 $0x2A$</td>
<td>🟢 $0x4A$</td>
<td>🟢 $0x6A$</td>
</tr>
<tr>
<td>$0x0B$ (Type A)</td>
<td>🟢 $0x2B$</td>
<td>🟢 $0x4B$</td>
<td>🟢 $0x6B$</td>
</tr>
<tr>
<td>$0x0C$ (Type A)</td>
<td>🟢 $0x2C$</td>
<td>🟢 $0x4C$</td>
<td>🟢 $0x6C$</td>
</tr>
<tr>
<td>$0x0D$ (Type A)</td>
<td>🟢 $0x2D$</td>
<td>🟢 $0x4D$</td>
<td>🟢 $0x6D$</td>
</tr>
<tr>
<td>$0x0E$ (Type A)</td>
<td>🟢 $0x2E$</td>
<td>🟢 $0x4E$</td>
<td>🟢 $0x6E$</td>
</tr>
<tr>
<td>$0x0F$ (Type A)</td>
<td>🟢 $0x2F$</td>
<td>🟢 $0x4F$</td>
<td>🟢 $0x6F$</td>
</tr>
<tr>
<td>$0x10$ (Type A)</td>
<td>🟢 $0x30$</td>
<td>🟢 $0x50$</td>
<td>🟢 $0x70$</td>
</tr>
<tr>
<td>$0x11$ (Type A)</td>
<td>🟢 $0x31$</td>
<td>🟢 $0x51$</td>
<td>🟢 $0x71$</td>
</tr>
<tr>
<td>$0x12$ (Type A)</td>
<td>🟢 $0x32$</td>
<td>🟢 $0x52$</td>
<td>🟢 $0x72$</td>
</tr>
<tr>
<td>$0x13$ (Type A)</td>
<td>🟢 $0x33$</td>
<td>🟢 $0x53$</td>
<td>🟢 $0x73$</td>
</tr>
<tr>
<td>$0x14$ (Type A)</td>
<td>🟢 $0x34$</td>
<td>🟢 $0x54$</td>
<td>🟢 $0x74$</td>
</tr>
<tr>
<td>$0x15$ (Type A)</td>
<td>🟢 $0x35$</td>
<td>🟢 $0x55$</td>
<td>🟢 $0x75$</td>
</tr>
<tr>
<td>$0x16$ (Type A)</td>
<td>🟢 $0x36$</td>
<td>🟢 $0x56$</td>
<td>🟢 $0x76$</td>
</tr>
<tr>
<td>$0x17$ (Type A)</td>
<td>🟢 $0x37$</td>
<td>🟢 $0x57$</td>
<td>🟢 $0x77$</td>
</tr>
<tr>
<td>$0x18$ (Type A)</td>
<td>🟢 $0x38$</td>
<td>🟢 $0x58$</td>
<td>🟢 $0x78$</td>
</tr>
<tr>
<td>$0x19$ (Type A)</td>
<td>🟢 $0x39$</td>
<td>🟢 $0x59$</td>
<td>🟢 $0x79$</td>
</tr>
<tr>
<td>$0x1A$ (Type A)</td>
<td>🟢 $0x3A$</td>
<td>🟢 $0x5A$</td>
<td>🟢 $0x7A$</td>
</tr>
<tr>
<td>$0x1B$ (Type A)</td>
<td>🟢 $0x3B$</td>
<td>🟢 $0x5B$</td>
<td>🟢 $0x7B$</td>
</tr>
<tr>
<td>$0x1C$ (Type A)</td>
<td>🟢 $0x3C$</td>
<td>🟢 $0x5C$</td>
<td>🟢 $0x7C$</td>
</tr>
<tr>
<td>$0x1D$ (Type A)</td>
<td>🟢 $0x3D$</td>
<td>🟢 $0x5D$</td>
<td>🟢 $0x7D$</td>
</tr>
<tr>
<td>$0x1E$ (Type A)</td>
<td>🟢 $0x3E$</td>
<td>🟢 $0x5E$</td>
<td>🟢 $0x7E$</td>
</tr>
<tr>
<td>$0x1F$ (Type A)</td>
<td>🟢 $0x3F$</td>
<td>🟢 $0x5F$</td>
<td>🟢 $0x7F$</td>
</tr>
</tbody>
</table>

**Type A**

**Type B**

DEL
Consider ASCII character \( M_1 = (0, b_1, b_2, b_3, b_4, b_5, b_6, b_7) \)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{0x00} ) (Type A)</td>
<td>( \text{0x20} )</td>
<td></td>
<td>( \text{0x40} )</td>
<td>@</td>
<td></td>
<td>( \text{0x60} )</td>
</tr>
<tr>
<td>( \text{0x01} ) (Type A)</td>
<td>( \text{0x21} )</td>
<td>!</td>
<td></td>
<td>( \text{0x41} )</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>( \text{0x02} ) (Type A)</td>
<td>( \text{0x22} )</td>
<td>&quot;</td>
<td></td>
<td>( \text{0x42} )</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>( \text{0x03} ) (Type A)</td>
<td>( \text{0x23} )</td>
<td>#</td>
<td></td>
<td>( \text{0x43} )</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>( \text{0x04} ) (Type A)</td>
<td>( \text{0x24} )</td>
<td>$</td>
<td></td>
<td>( \text{0x44} )</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>( \text{0x05} ) (Type A)</td>
<td>( \text{0x25} )</td>
<td>%</td>
<td></td>
<td>( \text{0x45} )</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>( \text{0x06} ) (Type A)</td>
<td>( \text{0x26} )</td>
<td>&amp;</td>
<td></td>
<td>( \text{0x46} )</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>( \text{0x07} ) (Type A)</td>
<td>( \text{0x27} )</td>
<td>'</td>
<td></td>
<td>( \text{0x47} )</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>( \text{0x08} ) (Type A)</td>
<td>( \text{0x28} )</td>
<td>(</td>
<td></td>
<td>( \text{0x48} )</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>( \text{0x09} ) (Type A)</td>
<td>HT</td>
<td></td>
<td>( \text{0x29} )</td>
<td>)</td>
<td></td>
<td>( \text{0x49} )</td>
</tr>
<tr>
<td>( \text{0x0A} )  (Type A)</td>
<td>LF</td>
<td></td>
<td>( \text{0x2A} )</td>
<td>*</td>
<td></td>
<td>( \text{0x4A} )</td>
</tr>
<tr>
<td>( \text{0x0B} ) (Type A)</td>
<td>( \text{0x2B} )</td>
<td>+</td>
<td></td>
<td>( \text{0x4B} )</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>( \text{0x0C} ) (Type A)</td>
<td>( \text{0x2C} )</td>
<td>,</td>
<td></td>
<td>( \text{0x4C} )</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>( \text{0x0D} ) (Type A)</td>
<td>CR</td>
<td></td>
<td>( \text{0x2D} )</td>
<td>-</td>
<td></td>
<td>( \text{0x4D} )</td>
</tr>
<tr>
<td>( \text{0x0E} ) (Type A)</td>
<td>( \text{0x2E} )</td>
<td>.</td>
<td></td>
<td>( \text{0x4E} )</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>( \text{0x0F} ) (Type A)</td>
<td>( \text{0x2F} )</td>
<td>/</td>
<td></td>
<td>( \text{0x4F} )</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>( \text{0x10} ) (Type A)</td>
<td>( \text{0x30} )</td>
<td>0</td>
<td></td>
<td>( \text{0x50} )</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>( \text{0x11} ) (Type A)</td>
<td>( \text{0x31} )</td>
<td>1</td>
<td></td>
<td>( \text{0x51} )</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>( \text{0x12} ) (Type A)</td>
<td>( \text{0x32} )</td>
<td>2</td>
<td></td>
<td>( \text{0x52} )</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>( \text{0x13} ) (Type A)</td>
<td>( \text{0x33} )</td>
<td>3</td>
<td></td>
<td>( \text{0x53} )</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>( \text{0x14} ) (Type A)</td>
<td>( \text{0x34} )</td>
<td>4</td>
<td></td>
<td>( \text{0x54} )</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>( \text{0x15} ) (Type A)</td>
<td>( \text{0x35} )</td>
<td>5</td>
<td></td>
<td>( \text{0x55} )</td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>( \text{0x16} ) (Type A)</td>
<td>( \text{0x36} )</td>
<td>6</td>
<td></td>
<td>( \text{0x56} )</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>( \text{0x17} ) (Type A)</td>
<td>( \text{0x37} )</td>
<td>7</td>
<td></td>
<td>( \text{0x57} )</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>( \text{0x18} ) (Type A)</td>
<td>( \text{0x38} )</td>
<td>8</td>
<td></td>
<td>( \text{0x58} )</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>( \text{0x19} ) (Type A)</td>
<td>( \text{0x39} )</td>
<td>9</td>
<td></td>
<td>( \text{0x59} )</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>( \text{0x1A} ) (Type A)</td>
<td>( \text{0x3A} )</td>
<td>:</td>
<td></td>
<td>( \text{0x5A} )</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>( \text{0x1B} ) (Type A)</td>
<td>( \text{0x3B} )</td>
<td>;</td>
<td></td>
<td>( \text{0x5B} )</td>
<td>[</td>
<td></td>
</tr>
<tr>
<td>( \text{0x1C} ) (Type A)</td>
<td>( \text{0x3C} )</td>
<td>&lt;</td>
<td></td>
<td>( \text{0x5C} )</td>
<td>\</td>
<td></td>
</tr>
<tr>
<td>( \text{0x1D} ) (Type A)</td>
<td>( \text{0x3D} )</td>
<td>=</td>
<td></td>
<td>( \text{0x5D} )</td>
<td>]</td>
<td></td>
</tr>
<tr>
<td>( \text{0x1E} ) (Type A)</td>
<td>( \text{0x3E} )</td>
<td>&gt;</td>
<td></td>
<td>( \text{0x5E} )</td>
<td>^</td>
<td></td>
</tr>
<tr>
<td>( \text{0x1F} ) (Type A)</td>
<td>( \text{0x3F} )</td>
<td>?</td>
<td></td>
<td>( \text{0x5F} )</td>
<td>_</td>
<td></td>
</tr>
</tbody>
</table>
Overview

1. Breaking XML Signature
   - Cloud Computing Management Interfaces
   - Amazon EC2 SOAP Interface
   - XML Signature Wrapping on Eucalyptus and Amazon
   - Countermeasures and Conclusion

2. Breaking XML Encryption
   - Attack Scenario
   - Decrypting by checking plaintext validity
   - Application to CBC mode of operation in XML Encryption
   - Countermeasures and Conclusion

3. Conclusion


Computing $\text{Enc}(M \oplus \text{msk})$ from $\text{Enc}(M)$

- XML Encryption uses block ciphers in *cipher-block chaining* (CBC) mode
- Known weakness of CBC
  - Padding oracle attacks
    (Vaudenay Eurocrypt 2002, and many more)
  - Error oracle attacks
    (Mitchell ISC 2005)
  - Chosen-plaintext attacks on SSL
    (Bard Cryptology ePrint 2004, Duong and Rizzo Ekoparty 2011)
Computing $\text{Enc}(M \oplus \text{msk})$ from $\text{Enc}(M)$

- Transform encryption of $M$ into encryption of $M \oplus \text{msk}$ for arbitrary $\text{msk}$!
- Applicable only to single-block ciphertexts
Multi-block ciphertexts

- CBC: Each block serves as IV for next block
  - Long ciphertexts “consist of many single-block ciphertexts”
- Apply single-block attack to decrypt longer ciphertexts block-wise
  - Decrypt Block 1
  - Decrypt Block 2 with Block 1 as IV
  - Decrypt Block \( i+1 \) with Block \( i \) as IV
Experimental Results

Apache Axis 2, localhost, random plaintexts:

- Timing depends on system performance, network latency, ...
- Approx. 14 server requests/plaintext byte
  - Padding oracle attacks: ca. 128 requests/byte
Improvements and Variations

- XML schema is often public
  - Known structure of XML document
  - Skip blocks containing known plaintext
- Reduced plaintext set
  - Numbers, Base64, “Yes”/“No”, etc.
  - Less plaintext validity checks
- *Encryption* is possible, too
  (Following Rizzo and Duong, Usenix WOOT 2010)
Overview

1. Breaking XML Signature
   - Cloud Computing Management Interfaces
   - Amazon EC2 SOAP Interface
   - XML Signature Wrapping on Eucalyptus and Amazon
   - Countermeasures and Conclusion

2. Breaking XML Encryption
   - Attack Scenario
   - Decrypting by checking plaintext validity
   - Application to CBC mode of operation in XML Encryption
   - Countermeasures and Conclusion

3. Conclusion


Countermeasures

- XML Signature
  - Signature Wrapping attacks
- **Encryption Wrapping attack**: WS-Security Policy says, what must be encrypted...but it says not, what must not be encrypted

Envelope

<table>
<thead>
<tr>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature</td>
</tr>
<tr>
<td>Reference URI=&quot;#signed&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EncryptedKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataReference URI=&quot;#1&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Body Id=&quot;signed&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>EncryptedData Id=&quot;1&quot;</td>
</tr>
</tbody>
</table>

Envelope

<table>
<thead>
<tr>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature</td>
</tr>
<tr>
<td>Reference URI=&quot;#signed&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EncryptedKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataReference URI=&quot;#1&quot;</td>
</tr>
<tr>
<td>DataReference URI=&quot;#2&quot;</td>
</tr>
</tbody>
</table>

| EncryptedData Id="2" |
| Body Id="signed" |
| EncryptedData Id="1" |
Countermeasures

• Authenticated encryption!
  • Not a standard-conformant option
Breaking XML Encryption – Conclusion

- Attack on XML Encryption
  - Applicable in particular to Web Services
  - All major WS frameworks are vulnerable
- No generic \textit{ad-hoc} countermeasure
- W3C plans update of XML Encryption standard
Overview

1. Breaking XML Signature
   - Cloud Computing Management Interfaces
   - Amazon EC2 SOAP Interface
   - XML Signature Wrapping on Eucalyptus and Amazon
   - Countermeasures and Conclusion

2. Breaking XML Encryption
   - Attack Scenario
   - Decrypting by checking plaintext validity
   - Application to CBC mode of operation in XML Encryption
   - Countermeasures and Conclusion

3. Conclusion


Conclusion

• XML Security applies cryptographic primitives on the message level
• It brings advantages in many applications: business process scenarios, Single Sign-On ...
• However, the attacks exist...
  
  1. XML Signature Wrapping: pay attention when applying XML Signatures in your applications
  2. XML Encryption is broken:
    • You can use XML Signatures to ensure authenticity: XML Signature and XML Encryption Wrapping?
    • There are different countermeasures, but they are application and scenario specific
    • New standard with another chaining mode coming soon
Responsible disclosure

• Attack disclosed in Feb 2011 to:
  • W3C, Apache, IBM, RedHat JBoss, Microsoft, governmental CERTs, vendor-sec mailing list, ...
  • All have confirmed that attack is applicable

• Intensive cooperation with some developers
  • More than 100 e-mails since Feb 2011

• In contact with W3C working group
  • Authenticated encryption planned for v2.0
XML Signature Wrapping - Countermeasures

- Amazon Countermeasure + checking for duplicate Ids:
  - Not as easy as it seems to be
  - Id vs ID vs wsu:Id ....?
  - XML Entities?
- See what is signed:
  - Validate signature first
  - Forward only validated document parts
  - XML message becomes not well-formed, could lead to problems e.g. in XML Security Gateways
- Usage of XPath for position fixation
  - Another attacks [Jensen et al.: The curse of namespaces in the domain of xml signature]
CBC and padding in XML Encryption

Encryption

Initialization Vector (IV)

... ...

Ciphertext Block 1

Key → AES

Ciphertext Block 2

Key → AES

Decryption

Padding

03

03

Padding

AES Key

AES Key

AES Key

AES Key
Extracting ‘<’

1. Valid Response: Valid Padding = 0x10
   1. Byte = ‘<’

14 Valid Responses: Valid Paddings = 0x03...0x10
   14. Byte = ‘<’

16 Valid Responses: Valid Paddings = 0x01...0x10
   Set Padding Byte to 0x01

\[ iv_{16} = iv_{16} \oplus 0x01 \]
Difference to Vaudenay’s attack

- Misused security responses caused by incorrect **PKCS** padding
- Vaudenay: IPSEC, SSL, WTLS
- Rizzo and Duong: .NET Framework, JSF View States, Captchas