How to Break XML Encryption – Automatically

Dennis Kupser, Christian Mainka, Jörg Schwenk, Juraj Somorovsky

Ruhr University Bochum

@jurajsomorovsky
About Me and Our Institute

• Security Researcher at:
  – Chair for Network and Data Security
    • Prof. Dr. Jörg Schwenk
    • Web Services, Single Sign-On, (Applied) Crypto, SSL, crypto currencies
    • Provable security, attacks and defenses
  – Horst Görtz Institute for IT-Security
    • Further topics: embedded security, malware, crypto...
  – Ruhr University Bochum

• Penetration tests, security analyses, workshops...
Overview

1. What is a Web Service and XML Security
2. XML Signature Wrapping
3. Attacks on XML Encryption
4. Attacks on Symmetric Encryption Scheme
   1. Attack Scenario
   2. Plaintext Validity
   3. Using Web Service for Plaintext Validation
   4. Decrypting by Checking Plaintext Validity

5. Countermeasures and Problems
6. WS-Attacker
What is a (SOAP) Web Service?

Client

Envelope

Body

getPrime

Server

Envelope

Body

thePrime

11
More complicated scenarios ...
Security?

• SSL / TLS: Transport-Level Security

• Messages are only secured during transport
Motivation – XML Security

• Message Level Security

• Messages protected directly

• XML Security
XML Security

• Methods for cryptographic algorithms in XML
• XML Signature: authenticity and integrity
• XML Encryption: confidentiality
• Flexible

```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 Security Areas

• Financial services:
  – Electronic Banking Internet Communication (EBICS)

• Healthcare:
  – Australian eHealth Technical Specification

• Governmental services:
  – ID cards in Estonia, Germany, Hungary, ...

• System integration, firewalls
Overview

1. What is a Web Service and XML Security
2. XML Signature Wrapping
3. Attacks on XML Encryption
4. Attacks on Symmetric Encryption Scheme
   1. Attack Scenario
   2. Plaintext Validity
   3. Using Web Service for Plaintext Validation
   4. Decrypting by Checking Plaintext Validity
5. Countermeasures and Problems
6. WS-Attacker
How to Break XML Encryption – Automatically
Juraj Somorovsky

XML Signature

Envelope
  Header
  Security
    Signature
      SignedInfo
        Reference
          DigestValue
        Reference
          DigestValue
      SignatureValue

Timestamp
  Id=""Timestamp"

Body
  Id="body"

MonitorInstances
XML Signature Wrapping / Rewriting

McIntosh, Austel (2005)
Bhargavan, Fournet, Gordon, O’Shea (2005)
XML Signature Wrapping

Why does the attack work?
How to Break XML Encryption – Automatically

Juraj Somorovsky

XML Signature Wrapping

Server

Verification logic

Application logic

How to Break XML Encryption – Automatically

Juraj Somorovsky
XML Signature Wrapping

- Attacks on Amazon EC2 / Eucalyptus clouds
### Further Attacks: SAML

<table>
<thead>
<tr>
<th>Service Provider</th>
<th>0Sig</th>
<th>CF</th>
<th>XXEA</th>
<th>XSLTA</th>
<th>RA</th>
<th>XSW</th>
<th>TRC</th>
<th>AM3</th>
<th>CInj</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salesforce</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Google Apps</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Zoho</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Zendesk</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Clarizen</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>SAManage</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Shiftplanning</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Panorama9</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>UserVoice (Marketing)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Instructure</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>The Resumator</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>BambooHR</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>AppDynamics</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>IdeaScale</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Panopto</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>TimeOffManager</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>HappyFox</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>SpringCM</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ScreenSteps Live</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>LiveHive</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Howlr</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>CA Service Management</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>6</td>
<td>11</td>
<td>17</td>
<td>6</td>
<td>20/22</td>
<td></td>
</tr>
</tbody>
</table>


*How to Break XML Encryption – Automatically*  Juraj Somorovsky
Overview

1. What is a Web Service and XML Security
2. XML Signature Wrapping
3. Attacks on XML Encryption
4. Attacks on Symmetric Encryption Scheme
   1. Attack Scenario
   2. Plaintext Validity
   3. Using Web Service for Plaintext Validation
   4. Decrypting by Checking Plaintext Validity
5. Countermeasures and Problems
6. WS-Attacker
XML Encryption

Hybrid encryption scheme

1. Asymmetric encryption / decryption
   - URI="#enc"
   - Algorithm="...#rsa-1_5"

2. Symmetric encryption / decryption
   - Id="enc"
   - Algorithm="...#aes128-cbc"
Attacks on XML Encryption

• Attacks on EncryptedKey
  – Bleichenbacher’s Attack Strikes Again: Breaking PKCS#1 v1.5 in XML Encryption.
    Tibor Jager, Sebastian Schinzel, Juraj Somorovsky. ESORICS 2012

• Attacks on EncryptedData
  – How to Break XML Encryption.
    Tibor Jager, Juraj Somorovsky. CCS 2011

Adaptive chosen-ciphertext attacks
Adaptive chosen-ciphertext attack

XML Encryption ciphertext $C = \text{Enc}(M)$

Chosen ciphertext $C_1$

valid/invalid

Chosen ciphertext $C_2$

valid/invalid

\[ \ldots \]

(repeated several times)

<table>
<thead>
<tr>
<th>Encryption</th>
<th>CCS 2011</th>
<th>ESORICS 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-Queries</td>
<td>14 / plaintext byte</td>
<td>400k to 82M / key</td>
</tr>
<tr>
<td>Encryption</td>
<td>symmetric</td>
<td>asymmetric</td>
</tr>
</tbody>
</table>

How to Break XML Encryption — Automatically  Juraj Somorovsky
Overview

1. What is a Web Service and XML Security
2. XML Signature Wrapping
3. Attacks on XML Encryption
4. Attacks on Symmetric Encryption Scheme
   1. Attack Scenario
   2. Plaintext Validity
   3. Using Web Service for Plaintext Validation
   4. Decrypting by Checking Plaintext Validity
5. Countermeasures and Problems
6. WS-Attacker
**Attack Scenario**

XML Encryption ciphertext $C = \text{Enc}(M)$

- Chosen ciphertext $C_1$
  - **valid/invalid** plaintext
- Chosen ciphertext $C_2$
  - **valid/invalid** plaintext
- $M = \text{Dec}(C)$
- (repeated several times)

- What is a “valid” plaintext?
- How to use Web Service as “plaintext validity oracle”?
- How to use this oracle to decrypt $C$?
Overview

1. What is a Web Service and XML Security
2. XML Signature Wrapping
3. Attacks on XML Encryption
4. Attacks on Symmetric Encryption Scheme
   1. Attack Scenario
   2. Plaintext Validity
   3. Using Web Service for Plaintext Validation
   4. Decrypting by Checking Plaintext Validity
5. Countermeasures and Problems
6. WS-Attacker
Plaintext Validity

• XML is a text-based data format
• XML parsing
• Characters (usually) encoded in ASCII
### ASCII

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
</tr>
<tr>
<td>1</td>
<td>!</td>
</tr>
<tr>
<td>2</td>
<td>&quot;</td>
</tr>
<tr>
<td>3</td>
<td>#</td>
</tr>
<tr>
<td>4</td>
<td>$</td>
</tr>
<tr>
<td>5</td>
<td>%</td>
</tr>
<tr>
<td>6</td>
<td>&amp;</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
</tr>
<tr>
<td>8</td>
<td>(</td>
</tr>
<tr>
<td>9</td>
<td>)</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>+</td>
</tr>
<tr>
<td>12</td>
<td>,</td>
</tr>
<tr>
<td>13</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>.</td>
</tr>
<tr>
<td>15</td>
<td>/</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>21</td>
<td>5</td>
</tr>
<tr>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>23</td>
<td>7</td>
</tr>
<tr>
<td>24</td>
<td>8</td>
</tr>
<tr>
<td>25</td>
<td>9</td>
</tr>
<tr>
<td>26</td>
<td>A</td>
</tr>
<tr>
<td>27</td>
<td>B</td>
</tr>
<tr>
<td>28</td>
<td>C</td>
</tr>
<tr>
<td>29</td>
<td>D</td>
</tr>
<tr>
<td>30</td>
<td>E</td>
</tr>
<tr>
<td>31</td>
<td>F</td>
</tr>
<tr>
<td>32</td>
<td>G</td>
</tr>
<tr>
<td>33</td>
<td>H</td>
</tr>
<tr>
<td>34</td>
<td>I</td>
</tr>
<tr>
<td>35</td>
<td>J</td>
</tr>
<tr>
<td>36</td>
<td>K</td>
</tr>
<tr>
<td>37</td>
<td>L</td>
</tr>
<tr>
<td>38</td>
<td>M</td>
</tr>
<tr>
<td>39</td>
<td>N</td>
</tr>
<tr>
<td>40</td>
<td>O</td>
</tr>
<tr>
<td>41</td>
<td>P</td>
</tr>
<tr>
<td>42</td>
<td>Q</td>
</tr>
<tr>
<td>43</td>
<td>R</td>
</tr>
<tr>
<td>44</td>
<td>S</td>
</tr>
<tr>
<td>45</td>
<td>T</td>
</tr>
<tr>
<td>46</td>
<td>U</td>
</tr>
<tr>
<td>47</td>
<td>V</td>
</tr>
<tr>
<td>48</td>
<td>W</td>
</tr>
<tr>
<td>49</td>
<td>X</td>
</tr>
<tr>
<td>50</td>
<td>Y</td>
</tr>
<tr>
<td>51</td>
<td>Z</td>
</tr>
<tr>
<td>52</td>
<td>[</td>
</tr>
<tr>
<td>53</td>
<td>]</td>
</tr>
<tr>
<td>54</td>
<td>^</td>
</tr>
<tr>
<td>55</td>
<td>_</td>
</tr>
<tr>
<td>56</td>
<td>`</td>
</tr>
<tr>
<td>57</td>
<td>a</td>
</tr>
<tr>
<td>58</td>
<td>b</td>
</tr>
<tr>
<td>59</td>
<td>c</td>
</tr>
<tr>
<td>60</td>
<td>d</td>
</tr>
<tr>
<td>61</td>
<td>e</td>
</tr>
<tr>
<td>62</td>
<td>f</td>
</tr>
<tr>
<td>63</td>
<td>g</td>
</tr>
<tr>
<td>64</td>
<td>h</td>
</tr>
<tr>
<td>65</td>
<td>i</td>
</tr>
<tr>
<td>66</td>
<td>j</td>
</tr>
<tr>
<td>67</td>
<td>k</td>
</tr>
<tr>
<td>68</td>
<td>l</td>
</tr>
<tr>
<td>69</td>
<td>m</td>
</tr>
<tr>
<td>70</td>
<td>n</td>
</tr>
<tr>
<td>71</td>
<td>o</td>
</tr>
<tr>
<td>72</td>
<td>p</td>
</tr>
<tr>
<td>73</td>
<td>q</td>
</tr>
<tr>
<td>74</td>
<td>r</td>
</tr>
<tr>
<td>75</td>
<td>s</td>
</tr>
<tr>
<td>76</td>
<td>t</td>
</tr>
<tr>
<td>77</td>
<td>u</td>
</tr>
<tr>
<td>78</td>
<td>v</td>
</tr>
<tr>
<td>79</td>
<td>w</td>
</tr>
<tr>
<td>80</td>
<td>x</td>
</tr>
<tr>
<td>81</td>
<td>y</td>
</tr>
<tr>
<td>82</td>
<td>z</td>
</tr>
<tr>
<td>83</td>
<td>{</td>
</tr>
<tr>
<td>84</td>
<td>}</td>
</tr>
<tr>
<td>85</td>
<td>~</td>
</tr>
</tbody>
</table>

**Not Parsable:**

- Type A
- Type B

**Parsable:**

- Type B

"Valid" Plaintext contains only Type B characters
Overview

1. What is a Web Service and XML Security
2. XML Signature Wrapping
3. Attacks on XML Encryption
4. Attacks on Symmetric Encryption Scheme
   1. Attack Scenario
   2. Plaintext Validity
   3. Using Web Service for Plaintext Validation
   4. Decrypting by Checking Plaintext Validity
5. Countermeasures and Problems
6. WS-Attacker
Validity Oracle

- Using Web Services Server as plaintext validity oracle

1) Content Decryption
2) XML Parsing
3) XML Evaluation

- Invalid plaintext => Parsing error
- Parsing error => Fault message (or another side channel)
Overview

1. What is a Web Service and XML Security
2. XML Signature Wrapping
3. Attacks on XML Encryption
4. Attacks on Symmetric Encryption Scheme
   1. Attack Scenario
   2. Plaintext Validity
   3. Using Web Service for Plaintext Validation
   4. Decrypting by Checking Plaintext Validity
5. Countermeasures and Problems
6. WS-Attacker
Consider ASCII character \( M_1 = (0, b_1, b_2, b_3, b_4, b_5, b_6, b_7) \)

<table>
<thead>
<tr>
<th>ASCII</th>
<th>Type A</th>
<th>Type B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>NUL</td>
<td>0x20</td>
</tr>
<tr>
<td>0x01</td>
<td>(Type A)</td>
<td>0x21</td>
</tr>
<tr>
<td>0x02</td>
<td>(Type A)</td>
<td>0x22</td>
</tr>
<tr>
<td>0x03</td>
<td>(Type A)</td>
<td>0x23</td>
</tr>
<tr>
<td>0x04</td>
<td>(Type A)</td>
<td>0x24</td>
</tr>
<tr>
<td>0x05</td>
<td>(Type A)</td>
<td>0x25</td>
</tr>
<tr>
<td>0x06</td>
<td>(Type A)</td>
<td>0x26</td>
</tr>
<tr>
<td>0x07</td>
<td>BEL</td>
<td>0x27</td>
</tr>
<tr>
<td>0x08</td>
<td>BS</td>
<td>0x28</td>
</tr>
<tr>
<td>0x09</td>
<td>HT</td>
<td>0x29</td>
</tr>
<tr>
<td>0x0A</td>
<td>LF</td>
<td>0x2A</td>
</tr>
<tr>
<td>0x0B</td>
<td>(Type A)</td>
<td>0x2B</td>
</tr>
<tr>
<td>0x0C</td>
<td>(Type A)</td>
<td>0x2C</td>
</tr>
<tr>
<td>0x0D</td>
<td>CR</td>
<td>0x2D</td>
</tr>
<tr>
<td>0x0E</td>
<td>(Type A)</td>
<td>0x2E</td>
</tr>
<tr>
<td>0x0F</td>
<td>(Type A)</td>
<td>0x2F</td>
</tr>
<tr>
<td>0x10</td>
<td>(Type A)</td>
<td>0x30</td>
</tr>
<tr>
<td>0x11</td>
<td>(Type A)</td>
<td>0x31</td>
</tr>
<tr>
<td>0x12</td>
<td>(Type A)</td>
<td>0x32</td>
</tr>
<tr>
<td>0x13</td>
<td>(Type A)</td>
<td>0x33</td>
</tr>
<tr>
<td>0x14</td>
<td>(Type A)</td>
<td>0x34</td>
</tr>
<tr>
<td>0x15</td>
<td>(Type A)</td>
<td>0x35</td>
</tr>
<tr>
<td>0x16</td>
<td>(Type A)</td>
<td>0x36</td>
</tr>
<tr>
<td>0x17</td>
<td>(Type A)</td>
<td>0x37</td>
</tr>
<tr>
<td>0x18</td>
<td>(Type A)</td>
<td>0x38</td>
</tr>
<tr>
<td>0x19</td>
<td>(Type A)</td>
<td>0x39</td>
</tr>
<tr>
<td>0x1A</td>
<td>(Type A)</td>
<td>0x3A</td>
</tr>
<tr>
<td>0x1B</td>
<td>ESC</td>
<td>0x3B</td>
</tr>
<tr>
<td>0x1C</td>
<td>(Type A)</td>
<td>0x3C</td>
</tr>
<tr>
<td>0x1D</td>
<td>(Type A)</td>
<td>0x3D</td>
</tr>
<tr>
<td>0x1E</td>
<td>(Type A)</td>
<td>0x3E</td>
</tr>
<tr>
<td>0x1F</td>
<td>(Type A)</td>
<td>0x3F</td>
</tr>
</tbody>
</table>
Decrypting by checking plaintext validity

• ASCII exhibits nice pattern of Type A/B characters

• Suppose we can \textbf{flip} arbitrary plaintext bits and use a plaintext validity \textbf{oracle}

• What could go wrong?
Decrypting by checking plaintext validity

• Example

• We have eavesdropped a ciphertext
  \( C = \text{Enc}("Deepsec") \)

• How to determine \((b_1, b_2)\) of \(M_1 = "D"\)?
Consider ASCII character $M_1 = (0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$

<table>
<thead>
<tr>
<th>ASCII Code</th>
<th>Character</th>
<th>ASCII Code</th>
<th>Character</th>
<th>ASCII Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>NUL</td>
<td>0x20</td>
<td>@</td>
<td>0x60</td>
<td>'</td>
</tr>
<tr>
<td>0x01</td>
<td>(Type A)</td>
<td>0x21</td>
<td>!</td>
<td>0x61</td>
<td>a</td>
</tr>
<tr>
<td>0x02</td>
<td>(Type A)</td>
<td>0x22</td>
<td>&quot;</td>
<td>0x62</td>
<td>b</td>
</tr>
<tr>
<td>0x03</td>
<td>(Type A)</td>
<td>0x23</td>
<td>#</td>
<td>0x63</td>
<td>c</td>
</tr>
<tr>
<td>0x04</td>
<td>(Type A)</td>
<td>0x24</td>
<td>$</td>
<td>0x64</td>
<td>d</td>
</tr>
<tr>
<td>0x05</td>
<td>(Type A)</td>
<td>0x26</td>
<td>&amp;</td>
<td>0x65</td>
<td>e</td>
</tr>
<tr>
<td>0x06</td>
<td>(Type A)</td>
<td>0x27</td>
<td>'</td>
<td>0x66</td>
<td>f</td>
</tr>
<tr>
<td>0x07</td>
<td>BEL</td>
<td>0x28</td>
<td>(</td>
<td>0x67</td>
<td>g</td>
</tr>
<tr>
<td>0x09</td>
<td>HT</td>
<td>0x29</td>
<td>)</td>
<td>0x68</td>
<td>h</td>
</tr>
<tr>
<td>0x0A</td>
<td>LF</td>
<td>0x2A</td>
<td>)</td>
<td>0x69</td>
<td>i</td>
</tr>
<tr>
<td>0x0B</td>
<td>(Type A)</td>
<td>0x2B</td>
<td>+</td>
<td>0x6A</td>
<td>j</td>
</tr>
<tr>
<td>0x0C</td>
<td>(Type A)</td>
<td>0x2C</td>
<td>,</td>
<td>0x6B</td>
<td>k</td>
</tr>
<tr>
<td>0x0D</td>
<td>CR</td>
<td>0x2D</td>
<td>-</td>
<td>0x6C</td>
<td>l</td>
</tr>
<tr>
<td>0x0E</td>
<td>(Type A)</td>
<td>0x2E</td>
<td>.</td>
<td>0x6D</td>
<td>m</td>
</tr>
<tr>
<td>0x0F</td>
<td>(Type A)</td>
<td>0x2F</td>
<td>/</td>
<td>0x6E</td>
<td>n</td>
</tr>
<tr>
<td>0x10</td>
<td>(Type A)</td>
<td>0x30</td>
<td>0</td>
<td>0x6F</td>
<td>o</td>
</tr>
<tr>
<td>0x11</td>
<td>(Type A)</td>
<td>0x31</td>
<td>1</td>
<td>0x70</td>
<td>p</td>
</tr>
<tr>
<td>0x12</td>
<td>(Type A)</td>
<td>0x32</td>
<td>2</td>
<td>0x71</td>
<td>q</td>
</tr>
<tr>
<td>0x13</td>
<td>(Type A)</td>
<td>0x33</td>
<td>3</td>
<td>0x72</td>
<td>r</td>
</tr>
<tr>
<td>0x14</td>
<td>(Type A)</td>
<td>0x34</td>
<td>4</td>
<td>0x73</td>
<td>s</td>
</tr>
<tr>
<td>0x15</td>
<td>(Type A)</td>
<td>0x35</td>
<td>5</td>
<td>0x74</td>
<td>t</td>
</tr>
<tr>
<td>0x16</td>
<td>(Type A)</td>
<td>0x36</td>
<td>6</td>
<td>0x75</td>
<td>u</td>
</tr>
<tr>
<td>0x17</td>
<td>(Type A)</td>
<td>0x37</td>
<td>7</td>
<td>0x76</td>
<td>v</td>
</tr>
<tr>
<td>0x18</td>
<td>(Type A)</td>
<td>0x38</td>
<td>8</td>
<td>0x77</td>
<td>w</td>
</tr>
<tr>
<td>0x19</td>
<td>(Type A)</td>
<td>0x39</td>
<td>9</td>
<td>0x78</td>
<td>x</td>
</tr>
<tr>
<td>0x1A</td>
<td>(Type A)</td>
<td>0x3A</td>
<td>:</td>
<td>0x79</td>
<td>y</td>
</tr>
<tr>
<td>0x1B</td>
<td>ESC</td>
<td>0x3B</td>
<td>;</td>
<td>0x7A</td>
<td>z</td>
</tr>
<tr>
<td>0x1C</td>
<td>(Type A)</td>
<td>0x3C</td>
<td>&lt;</td>
<td>0x7B</td>
<td>{</td>
</tr>
<tr>
<td>0x1D</td>
<td>(Type A)</td>
<td>0x3D</td>
<td>=</td>
<td>0x7C</td>
<td></td>
</tr>
<tr>
<td>0x1E</td>
<td>(Type A)</td>
<td>0x3E</td>
<td>&gt;</td>
<td>0x7D</td>
<td>}</td>
</tr>
<tr>
<td>0x1F</td>
<td>(Type A)</td>
<td>0x3F</td>
<td>?</td>
<td>0x7E</td>
<td>~</td>
</tr>
</tbody>
</table>
Consider ASCII character $M_1 = (0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$

<table>
<thead>
<tr>
<th>ASCII Value</th>
<th>Character</th>
<th>Type A</th>
<th>Type B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>NUL</td>
<td>0x20</td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td>(Type A)</td>
<td>0x21</td>
<td>!</td>
</tr>
<tr>
<td>0x02</td>
<td>(Type A)</td>
<td>0x22</td>
<td>&quot;</td>
</tr>
<tr>
<td>0x03</td>
<td>(Type A)</td>
<td>0x23</td>
<td>#</td>
</tr>
<tr>
<td>0x04</td>
<td>(Type A)</td>
<td>0x24</td>
<td>$</td>
</tr>
<tr>
<td>0x05</td>
<td>(Type A)</td>
<td>0x25</td>
<td>%</td>
</tr>
<tr>
<td>0x06</td>
<td>(Type A)</td>
<td>0x26</td>
<td>&amp;</td>
</tr>
<tr>
<td>0x07</td>
<td>BEL</td>
<td>0x27</td>
<td>।</td>
</tr>
<tr>
<td>0x08</td>
<td>BS</td>
<td>0x28</td>
<td>(</td>
</tr>
<tr>
<td>0x09</td>
<td>HT</td>
<td>0x29</td>
<td>)</td>
</tr>
<tr>
<td>0x0A</td>
<td>LF</td>
<td>0x2A</td>
<td>*</td>
</tr>
<tr>
<td>0x0B</td>
<td>(Type A)</td>
<td>0x2B</td>
<td>+</td>
</tr>
<tr>
<td>0x0C</td>
<td>(Type A)</td>
<td>0x2C</td>
<td>,</td>
</tr>
<tr>
<td>0x0D</td>
<td>CR</td>
<td>0x2D</td>
<td>-</td>
</tr>
<tr>
<td>0x0E</td>
<td>(Type A)</td>
<td>0x2E</td>
<td>.</td>
</tr>
<tr>
<td>0x0F</td>
<td>(Type A)</td>
<td>0x2F</td>
<td>/</td>
</tr>
<tr>
<td>0x10</td>
<td>(Type A)</td>
<td>0x30</td>
<td>0</td>
</tr>
<tr>
<td>0x11</td>
<td>(Type A)</td>
<td>0x31</td>
<td>1</td>
</tr>
<tr>
<td>0x12</td>
<td>(Type A)</td>
<td>0x32</td>
<td>2</td>
</tr>
<tr>
<td>0x13</td>
<td>(Type A)</td>
<td>0x33</td>
<td>3</td>
</tr>
<tr>
<td>0x14</td>
<td>(Type A)</td>
<td>0x34</td>
<td>4</td>
</tr>
<tr>
<td>0x15</td>
<td>(Type A)</td>
<td>0x35</td>
<td>5</td>
</tr>
<tr>
<td>0x16</td>
<td>(Type A)</td>
<td>0x36</td>
<td>6</td>
</tr>
<tr>
<td>0x17</td>
<td>(Type A)</td>
<td>0x37</td>
<td>7</td>
</tr>
<tr>
<td>0x18</td>
<td>(Type A)</td>
<td>0x38</td>
<td>8</td>
</tr>
<tr>
<td>0x19</td>
<td>(Type A)</td>
<td>0x39</td>
<td>9</td>
</tr>
<tr>
<td>0x1A</td>
<td>(Type A)</td>
<td>0x3A</td>
<td>:</td>
</tr>
<tr>
<td>0x1B</td>
<td>ESC</td>
<td>0x3B</td>
<td>;</td>
</tr>
<tr>
<td>0x1C</td>
<td>(Type A)</td>
<td>0x3C</td>
<td>&lt;</td>
</tr>
<tr>
<td>0x1D</td>
<td>(Type A)</td>
<td>0x3D</td>
<td>=</td>
</tr>
<tr>
<td>0x1E</td>
<td>(Type A)</td>
<td>0x3E</td>
<td>&gt;</td>
</tr>
<tr>
<td>0x1F</td>
<td>(Type A)</td>
<td>0x3F</td>
<td>?</td>
</tr>
</tbody>
</table>
Performance

• 14 queries / byte
Why Flipping Possible?
Cipher Block Chaining Mode

- Flip arbitrary bits in plaintext
- Applied in padding oracle attacks
Overview

1. What is a Web Service and XML Security
2. XML Signature Wrapping
3. Attacks on XML Encryption
4. Attacks on Symmetric Encryption Scheme
   1. Attack Scenario
   2. Plaintext Validity
   3. Using Web Service for Plaintext Validation
   4. Decrypting by Checking Plaintext Validity
5. Countermeasures and Problems
6. WS-Attacker
Basic Idea

• Protect integrity and authenticity of ciphertexts

• Not easy...
XML Signature Wrapping?

Signature validation

Decryption
XML Encryption Wrapping?

- Envelope
- Header
- Security
  - Signature
    - Reference
      - URI="#body"
  - EncryptedKey
  - DataReference
    - URI="#oracle"
  - EncryptedData
    - Id="oracle"
  - CipherData
  - EncryptedKey
    - DataReference
      - URI="#enc"

- Body
  - Id="body"
  - EncryptedData
    - Id="enc"
  - CipherData

- Signature validation
- Decryption
- Decryption and Business logic
How to analyze Web Services Automatically?
Overview

1. What is a Web Service and XML Security
2. XML Signature Wrapping
3. Attacks on XML Encryption
4. Attacks on Symmetric Encryption Scheme
   1. Attack Scenario
   2. Plaintext Validity
   3. Using Web Service for Plaintext Validation
   4. Decrypting by Checking Plaintext Validity
5. Countermeasures and Problems
6. WS-Attacker
WS-Attacker

• Automatic penetration test tool for Web Services
• Supports many attacks
  – New plugin for XML Encryption

• Approach:
  1. Analyze message security
  2. Remove signature protection
  3. Attack (symmetric / asymmetric)
Automated Attack Workflow

Encrypted XML

Detection Phase

Avoid Phase

Attack Phase

Identify Security Elements
Detection Phase (Offline)

- Envelope
  - Header
  - Security
    - EncryptedKey
      - URI="#a"
    - Signature
      - URI="#b"
    - Signature
      - URI="#c"
    - Timestamp
      - Id="c"
- Body
  - EncryptedData
    - Id="a"
Automated Attack Workflow

- **Detection Phase**
  - Identify Security Elements
  - Encrypted XML

- **Avoid Phase**
  - Signed Timestamp?

- **Attack Phase**

Knowledge Pool
Detection Phase (Offline)

- **Envelope**
  - **Header**
  - **Security**
    - **EncryptedKey** URI="#a"
    - **Signature** URI="#b"
    - **Signature** URI="#c"
    - **Timestamp** Id="c"
- **Body** Id="b"
- **EncryptedData** Id="a"
Automated Attack Workflow

Knowledge Pool

Encrypted XML

Detect Security Elements

Signed Timestamp?

yes

xSW

Detection Phase

Avoid Phase

Attack Phase
Applying XSW - Complexity
Automated Attack Workflow

Detect Phase

Avoid Phase

Encrypted XML

Identify Security Elements

Signed Timestamp?

Signed Encrypted Element

Oracle

XSW

XSW

XEW

Identify

yes

no

yes

no

yes

no

yes

no

fail

fail

Detected Phase
Identify Oracle

• Map Server Responses to „valid“ or „invalid“

Chosen ciphertext <ok/>

Chosen ciphertext <failure/>

• Implementation dependent!
Automated Attack Workflow

1. **Detection Phase**
   - **Identify Security Elements**
     - **Encrypted XML**
     - **Signed Timestamp?**
       - **yes**
         - **xsw**
       - **no**
         - **Signed Encrypted Element**
           - **yes**
             - **xsw**
           - **no**
             - **Identify Oracle**
               - **yes**
                 - **Apply Attack**
                 - **Decrypted XML**
               - **no**
                 - **fail**

2. **Avoid Phase**
   - **XEW**
   - **fail**

3. **Attack Phase**
   - **Decrypted XML**
   - **fail**
# Results

<table>
<thead>
<tr>
<th>System</th>
<th>Asymmetric Attack</th>
<th>Symmetric Attack</th>
<th>Countermeasures applicable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Axis2 1.6.2</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Apache CXF 2.7.10</td>
<td>✓</td>
<td>✓</td>
<td>yes</td>
</tr>
<tr>
<td>Axway Gateway 7.3.1</td>
<td>✓</td>
<td>✓</td>
<td>yes</td>
</tr>
<tr>
<td>IBM Datapower XI50</td>
<td>✓</td>
<td>✓</td>
<td>yes</td>
</tr>
<tr>
<td>Microsoft WCF</td>
<td></td>
<td></td>
<td>yes</td>
</tr>
</tbody>
</table>
Load WSDL
How to Break XML Encryption – Automatically

Juraj Somorovsky

Send Test Request

File

WSDL Loader  Test Request  Plugin Configuration  Attack Overview  Log  Expert View  Configuration

URL Endpoint: http://

decrypt

XML Request  Additional HTTP Request Headers

2.   <soapenv:Header>
4.       <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
5.         <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" xmlns:ds
6.         <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
7.           <wsse:SecurityTokenReference>
8.             <wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-2k
9.               </wsse:SecurityTokenReference>
10.       </dsig:KeyInfo>
11.     </xenc:CipherData>
12.     <xenc:CipherValue>pTm4u6vay+exrUdf33Z59UZpnqyR0tgYnTqRffOGTkcAhdbajVliili0mFKK6c
13.     </xenc:CipherData>
14.   </xenc:ReferenceList>
15.   <xenc:DataReference URI="#body"/>
16. </xenc:ReferenceList>
17. </xenc:EncryptedKey>
18. </wsse:Security>

XML Response  HTTP Response Headers

2.   <soapenv:Header>
4.   </soapenv:Header>
Choose Attacks

File

- WSDL Loader
- Test Request
- Plugin Config
- Attack Overview
- Log
- Expert View

Active Plugins (1)
- Denial of Service (6)
  - Coercive Parsing (Ready)
  - Hash DOS Attack (Ready)
  - Oversized XML (Ready)
  - SOAP array attack (Ready)
  - Test DOS Attack (Ready)
  - XML Entity Expansion (recursive) (Ready)
- Security (1)
  - Signature (1)
    - Signature Wrapping (Ready)
- Spoofing Attacks (2)
  - SOAP Action Spoofing (Ready)
  - WS-Addressing Spoofing (Ready)
- Test (2)
- Alphabetical Sorted (16)

Signature Wrapping

Schema?
- Turn on, to not use any XML Schema.

Used Schema Files
- Set the Schema Files.
  - Soap11, Soap12, WSA, WSSE, WSU, DS and XPathFilter2 are included by default.

Search?
- SOAP Response must contain a specific String.

View?
- Display the wrapping messages.

Show
- Payload #1

Reference Element

  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n-
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
    <ds:Reference URI="#_13c5cf5aaa49075df9872ad43d5f936"/>
    <ds:Transforms>
      <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-sign
      <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    </ds:Transforms>
  </ds:Signature>
</ds:SignedInfo>

Timestamp?
- Yes

URL: #_13c5cf5aaa49075df9872ad43d5f936

Analyzing:

INFO] Has payload? false
Configure XML Encryption

Detected Encrypted Elements:

**Elements:**
- EncryptedKey
- TimeStamp

**Configuration:**
- Attack: CBC_ATTACK
- Oracle Type: ERROR_ORACLE
- Wrapping Attack: NO_WRAP
- StringCompare: DICE_COEFF
- Threshold Wrap Error: 0.9
- Threshold General Error: 0.9

EncryptedKey:
- isAttackPayload
- isSigned
- isAddWrap
- PKCS1 Strategy: CBC_WEAK

EncryptedData:
- 1/1
- isAttackPayload
- isSigned
- isAddWrap
- useTypeWeakness

Encoded XML:
```
<EncryptedData>
  <xenc:EncryptedKey>
    <xenc:KeyInfo>
      <xenc:EncryptedKey>
        <xenc:CipherValue>
          JXVBFonTrgKzbpBNNTwuyq1upPahyevvdwCDgZo1+Smf5s72pF1WZ8s39e8dugWklukjK5L/341ylXWk20qrZhU3c2h4Lo9XWnauN1calm4wriCoCg6ys1AEfc5QrKQ29CIjFXPTJTLolg1xj25eWz+JhPnyaBvgZl2wZK6O7dJJ3AmUItq43NFg3mwey205/1SEhvoXprXhx==</xenc:CipherValue>
        </xenc:EncryptedData>
      </xenc:EncryptedKey>
    </xenc:KeyInfo>
  </xenc:EncryptedKey>
</EncryptedData>
```

Encoded Base64:
```
"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
  <wss:SecurityTokenReference>
    <wss:KeyIdentifier EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssec-secext-1.0.xsd">"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
    </wss:KeyInfo>
  </wss:SecurityTokenReference>
</wss:EncryptionMethod>
```

How to Break XML Encryption – Automatically  Juraj Somorovsky
Start Attack

<table>
<thead>
<tr>
<th>Time</th>
<th>Level</th>
<th>Source</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>13:42:58.252</td>
<td>Info</td>
<td>XML-Encryption Attack</td>
<td>Plantext of encrypted data:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>xmlns:echo=&quot;&quot; xmlns:sam=&quot;<a href="http://sample01.samples.rampart.apache.org">http://sample01.samples.rampart.apache.org</a>&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>xmlns:soapenv=&quot;<a href="http://schemas.xmlsoap.org/soap/envelope/">http://schemas.xmlsoap.org/soap/envelope/</a>&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;-Optional:-&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><a href="">sam:arg0</a>/?&lt;/sam:arg0&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;/sam:echo&gt;</td>
</tr>
<tr>
<td>13:51:37.529</td>
<td>Info</td>
<td>XML-Encryption Attack</td>
<td>Number of Oracle Queries: 4855</td>
</tr>
</tbody>
</table>
Case Apache CXF

1. Asymmetric decryption
2. Symmetric decryption

CVE-2015-0226
CVE-2015-0227
Case Apache CXF: Symmetric

Envelope
  - Header
  - Security
    - EncryptedKey
      - EncryptionMethod: #rsa-1_5
    - CipherData
  - Body
    - EncryptedData
      - EncryptionMethod: #aes128-cbc
      - CipherData

Signature enforced with:
requireSignedEncryptedDataElements = "true"

CVE-2015-0226
CVE-2015-0227

Symmetric Decryption

Signature Wrapping
Case Apache CXF: Asymmetric

CVE-2015-0226
CVE-2015-0227
Case Apache CXF: Asymmetric

CVE-2015-0226
CVE-2015-0227

Invalid countermeasure
Playing with WS-Attacker

- Use Apache Axis2 and Apache Rampart
- Examples:
  - http://web-in-security.blogspot.de/
Countermeasures

• AES-CBC, RSA-PKCS#1 v1.5 insecure!
• XML Encryption updated (Version 1.1)
  • AES-GCM added
• Use of secure algorithms: RSA-OAEP, AES-GCM
• If secure algorithms not available, only decrypt signed XML ciphertexts
  • Example: IBM Datapower
Conclusion

• XML – especially XML Security – is complex

• WS-Attacker for evaluation:

• Our approach applicable to other scenarios
  – SAML, JSON, Web Crypto...

• Prefer authenticated encryption (AES-GCM instead of AES-CBC)