Attacks against Web Services and Countermeasures

Lecture "XML in Communication Systems"
Chapter 18

Dr.-Ing. Jesper Zedlitz
Research Group for Communication Systems
Dept. of Computer Science
Christian-Albrechts-University in Kiel
Recommended Reading

- Nils Gruschka:
  *Schutz von Web Services durch erweiterte und effiziente Nachrichtenvalidierung.*
  Göttingen (Cuvillier), 2008.

- Michael McIntosh, Paula Austel:
  XML Signature Element Wrapping Attacks and Countermeasures.
  In: Proceedings of the 2005 Workshop on Secure Web Services, 2005,
  Fairfax, VA, USA, 20–27
Overview

1. Attacks
2. Countermeasures: Concept
3. Message Schema Validation
4. Security Policy Validation
5. Message Sequence Validation
6. Evaluation
Chapter 18.1

Attacks
Security Model for Web Services

• A closer view on checking/filtering

Solutions for Web Services:
  ○ SOAP header inspection
  ○ WS message content analysis
Web Service Protocol Stack

• A layered view

<Envelope>
  <Body>
    <add>
      <a>12</a>
      <b>38</b>
      <c>27</c>
    </add>
  </Body>
</Envelope>

POST /Service.asmx
HTTP/1.1
Content-Type: text/xml
Content-Length: 523

Destination Port: 80
Destination Address 10.1.1.1

e.g. Cookie Poisoning

e.g. SYN Flooding
Attacks against Web Services

• "Web Services are exposed to the same common attacks as all other networked services."
  
  – Fraud:
    different kinds of attacks against message confidentiality, integrity, authenticity
  
  – DoS attacks:
    attacker tries to exhaust victim's resources

Is it different with Web Services?
Fraud Attacks

• Signature Wrapping

McIntosh and Austel, 2005
Attacks against Web Services

- Denial of Service (DoS) attacks
  - Attempt to block availability
  - Exploiting weakness within implementation
  - Protocol deviation
  - Resource exhaustion
  - Even more dangerous: Distributed Denial of Service (DDoS) attacks
Attacks against Web Services

• Oversize Payload:
  – .NET: 4 MB payload $\rightarrow$ 9 MB memory usage
  – Axis2: 3 MB payload $\rightarrow$ 15 MB memory usage
    6 MB payload $\rightarrow$ 51 MB memory usage
  – Oracle BPEL: 8 MB payload $\rightarrow$ 250 MB memory usage

• Coercive Parsing: Axis2
  – Sending a non-ending sequence of opening tags
    (e.g. $<x>$ $<x>$ $<x>$ $<x>$ $<x>$ ... ) with 150 byte/s
  – 100% CPU load without any connection break-up
DoS Attacks

Resources:
- CPU
- Memory
- Bandwidth

Impact Amplification

© N. Luttenberger
DoS Attacks

Resources:
- CPU
- Memory
- Bandwidth

© N. Luttenberger
DoS Attacks

<Envelope>
  <Header>
    </Header>
  <Body>
    <getWeather>
      <trash>???</trash>
      <trash>???</trash>
      <trash>???</trash>
      <trash>???</trash>
      <trash>???</trash>
      <trash>???</trash>
      ...
    </getWeather>
  </Body>
</Envelope>

© N. Luttenberger
DoS Attacks

<Envelope>
  <Header>
  </Header>
  <Body>
    <EncryptedData>
      ...
    </EncryptedData>
    <CipherValue>
      ADAIfhgdkDGDFrt3
      GDsg3fw45tefsfe
      dgSDGhtjhuqJWq
      Ergr4GRgrgR332K
      ...
    </CipherValue>
  </Body>
</Envelope>

Attack Obfuscation
DoS Attacks

• Metric for impact amplification
  – ratio:
    server memory consumption / wireline message size
  – abbreviated:
    m/m
DoS Attacks

• Conventional attack: **TCP SYN flooding**
  
  – wireline message size: 60 bytes
  
  – server memory consumption (half-open TCP connection): 280 bytes

\[ \frac{m}{m} = 4.7 \]
DoS Attacks

• Experimental **oversize payload** attack against Apache Axis:
  – wireline message size: 1.8 Mbytes
  – server memory consumption (msg buffer, parser): 50 Mbytes

\[
m/m = 28
\]
DoS Attacks

• Experimental attack obfuscation against Apache Axis2 + Rampart:
  – wireline message size: 1 Mbytes
  – server memory consumption (msg buffer, decryption buffer, parser): > 90 Mbytes

m/m = 90
DoS Attacks

- BPEL

Client

Server

State Deviation

© N. Luttenberger
DoS Attacks

• Experimental state deviation attack against Oracle BPEL engine:
  – wireline message size: 0.5 Mbytes
  – server memory consumption (msg buffer, instance lookup, parser): > 350 Mbytes

\[ \frac{m}{m} = 700 \]
DoS Attacks

- TCP Syn Flooding: 4.7
- Oversize Payload: 28
- Attack Obfuscation: 90
- State Deviation: 700

© N. Luttenberger
Attacks against Web Services

- Oversize payload
- Coercive parsing
- SOAPAction spoofing
- XML injection
- WSDL scanning
- Metadata spoofing
- Attack obfuscation
- Oversized cryptography
- Signature wrapping

- BPEL state deviation
- Instantiation flooding
- against all Web Services
- WS-Addressing Spoofing
- against WS-Security-enhanced Web Services
- against BPEL-based Web Services
Chapter 18.2

Countermeasures: Concept
The `XML-SOAP-WS-*-BPEL` pack: a blessing and a curse

Complex XML processing "invites" attacks with malicious messages.

Formal WS metadata enable message validation: Make it efficient!
What happens with non-schema-valid messages?

- an example:: .NET

Correct Message:

```xml
<Envelope>
  <Body>
    <twoParams>
      <a>1</a>
      <b>2</b>
    </twoParams>
  </Body>
</Envelope>
```

Message accepted
Result: a = 1, b = 2

Incorrect Message:

```xml
<Envelope>
  <Body>
    <twoParams>
      <a>
        <b>1</b>
      </a>
      <b>2</b>
    </twoParams>
  </Body>
</Envelope>
```

Message accepted
Result: a = 1, b = 0
WS Metadata

• Message syntax:
  message schemas ← WS description (WSDL)

• Message security:
  message encryption and signature
  ← endpoint/message policies (WS-SecurityPolicy)

• WS compositions:
  message sequence ← BPEL script
WS Security Proxy

- Protection of WS Server by message validation

check for:

Make it efficient!

SOAP Msg. Schema

Security Policy

Protocol Automaton

Web Service Server

conventional network layer firewall

"one for all"
WS Security Proxy

Web Service Client

WS Security Proxy

Validator

Protocol Automaton

SOAP Request

SOAP Response

SOAP Request

SOAP Response

BPEL Script

Compiler

© N. Luttenberger
Digression: XML Tree Processing

- **Envelopes**
  - `<Body>`
    - `<add>`
      - `<x>`12</x>`
      - `<x>`38</x>`
      - `<x>`27</x>`
    - `</add>`
  - `</Body>`

- **Advantages**
  - Convenient tree navigation
  - Straightforward implementation of modification operations

- **Disadvantages**
  - Consumes 10 to 100 times more memory than XML text
  - Tree building must be completed before application processing starts
Digression: Stream-based XML Processing

• Advantages
  – low memory consumption
  – XML processing starts with first parsing event
  – invalid documents need not to be read completely

• Disadvantages
  – no internal document representation for navigation and processing
Digression: Resource Model for XML Processing

- Number of elements: \( n \)
- Number of processing components: \( k \)
- Position of invalid element in document: \( i \)
- Position of detecting component: \( j \)

\[
\begin{align*}
\text{space}^\text{tree} &= n \\
\text{time}^\text{tree} &= n \cdot (j - 1) + i \\
\text{space}^\text{event} &= i \\
\text{time}^\text{event} &= k \cdot (i - 1) + j
\end{align*}
\]
Chapter 18.3

Message Schema Validation
WS Message Schema Validation

• XML Schema validity
  – offline: extract message schema from WS description (⇒ WSDL)
  – online: use message schema for validation
WS Message Schema Validation

- Two building blocks
  1. offline: WSDL compiler
  2. online ("on-the-fly"): Validation Engine

```
+-------------------+            +------------------+
| WSDL              | ->         | WSDL Compiler    |
+-------------------+            +------------------+
                  |            | Validation Engine|

+-------------------+            +------------------+
| WS msg.  |            | Validation Engine| valid WS msg.     |
+-------------------+            +------------------+
                  |            +------------------+
                  |            | Server/Client     |
                  |            | invalid messages  |
```
WS Message Schema Validation

• Validation range

<table>
<thead>
<tr>
<th>SOAP body/[fault]</th>
<th>SOAP header</th>
<th>http header</th>
<th>TCP/IP etc.</th>
</tr>
</thead>
</table>

WS message
to be validated by validation engine

• Standards Conformity
  – WS Interoperability Organization: Basic Profile Version 1.0
  – WSDL 1.1
  – SOAP 1.1
  – W3C XML Schema 2001
WS Message Schema Validation

• Hardening the message schema

```xml
<schema elementFormDefault="qualified" targetNamespace="http://example.com/AddService">
  <element name="add">
    <complexType>
      <sequence>
        <element minOccurs="1" maxOccurs="unbounded" name="x" type="int" />
      </sequence>
    </complexType>
  </element>
</schema>
```

- Restricting the number of elements in sequences
- Modifying the maxOccurs="unbounded" attribute:

```xml
<element minOccurs="1" maxOccurs="100" name="x" type="int" />
```
WS Message Schema Validation

• Hardening the message schema
  – Example: Amount element of FinTS single remittance request

```
<schema>
  <element name="Amount">
    <complexType>
      <sequence>
        <element name="Value" type="decimal"/>
        <element name="Currency" type="string">
          <restriction base="string">
            <enumeration value="EUR"/>
            <enumeration value="USD"/>
          </restriction>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

Adding Schema Facets to further restrict the set of acceptable messages.
WS Message Schema Validation

- http processing: xhttp format

<table>
<thead>
<tr>
<th>http message</th>
<th>xhttp message</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST /ws/Servlet HTTP/1.1\r\n</td>
<td><a href="">xhttp:request</a></td>
</tr>
<tr>
<td>Host: <a href="http://www.ws.de%5Cr%5Cn">www.ws.de\r\n</a></td>
<td><a href="">xhttp:method</a>POST&lt;/xhttp:method&gt;</td>
</tr>
<tr>
<td>Content-Type: text/xml\r\n</td>
<td><a href="">xhttp:uri</a>/ws/Servlet&lt;/xhttp:uri&gt;</td>
</tr>
<tr>
<td>Content-Length: 46\r\n</td>
<td><a href="">xhttp:http-version</a>HTTP/1.1&lt;/xhttp:http-version&gt;</td>
</tr>
<tr>
<td>SoapAction: &quot;urn:get&quot;\r\n</td>
<td><a href="">xhttp:content-type</a>text/xml&lt;/xhttp:content-type&gt;</td>
</tr>
<tr>
<td></td>
<td><a href="">xhttp:content-length</a>47&lt;/xhttp:content-length&gt;</td>
</tr>
<tr>
<td></td>
<td><a href="">xhttp:soapaction</a>urn:get&lt;/xhttp:soapaction&gt;</td>
</tr>
<tr>
<td></td>
<td><a href="">xhttp:body</a></td>
</tr>
<tr>
<td></td>
<td>&lt;tns:echo xmlns:tns=&quot;&quot;&gt;Hello, world&lt;/tns:echo&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xhttp:body&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/xhttp:request&gt;</td>
</tr>
</tbody>
</table>
WS Message Schema Validation

- http processing
**Valid request processing**

- **Client**
  - send request

- **Firewall**
  - receive request
  - validate request
  - send request
  - receive response
  - validate response
  - send response

- **Server**
  - receive request
  - perform operation
  - send response
  - receive response

© N. Luttenberger
WS Message Schema Validation

- Valid request, fault response

Client

send request

Firewall

receive request
validate request
send request

Server

receive request
perform operation
send fault
receive fault
validate fault
send fault
receive fault

© N. Luttenberger
WS Message Schema Validation

• Invalid request

Client

send request

Firewall

receive request

validate request

send "reject" (opt.)

Server

receive reject

© N. Luttenberger
WS Message Schema Validation

- Valid request, invalid response/fault

Client
- send request

Firewall
- receive request
- validate request
- send request
- receive request
- perform operation
- send response/fault
- receive resp./fault
- validate resp./fault
- generate (new) fault
- send resp./fault

Server
- receive response/fault
- to be configured

© N. Luttenberger
WS Message Schema Validation

- Environment

![Diagram](image-url)

- Linux
- Tomcat Application Server
- WSDL compiler
- HTML GUI
- WS messages

© N. Luttenberger
WS Message Schema Validation

- Integration into widely accepted perimeter configuration
Chapter 18.4

Security Policy Validation
Online: WS Security Proxy – Processing Chain

SOAP Message ➔ SAX Parser ➔ Security Component

Security Component:
- WS-Security Handler
- Encryption Handler
- Signature Handler
- Policy Handler

Security Policy Validator

From WSDL:
- Msg. gram.
- Protocol automaton
- Message Validator
- Message Sequence Checking

SAX events ➔ eSAX events ➔ Policy events

© N. Luttenberger
Offline: Endpoint, Message Policy Compilation

Endpoint Policy

<Message Policy>
  <wsp:Policy>
    <sp:EncryptedElements>
      <sp:XPath>/Envelope/Body/payOrder</sp:XPath>
    </sp:EncryptedElements>
    <sp:SignedParts>
      <sp:Body/>
    </sp:SignedParts>
  </wsp:Policy>
</Message Policy>

General Assertions:
- Timestamp
- Protection order

Protection Assertions:
- Sign: SOAP body
  - using X.509 certificate
  - using RSAv1.5
- Encrypt: element “payorder” in body
  - using X.509 certificate
  - Using 3DES
"The normal form represents a policy as a collection of policy alternatives, and a policy alternative as a collection of policy assertions ..."
Create sets $\text{Sig}^p$ and $\text{Enc}^p$ from security policy:

- $\text{Sig}^p \subseteq T^p \times R^p \times A^p$ the required signatures,
- $\text{Enc}^p \subseteq T^p \times R'^p \times A^p$ the required block encryptions, with:

$T^p$ the set of token type requirements, $R^p$ and $R'^p$ the set of references to signed/encrypted elements, and $A^p$ the set of algorithm suites.
Online: WS-Security Processing

• Checking timestamps
• Validating certificates
• Signature verification
  – Verifying signature element
  – Hashing of signed messages part
• Decrypting encrypted message parts
Online: XML Signature Verification

- **Signed blocks:** not explicitly marked!

- **Forward references:**
  - Verify signature
  - Store reference + hash value
  - Detect signed block (via Id attribute)
  - Canonicalize + hash signed block
Online: XML Signature Verification

- Signed blocks: not explicitly marked!
- Backward references:
  - Elements with Id attribute: possibly signed
  - Canonicalize + hash possibly signed block
  - Verify signature
  - Small memory and processing overhead incurred
Order of Signature and Encryption

<env:Header>
  <ds:Signature>
    <ds:Reference URI="#sig-1"/>
  </ds:Signature>
  ...  
  <ns1:Operation>
    <xenc:ReferenceList>
      <xenc:DataReference URI="#enc-1"/>
    </xenc:ReferenceList>
  </ns1:Operation>
  ...  
  <ds:Signature>
    <ds:Reference URI="#sig-2"/>
  </ds:Signature>
</env:Header>

<env:Body Id="sig-1">
  <ns1:Operation Id="sig-2">
    <xenc:EncryptedData Id="enc-1">
      tSm0WR0NVy6gLdLkLkDc...
    </xenc:EncryptedData>
  </ns1:Operation>
</env:Body>

(order of handlers)

(start(Body) -> start(EncryptedData) -> character(“tSmo...“) -> end(EncryptedData) -> end(Body))

(start(Body) -> start(EncryptedData) -> character(“tSmo...“) -> start(SecretElem) -> character(“secret“) -> end(SecretElem) -> end(EncryptedData) -> end(Body))

(start(EncryptedData) -> start(SecretElem) -> character(“secret“) -> end(SecretElem) -> end(EncryptedData) -> end(Body))

© N. Luttenberger
Meta-Events for Policy Verification

• Results of WS-Security processing passed to Policy component as additional events:
  – securityToken(st)
  – signature(s)
  – signedElement(e)
  – encryptedElement(e)
  – timestamp(t)
Online: Policy Validation

- Create sets $\text{Sig}^M$ and $\text{Enc}^M$ from SOAP message and match sets $\text{Sig}^M$ and $\text{Enc}^M$ against sets $\text{Sig}^P$ and $\text{Enc}^P$
Example: Internet Election System

```xml
<sequence>
  <receive operation="init_election" /> 
  <receive operation="set_candidates" /> 
  <receive operation="set_number_of_voters" /> 
  <while condition="vote_on()">
    <receive operation="vote" />
  </while>
  <switch>
    <case condition="not(equal(n1,n2))">
      <invoke operation="announce_winner" />
    </case>
    <otherwise>
      <invoke operation="announce_no_winner" />
    </otherwise>
  </switch>
</sequence>
```
Example: Internet Election System

- Sample attack

init_election → set_candidate → set_number_of_voters → vote → vote → vote → announce winner → announce no_winner

© N. Luttenberger
Message Sequence Validation

• State Automaton
  – BPEL script implies stateful communication protocol
  – WS firewall must know about the current state inside the BPEL engine (cf. stateful TCP firewall)
  – State automaton for message sequence validation needed

⇒ Derive state automaton from BPEL script
Message Sequence Validation

- Modeling BPEL scripts: Successor State Automaton (SSA)

Problem:
Firewall cannot always predict internal state of BPEL engine \( \rightarrow \) FSM might become too complex or too restrictive.

Solution:
\( \rightarrow \) accept set of successors

deterministic FSM
Message Sequence Validation

- Firewall uses special automaton:
  Successor Set Automaton (SSA)
  - SSA checks incoming messages
    - if transition from current state to state in successor set exists
      → accept message + advance current state
    - otherwise → reject message
  - SSA is generated from BPEL document:
    - recursively iterate over activities, and
      create automaton fragments for each activity
    - assemble complete automaton
Message Sequence Validation

• Successor Set Automaton

```
<parent>
  <preceeding-sibling />
  <preceeding-sibling />  
  ...  
  <activity>
    <first-child />
    <second-child />
    ...  
    <last-child />
  </activity>
  
</parent>
```
Message Sequence Validation

• SSA Construction
  
  mappings for `<receive>`, `<reply>`, `<invoke>`

```xml
<parent>
  <preceeding-sibling />
  <preceeding-sibling />
  ...
  <receive
      partnerLink="PL"
      portType="PT"
      operation="OP"
      ...
    />
  <following-sibling />
  <following-sibling />
  ...
</parent>
```
Message Sequence Validation

- SSA Construction
  - mapping for `<sequence>`

```xml
<parent>
  <preceeding-sibling />
  <preceeding-sibling />
  ...
  <sequence>
    <first-child />
    <second-child />
    ...
    <last-child />
  </sequence>
  </sequence>
  ...
  <following-sibling />
  <following-sibling />
  ...
</parent>
```
Message Sequence Validation

• SSA Construction
  – mapping for `<switch>`

... `<switch>`
  `<case condition="c1">`
    `<first-child />`
  `</case>`
  ...
  `<otherwise>`
    `<last-child />`
  `</otherwise>`
 `<switch>`
...
Message Sequence Validation

• SSA Construction
  – mapping for <while>

<parent>
  <preceeding-sibling />  
  <preceeding-sibling />  
  ...
  <while condition="c1">
    <single-child>
      ...
    </single-child>
  </while>
  <following-sibling />  
  <following-sibling />  
  ...
</parent>
Example revisited: SSA generation

```xml
<sequence>
  <receive op="init_election" />
  <receive op="set_candidates" />
  <receive op="set_number_of_voters" />
  <while condition="vote_on()">
    <receive op="vote" />
  </while>
  <switch>
    <case condition="not(equal(n1,n2))">
      <invoke op="announce_winner" />
    </case>
    <otherwise>
      <invoke op="announce_no_winner" />
    </otherwise>
  </switch>
</sequence>
```
Example revisited: SSA generation

<sequence>
  <receive op="init_election" />
  <receive op="set_candidates" />
  <receive op="set_number_of_voters" />
  <while condition="vote_on()">
    <receive op="vote" />
  </while>
  <switch>
    <case condition="not(equal(n1,n2))">
      <invoke op="announce_winner" />
    </case>
    <otherwise>
      <invoke op="announce_no_winner"
    </otherwise>
  </switch>
</sequence>
Example revisited: SSA generation

<sequence>
  <receive op="init_election" />
  <receive op="set_candidates" />
  <receive op="set_number_of_voters" />
  <while condition="vote_on()">
    <receive op="vote" />
  </while>
  <switch>
    <case condition="not(equal(n1,n2))">
      <invoke op="announce_winner" />
    </case>
    <otherwise>
      <invoke op="announce_no_winner" />
    </otherwise>
  </switch>
</sequence>
Example revisited: SSA generation

```xml
<sequence>
  <receive op="init_election" />
  <receive op="set_candidates" />
  <receive op="set_number_of_voters" />
  <while condition="vote_on()">
    <receive op="vote" />
  </while>
  <switch>
    <case condition="not(equal(n1,n2))">
      <invoke op="announce_winner" />
    </case>
    <otherwise>
      <invoke op="announce_no_winner" />
    </otherwise>
  </switch>
</sequence>
```
Example revisited: SSA generation

<sequence>
  <receive op="init_election" />
  <receive op="set_candidates" />
  <receive op="set_number_of_voters" />
  <while condition="vote_on()">
    <receive op="vote" />
  </while>
  <switch>
    <case condition="not(equal(n1,n2))">
      <invoke op="announce_winner" />
    </case>
    <otherwise>
      <invoke op="announce_no_winner" />
    </otherwise>
  </switch>
</sequence>
Example revisited: SSA generation

```xml
<sequence>
  <receive op="init_election" />
  <receive op="set_candidates" />
  <receive op="set_number_of_voters" />
  <while condition="vote_on()">
    <receive op="vote" />
  </while>
  <switch>
    <case condition="not(equal(n1,n2))">
      <invoke op="announce_winner" />
    </case>
    <otherwise>
      <invoke op="announce_no_winner" />
    </otherwise>
  </switch>
</sequence>
```
Example revisited: SSA generation

```xml
<sequence>
  <receive op="init_election" />
  <receive op="set_candidates" />
  <receive op="set_number_of_voters" />
  <while condition="vote_on()">
    <receive op="vote" />
  </while>
  <switch>
    <case condition="not(equal(n1,n2))">
      <invoke op="announce_winner" />
    </case>
    <otherwise>
      <invoke op="announce_no_winner" />
    </otherwise>
  </switch>
</sequence>
```
Example revisited: Complete Automaton
Automaton Refinement
Chapter 18.6
Evaluación
Evaluation: Runtime for Unlimited Schema

- Unlimited Schema
- Limited Schema

Invalid Messages

Number of elements vs. Runtime (ms)
Evaluation: Memory Consumption

- Unlimited Schema
- Limited Schema

Memory consumption (KByte) vs. Number of elements graph.
Evaluation Procedure

• "Typically" secured SOAP Message
  – Including one signature
  – Encrypted SOAP-Body

• Two test series:
  – Correct messages
  – Incorrect messages
    • schema violation inside encrypted part
    • no policy violation

• Increase encrypted body size in each test step
Evaluation: Memory Consumption

• In both cases (up to 500,000 elements):
  – constant memory usage: 9 MB

• Apache XML Security Framework (only decryption!):
  – 80,000 elements: 64 MB
  – 85,000 elements: memory overflow
Evaluation: Security Processing

![Graph showing memory consumption vs. number of elements]

CheckWay
Axis2

Memory consumption (KByte)

Number of elements

© N. Luttenberger
Evaluation: Runtime

Runtime [s] vs. Number of XML elements

Correct messages

Incorrect messages

© N. Luttenberger
Evaluation

• Message sequence validation
  – No significant overhead for valid messages
  – Efficient detection and rejection of "red" messages
    → no resource consumption at BPEL engine
Evaluation

- Performance Test for BPEL state deviation attack: correlation-invalid messages
  - BPEL engine
    test 1:
    3500 offending messages
    memory: 280 → 650 MB
    99% CPU load for 5 min.
  
  test 2 (10 min later):
  1100 offending messages
  memory: 280 → 658 MB
  99% CPU load for 145 min.

- WS Firewall
  single test:
  8000 offending messages
  memory: 20 → 23 MB
  50% CPU load during attack
Results

• Protection effect:
  – protection from non-distributed DoS attacks
  – protection from several fraud attacks

• Memory consumption + runtime:
  – upper limit depends on WS description and WS Security Policy
  – thus: limits apply for attacker also

• But:
  – Careful (manual) inspection of WS description and WS Security Policy may still be required!
Attacks against Web Services

• Fraud
  – WS-Security has focus on integrity and confidentiality
    ◦ Cannot protect from attacks targeting availability
    ◦ Cannot protect from XML rewriting attacks

• New attacks introduced by WS-Security
  – Policy non-conforming message
    ◦ Missing security tokens
  – Encrypted malicious content
    ◦ No syntactic checks possible
    ◦ Can mask prior mentioned attacks