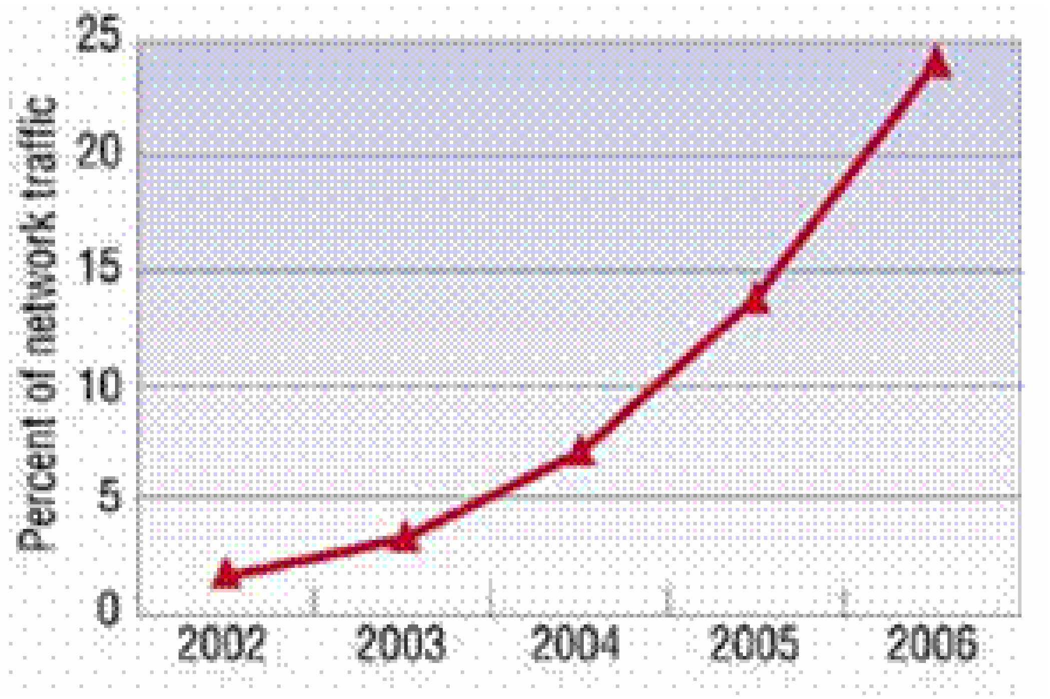
Don’t drop the SOAP!

Dr. Tony Palmer
Vordel

OWASP
Ireland
Wed 22nd June 2005

Why are we here?

"By 2005, XML based communications using SOAP will have reopened 70 percent of the attack paths against Internet-connected systems, which were closed by network firewalls in the 1990s" - Gartner, October 2002



"A recent research report from ZapThink LLC estimates that XML traffic will triple on enterprise networks by 2008, accounting for close to 50% of the messages sent "

– April 2005

Why such a growth?

Simple XML/SOAP Message

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tns="http://www.4d.com/namespace/default"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <tns:LOTTO_NumCheck_SingleSet_Hist>
      <tns:inN1>3</tns:inN1>
      <tns:inN2>4</tns:inN2>
      <tns:inN3>12</tns:inN3>
      <tns:inN4>34</tns:inN4>
      <tns:inN5>21</tns:inN5>
      <tns:inN6>16</tns:inN6>
      <tns:inHistoric>true</tns:inHistoric>
    </tns:LOTTO_NumCheck_SingleSet_Hist>
  </soap:Body>
</soap:Envelope>
```

(courtesy http://www.xmethods.com and Mindreef)

Tools/Standards required to implement Web Services today

Transport

- HTTP(s)
- FTP
- SMTP
- MQ
- Something else

Processing

- XML Parser
- REST
- Text Manipulation

If this is all we need...

Standards/organisations/specifications

HTTP(S) UDDI CRL XKMS

SAML 1.0 XACML

W3C SSL/TLS Soap 1.2

WS-SecureConversation WS-Privacy

REST

WS-Policy LDAP XML Schema

Soap 1.1 SAML 1.1 X509 PKCS*

WS-* WS-SecurityPolicy

OASIS

SAML 2.0 XML Signature

DTD WSDL

WS-Security OCSP WSI WS-Federation

Liberty Alliance

Back Up! WS-Trust MIME/DIME/MTOM

XML security and management

1. Transport
2. Authentication
3. Authorization
4. Confidentiality
5. Non Repudiation
6. Content Filtering
7. Federated Identity
8. Interoperation+ Integration
9. Audit
10. Availability

If everybody did things their own way it could never work!

Look at transport…

Transport

Making transport less important.

- HTTPBasic Username/Passwords very common.
- SSL/TLS standard transport layer security
- But messages may travel over several protocols!

Take security contexts from transport specific mechanisms and embed them into the message.

Now with this in mind..

Authentication + Authorization

Various ways to authenticate

- Username/Password
- Certificate
- Other e.g. Kerberos, Biometrics…

Authorization

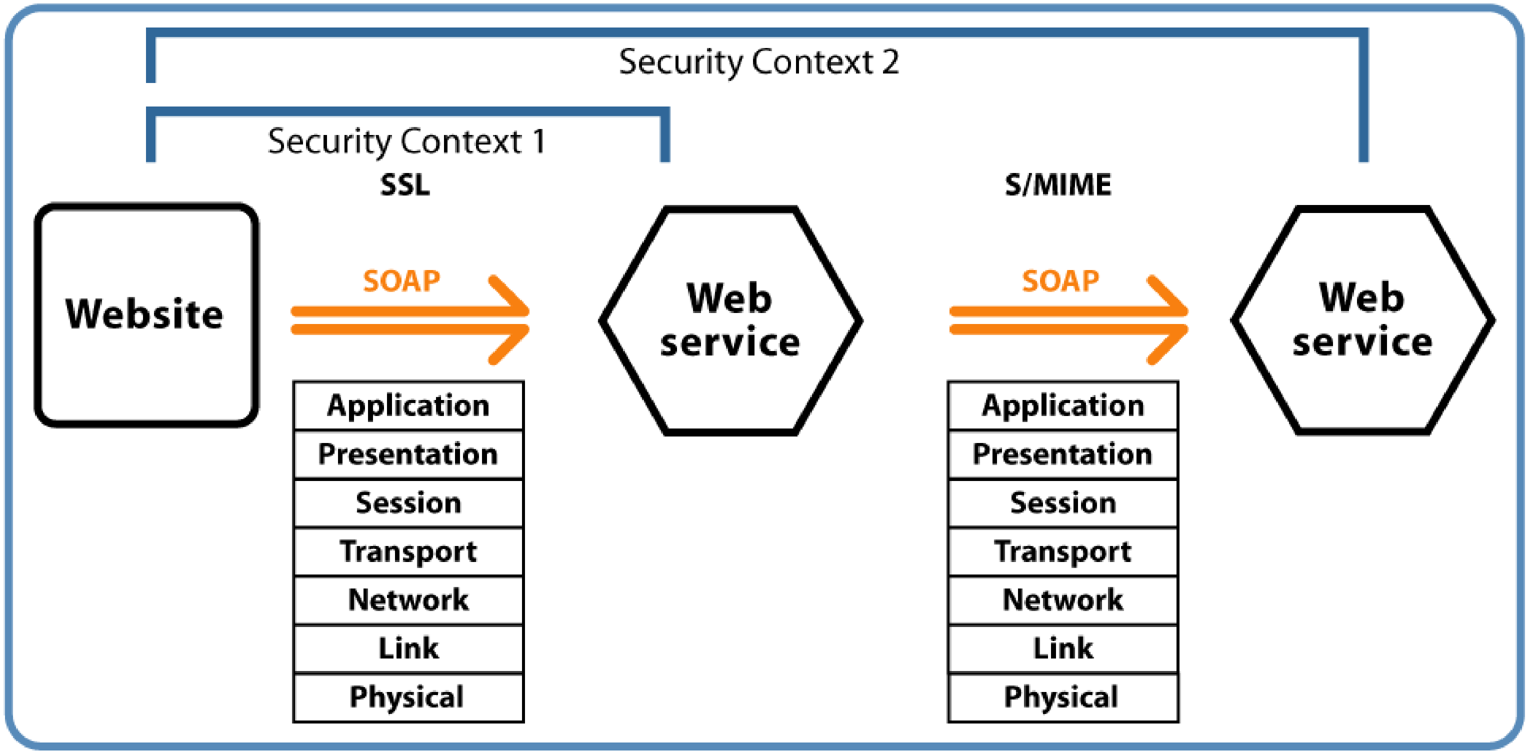
- Dynamic checks
- 3rd Party Token

Need standard mechanisms for inserting information into the message.

Confidential…?

Confidentiality

SOAP messages may pass over multiple "hops". If only transport-level security is used, then the security context is lost after each hop. This is another reason why security information must be in the message itself.



Proof…

Non Repudiation

Typically the exchange of XML between organizations reflects business processes.

Exposing potentially high value Web Services to partners could lead to scenarios where users of a web services can deny that transactions have occurred.

e.g.

We did not order 5000 widgets last month. We ordered *3000,*

OR

***That* message did not originate with us.**

The need to have a standard mechanism to prove message authenticity and making them tamper proof becomes apparent.

Bad messages…?

Content Filtering

Firewalls cannot distinguish between good and bad XML-over-HTTP – so it's "all or nothing" – block all XML, or let it all through.

Remember SSL may be your enemy!!

The following two messages look the same to most firewalls:

```
<SOAP-ENV:Envelope
 SOAP-ENV:encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENV = "http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header></SOAP-ENV:Header>
<SOAP-ENV:Body>
<SOAPAPP:GetTime xmlns:SOAPAPP = "http://tempuri.org/message/">
<TimeZone>GMT</TimeZone>
</SOAPAPP:GetTime>
</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

```
<SOAP-ENV:Envelope
 SOAP-ENV:encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENV = "http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header></SOAP-ENV:Header>
<SOAP-ENV:Body>
<SOAPAPP: PleaseDontRunMe xmlns:SOAPAPP="http://tempuri.org/message/">
Hack Hack Hack Hack
</SOAPAPP: PleaseDontRunMe >
</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

Federated Identity

Tony logs on at one location and authenticates.

Tony needs to access systems within other domains or organizations.

Tony wants to avoid having to re-authenticate at each location.

Each location may need to have an ACL for Tony.

Tony may have a different identity (e.g. Mark) at certain locations.

-OR-

A Web Service consists of an orchestrated collection of discrete atomic Web Services, each on different systems requiring their own Authn/AuthZ

With more systems being leveraged to become part of an SOA the need to cut down on complexity and obtain Single Sign On (SSO) becomes increasingly important.

All together…

Interoperation and Integration

Will your Web Services deployment work with your partners?

Can you exchange security tokens?

Is my investment future proof?

Can I leverage my current infrastructure?

Do we speak the same (security) language?

What about newer applications?

What about new standards?

So far

Up to now we have not discussed a single XML Security specific acronym or specification!

Focus on what you want to achieve rather than on the tools that you can use.

Don't think I need signatures…

Think I need to prove that my messages have not been tampered with.

Now let's look at some specifications and standards.

XML Signature

XML Signature is a joint IETF (Internet Engineering Task Force) and W3C (World-Wide Web Consortium) initiative.

Used for signing "Any digital content" – not just XML.

- Defines an XML syntax for a signature
- XML Signature is not just for signing XML

XML Signature may be:

- Enveloped (XML signature located in source XML)
- Enveloping (XML signature wraps around the source XML)
- External (XML signature in a separate document to the source)

http://www.w3.org/Signature/

Example…

XML Signature Example

```
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
 <soap-env:Header>
  <dsig:Signature id="Tony Palmer" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
   <dsig:SignedInfo>
    <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n"/>
    <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <dsig:Reference URI="">
     <dsig:Transforms>
      <dsig:Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
       <dsig:XPath xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">ancestor::soap-env:Body</dsig:XPath>
      </dsig:Transform>
      <dsig:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n"/>
     </dsig:Transforms>
     <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
     <dsig:DigestValue>fN4KWANjdhzJCMuJAswPRWVh+T8=</dsig:DigestValue>
    </dsig:Reference>
   </dsig:SignedInfo>
   <dsig:SignatureValue/>
   <dsig:KeyInfo>
    <dsig:X509Data>
     <dsig:X509SubjectName>CN=Tony Palmer,O=Vordel,C=ie</dsig:X509SubjectName>
   </dsig:X509Data>
   </dsig:KeyInfo>
  </dsig:Signature>
 </soap-env:Header>
 <soap-env:Body>
  <product version="3.1">
   <name>VordelSecure</name>
   <company>Vordel</company>
   <description>WebServices Security</description>
  </product>
 </soap-env:Body>
</soap-env:Envelope>
```

XML Encryption

A W3C Recommendation

Includes XML syntax for:

- Representing encrypted XML content
- Containing information needed to decrypt encrypted content

Allows per-element encryption.

- Allows you to encrypt any elements/nodes in a document
- The recipient may be able to decrypt only some of them
- Multiple recipients / partitioned data

Tells the recipient how to decrypt the message.

- Algorithm needed to decrypt, etc.
- Not the private key (obviously)

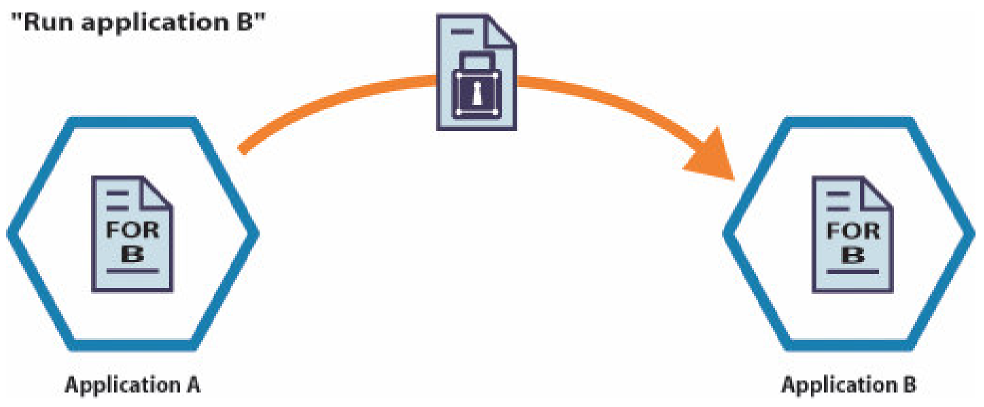
http://www.w3.org/Encryption

Architectural view…

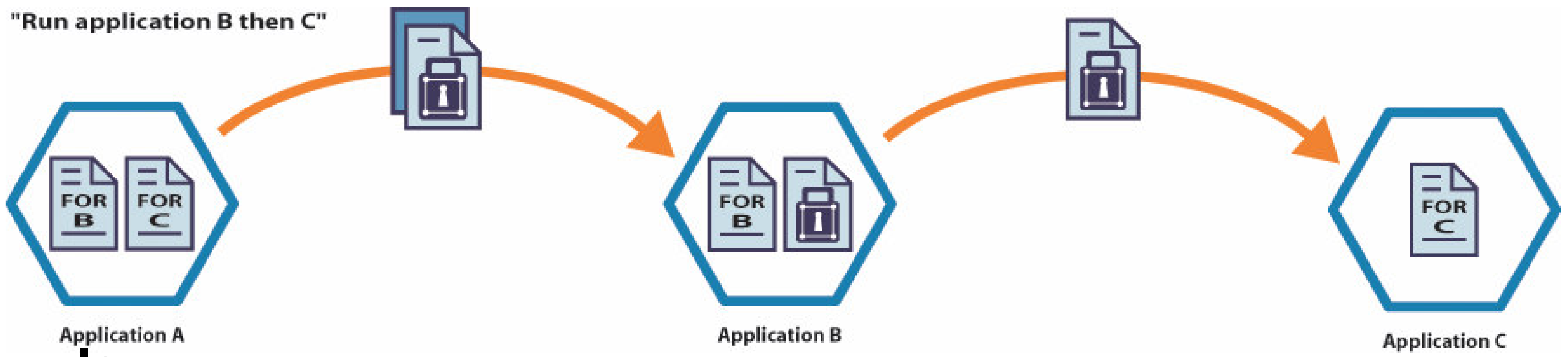
Encryption for SOAP

For full end-to-end security.

Transport level security is just point-to-point security.



For full end-to-end security, *persistent security* is required.



Example…

XML Encryption Example

Credit Card data to be encrypted

```
<?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>Bank of the Internet</Issuer>
            <Expiration>04/02</Expiration>
    </CreditCard>
</PaymentInfo>
```

XML Encryption

Scenario: Encrypt only the card's number, but indicate that the number exists

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith<Name/>
  <CreditCard Limit='5,000' Currency='USD'>
   <Number>
           <EncryptedData
  xmlns='http://www.w3.org/2001/04/xmlenc#'
           Type='http://www.w3.org/2001/04/xmlenc#Content'>
           <CipherData>
                   <CipherValue>A23B45C56</CipherValue>
           </CipherData>
           </EncryptedData>
   </Number>
   <Issuer>Bank of the Internet</Issuer>
   <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

SAML overview

OASIS Recommendation

SAML - Security Assertion Markup Language

- A specification for using XML to describe:
  - An authentication act (e.g. "Joe User authenticated using a username and password at 9am today")
  - Authorization information (e.g. "Joe User is allowed to access the GetStockQuote Web Service")
  - Profile information (e.g. "Joe User has a credit rating of 4.5")

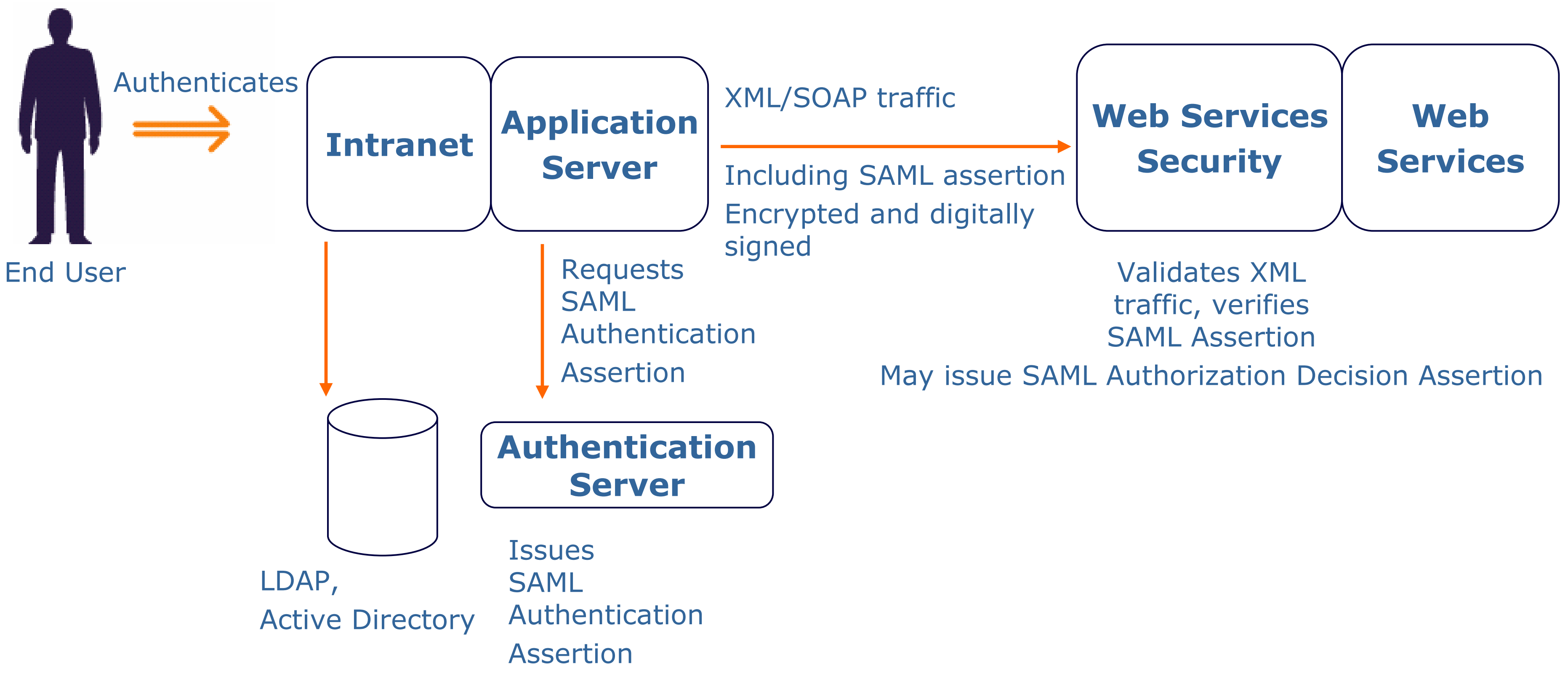
- Allows an end-user to use Web Services of another company *as if they had a user profile there*
- Must trust the Issuer of the Assertion
- Local downstream consumption
- Saml 1.0 Nov 2002, **1.1 Sept 2003**, 2.0 Mar 2005 (Final Spec)

http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

Scenario…

Single Sign-On using SAML

Scenario: End-user authenticates once (Single-Sign On) and uses multiple local and third-party Web Services, without a need to re-authenticate.


Authenticates
End User
Intranet
Application Server
XML/SOAP traffic
Including SAML assertion
Encrypted and digitally signed
Web Services Security
Web Services
Validates XML traffic, verifies SAML Assertion
May issue SAML Authorization Decision Assertion
Requests SAML Authentication Assertion
Authentication Server
Issues SAML Authentication Assertion
LDAP,
Active Directory

So how does it fit together?

So we have the standard building blocks for

- Signing XML data
- Encrypting XML
- - Exchanging security credentials

So how does this all fit together in a standard *interoperable* manner?

- Enter WS-Security -

Timeline of WS-Security

Initially released by Microsoft in October 2001.

April 2002, IBM and Microsoft released "Security in a Web Services World" document.

- Defined a security framework for Web Services, the first of which to be released (in conjunction with VeriSign) was WS-Security.

Later specifications for Web Services security included WS-Trust, WS Policy, and WS-SecureConversation.

June 2002 - WS-Security submitted to the OASIS.

- A Web Services Security group was formed in OASIS to develop WS-Security as an OASIS standard.

Became an OASIS Recommendation in April 2004

Applies only to SOAP

So what is WS-Security?

WS-Security defines element names for packaging security tokens into SOAP messages

Abstracts different security technologies into “claims” and “tokens.”

The additional Web Services Security road map (“WS-*”) specifications build on these concepts: -

- Explaining how to apply for a security token
- How tokens are linked to identity
- How security information may be associated with a Web Service

Nuts and Bolts of WS-Security

One simple, and important, thing which WS-Security does is to define a “Security” element to go into the SOAP Header, separate from the data.

```
<soap:Envelope>
  <soap:Header>
    <Security soap:actor=“gateway" soap:mustUnderstand="1">
       . . .
    </Security>
  </soap:Header>

  <soap:Body>
     <Data/>
  </soap:Body>

</soap:Envelope>
```

Examples…

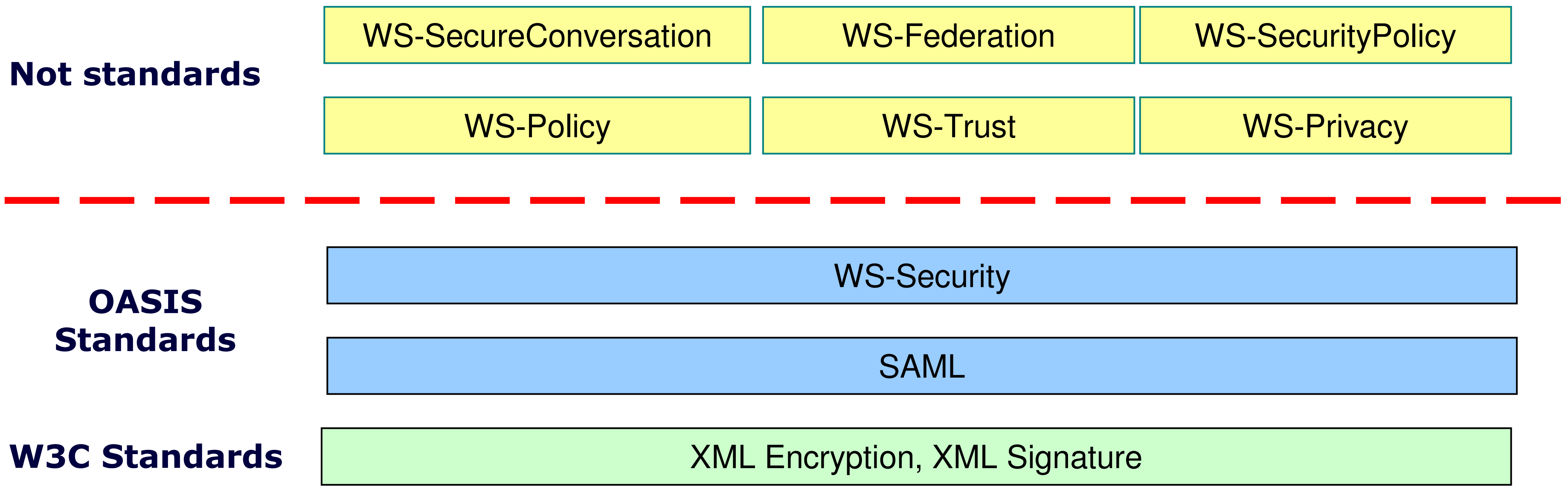
Multiple credentials in the same message

```
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
  <soap-env:Header>
  <wsse:Security soap-env:actor="application1" xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext">
    <wsse:UsernameToken wsu:Id="Tony" xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
      <wsse:Username>Tony</wsse:Username>
      <wsse:Password Type="wsse:PasswordText">password!</wsse:Password>
      <wsu:Created>2005-05-26T21:20:48Z</wsu:Created>
    </wsse:UsernameToken>
  </wsse:Security>
  <wsse:Security soap-env:actor="application2" xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext">
    <wsse:UsernameToken wsu:Id="Tony Palmer" xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
      <wsse:Username>Tony Palmer</wsse:Username>
      <wsse:Nonce EncodingType="UTF-8">4cbq7VUPaUBo4YpbnL12OuTZRXb4Df1ZZW/PGRJbk24=</wsse:Nonce>
      <wsse:Password Type="wsse:PasswordDigest">EWpuFmLYS3A/RbWZBEcLSgCgnmw=</wsse:Password>
      <wsu:Created>2005-05-26T21:24:19Z</wsu:Created>
    </wsse:UsernameToken>
  </wsse:Security>
  </soap-env:Header>
  <soap-env:Body>
    <description>XML Security Company</description>
</soap-env:Body>
</soap-env:Envelope>
```

SAML Attribute Assertion

```
 <wsse:Security soap-env:actor="Policy engine"
xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext">
   <saml:Assertion AssertionID="Id-1370134408067483383-1117143936207" IssueInstant="2005-05-
26T21:45:36Z" Issuer="CN=Administrator,OU=HR,O=Vordel Ltd.,L=Dublin 4,ST=Dublin,C=IE"
MajorVersion="1" MinorVersion="1" xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
     <saml:Conditions NotBefore="2005-05-26T21:15:50Z" NotOnOrAfter="2008-05-26T23:15:50Z"/>
     <saml:AttributeStatement>
       <saml:Subject>
         <saml:NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:X509SubjectName">Tony Palmer</saml:NameIdentifier>
       </saml:Subject>
       <saml:Attribute AttributeName="dept" AttributeNamespace="http://www.vordel.com">
         <saml:AttributeValue>Engineering</saml:AttributeValue>
       </saml:Attribute>
       <saml:Attribute AttributeName="id" AttributeNamespace="http://www.vordel.com">
         <saml:AttributeValue>3456542</saml:AttributeValue>
       </saml:Attribute>
       <saml:Attribute AttributeName="unix_login" AttributeNamespace="">
         <saml:AttributeValue>tony</saml:AttributeValue>
       </saml:Attribute>
     </saml:AttributeStatement></saml:Assertion>
  </wsse:Security>
```

Standards and specifications



System Design…

Besides the standards, what are the other XML security requirements?

XML threat-awareness

- Ensure that XML messages have valid format and content
- Ensure that SOAP attachments are of the appropriate format and size, scan for virus
  - MIME and DIME
- Block XML denial-of-service attempts

Integration with existing security infrastructure

- Link to silos of identity information e.g. LDAP directories and Active Directory
- Send alerts to enterprise monitoring tools e.g. SNMP

Auditability

- Log all XML activity and provide a reporting interface

Administration

- Allow policies to be updated without a requirement to edit code or edit XML

Availability

- Alerts when XML transactions do not match a Service Level Agreement
- Handle the case when a Web Service becomes unresponsive

Hacks…

Attacks moving up the stack

Application Layer security has existed long before Web Service.

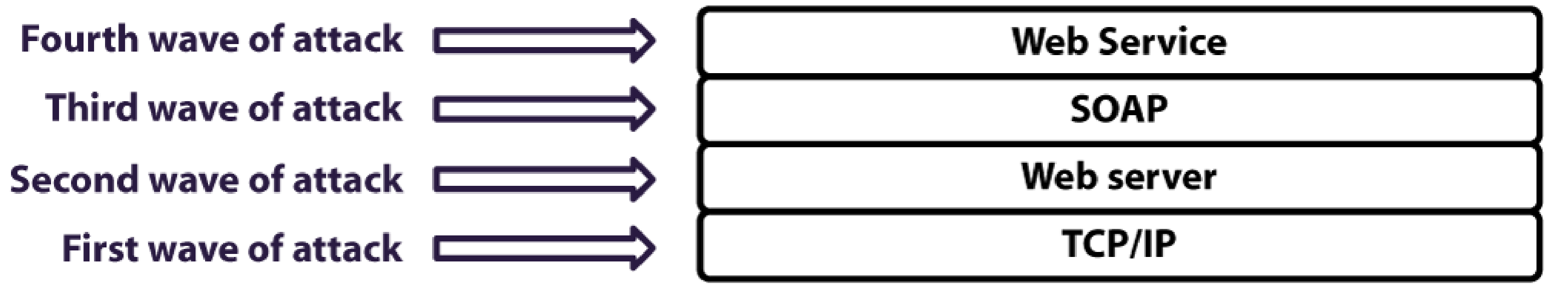
Application Layer security began with securing the Web server itself.

- Patches, security updates

Next came "Web Application Security".

- A Web application is a CGI-based application with which a user interacts using a web browser.
- Attacks include Cross-Site Scripting, Cookie "poisoning", changing URL parameters (e.g. trying to guess a session ID to get access to an online bank account)

SOAP itself can be seen as a "Web Application".



SQL Injection

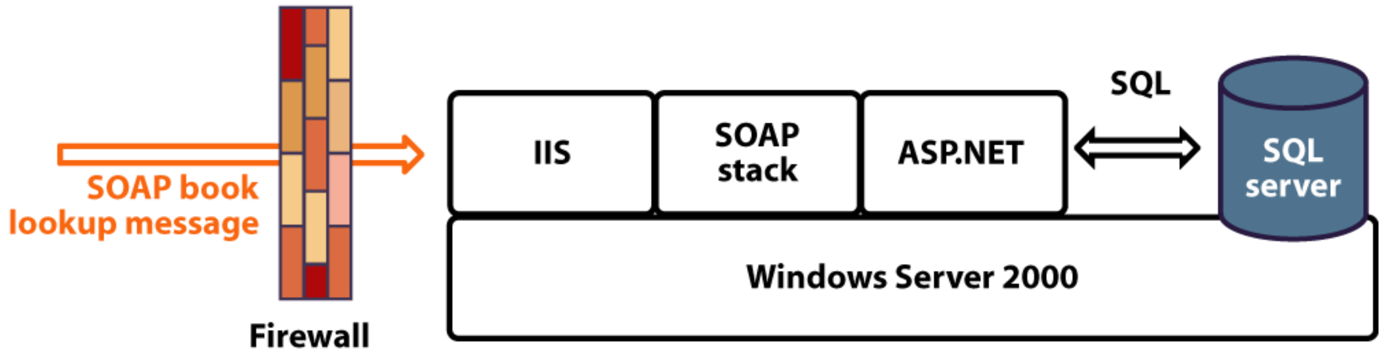
Web Application SQL Injection Attacks

- Inserting SQL statements into web forms in order to force a database to return inappropriate data, or to produce an error which reveals database access information.

Web Services SQL Injection Attacks

- For Web Services, this category of attack translates into manipulating data in a SOAP message to include SQL statements which will be interpreted by a back-end database.

Anatomy of a SQL Injection Attack



```
<SOAP-ENV:Envelope xmlns:SOAP-
   ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header></SOAP-ENV:Header>
<SOAP-ENV:Body
<BookLookup:searchByIBSN xmlns:BookLookup="https://www.books.com/Lookup">
<BookLookup:IBSN>1234567890<BookLookup:IBSN>
</BookLookup:searchByIBSN>
</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

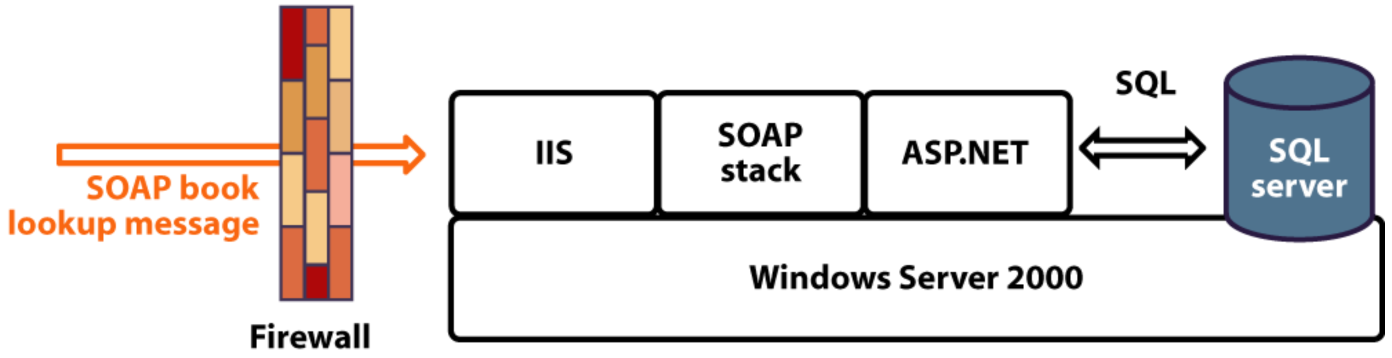
VB.NET code:

```
Set myRecordset = myConnection.execute("SELECT * FROM myBooksTable WHERE IBSN
='" & IBSN_Element_Text & "'")
```

Becomes:

```
SELECT * FROM myBooksTable WHERE IBSN = '1234567890'
```

Anatomy of a SQL Injection Attack



```
<SOAP-ENV:Envelope xmlns:SOAP-
   ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header></SOAP-ENV:Header>
<SOAP-ENV:Body
<BookLookup:searchByIBSN xmlns:BookLookup="https://www.books.com/Lookup">
<BookLookup:IBSN>' exec master..xp_cmdshell 'net user Joe pass
   /ADD'<BookLookup:IBSN>
</BookLookup:searchByIBSN>
</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

VB.NET code:

```
Set myRecordset = myConnection.execute("SELECT * FROM myBooksTable WHERE IBSN
='" & IBSN_Element_Text & "'")
```

Becomes:

```
SELECT * FROM myTable WHERE IBSN =‘’ exec master..xp_cmdshell 'net user Joe pass
/ADD
```

Defending against a SQL injection attack

Ensure the format of incoming SOAP parameters using an XML Schema

```
<simpleType name="ibsn">
  <restriction base="string">
     <pattern value="[0-9]{10}"/>
  </restriction>
</simpleType>
```

Validate this Schema against the data isolated by the following XPath expression:

```
/Body/BookLookup:searchByIBSN/BookLookup:IBSN
```

`1234567890` passes

`' exec master..xp_cmdshell 'net user Joe pass /ADD' --` fails

Analogous XPath Injection

Clog…

XML Denial-of-Service using DTD recursion

Scenario: DTDs are vulnerable to recursion attacks

For example, the following DTD contains a recursively defined entity "&x100;" that would be expanded into the huge amount of 2^100 repetitions of the string "hello" by any XML 1.0 standard compliant parser. This would cause excessive memory usage (and subsequent failure) and/or excessive CPU usage:

```
<?xml version="1.0" encoding="utf-8"?>
    <!DOCTYPE foobar [
        <!ENTITY x0 "hello">
        <!ENTITY x1 "&x0;&x0;">
        <!ENTITY x2 "&x1;&x1;">
        <!ENTITY x3 "&x2;&x2;">
        <!ENTITY x4 "&x3;&x3;">
         ...
        <!ENTITY x98 "&x97;&x97;">
        <!ENTITY x99 "&x98;&x98;">
        <!ENTITY x100 "&x99;&x99;">
    ]>
    <foobar>&x100;</foobar>
```

XML Denial-of-Service using DTD recursion

Platforms requiring patches for this attack were:

ColdFusion MX, Sybase EAServer, IBM WebSphere, Microsoft .NET.

Impact:

Web Services platforms could be disabled by sending them a *single SOAP message.*

Solution:

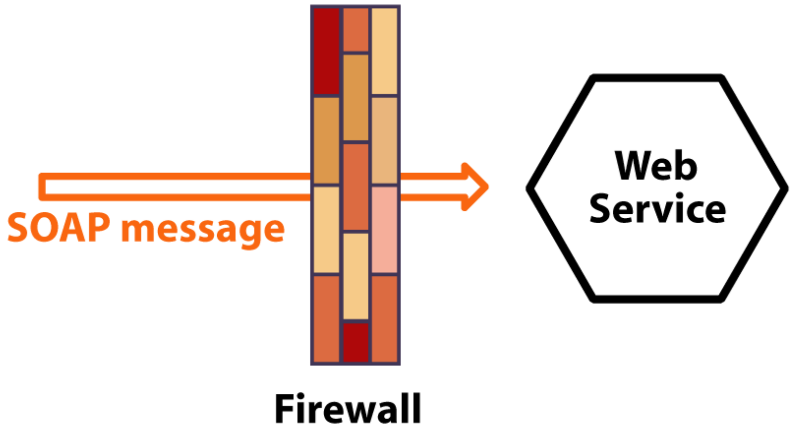
The SOAP specification states “A SOAP message MUST NOT contain a Document Type Declaration" (http://www.w3.org/TR/SOAP/ Section 3).

However, some SOAP-enabled products were vulnerable because they parsed DTDs. The solution is to not support DTDs in SOAP.

Bad Security…

XML Signature processing attack

Scenario: XML Signature library is used to process signed XML message



XML is received, signature is validated over SOAP body.

Validated message is passed into business logic.

XML Signature processing attack

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-
envelope"xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
 <env:Header>
<wsse:Security wsse:actor="http://www.w3.org/2002/06/soap-envelope/role/next">
<wsse:BinarySecurityToken wsse:Id="BestsellersToken"wsse:ValueType="wsse:X5090v3"
wsse:EncodingType="wsse:Base64Binary">asDVIWMI389MJmdn . . .</BinarySecuritytoken>
<dsig:Signature>
<dsig:SignedInfo>
<dsig:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
<dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<dsig:Reference
URI="http://ardownload.adobe.com/pub/adobe/acrobatreader/win/5.x/5.1/AcroReader51_ENU_fu
ll.exe">
</dsig:Reference>
</disg:SignedInfo>
<dsig:SignatureValue>wertwertwert</dsig:SignatureValue>
<dsig:KeyInfo><wsse:SecurityTokenReference>
<wsse:Reference URI="#BestSellersToken"/></wsse:SecurityTokenReference>
</dsig:KeyInfo></dsig:Signature>
</wsse:Security></env:Header>
<env:Body>
   <p:invoice ID="bookinvoice" xmlns:p=http://bestsellers.com/invoice>
      <p:item>NewBook</p:item>
      <p:shipto>Jane Doe</p:shipto>
   </p:invoice>
</env:Body>
</env:Envelope>
```

Replay Attacks

Scenario:

A Web Service is protected by an XML Gateway that scans incoming requests for X.509 certificates contained within SOAP messages; and ensures messages are encrypted and signed. This system is vulnerable to a replay attack which simply replays a valid message, gaining unauthorized access.

Impact: Unauthorized access

Solution:

The usage of timestamps to block replay attacks. WS-Security includes support for timestamps. A replayed message will include the same timestamp as the original message. This means that *both* messages must be discarded, because it cannot be established which message was the original, and which is the copy.

Beware of any solution which claims "this is secure because all incoming messages are signed".

Caution:

Don't confuse replay attacks with "flooding" denial-of-service attacks.

XML Parser Overload

- Verbose by design
- Are eXtensible
- Really any type of data including multimedia or binary data
- Parsers based on the DOM model are especially susceptible

Solution

- Impose data limits *before* parsing
- Block soap attachments

REST – an alternative to SOAP

Stands for REpresentational State Transfer

Uses HTTP, URIs and “GET” strings, rather than SOAP, to request XML

- E.g. services.mycompany.com/GetQuote?Ticker=MSFT

Just as in the case of SOAP, XML is returned by the Web Service.

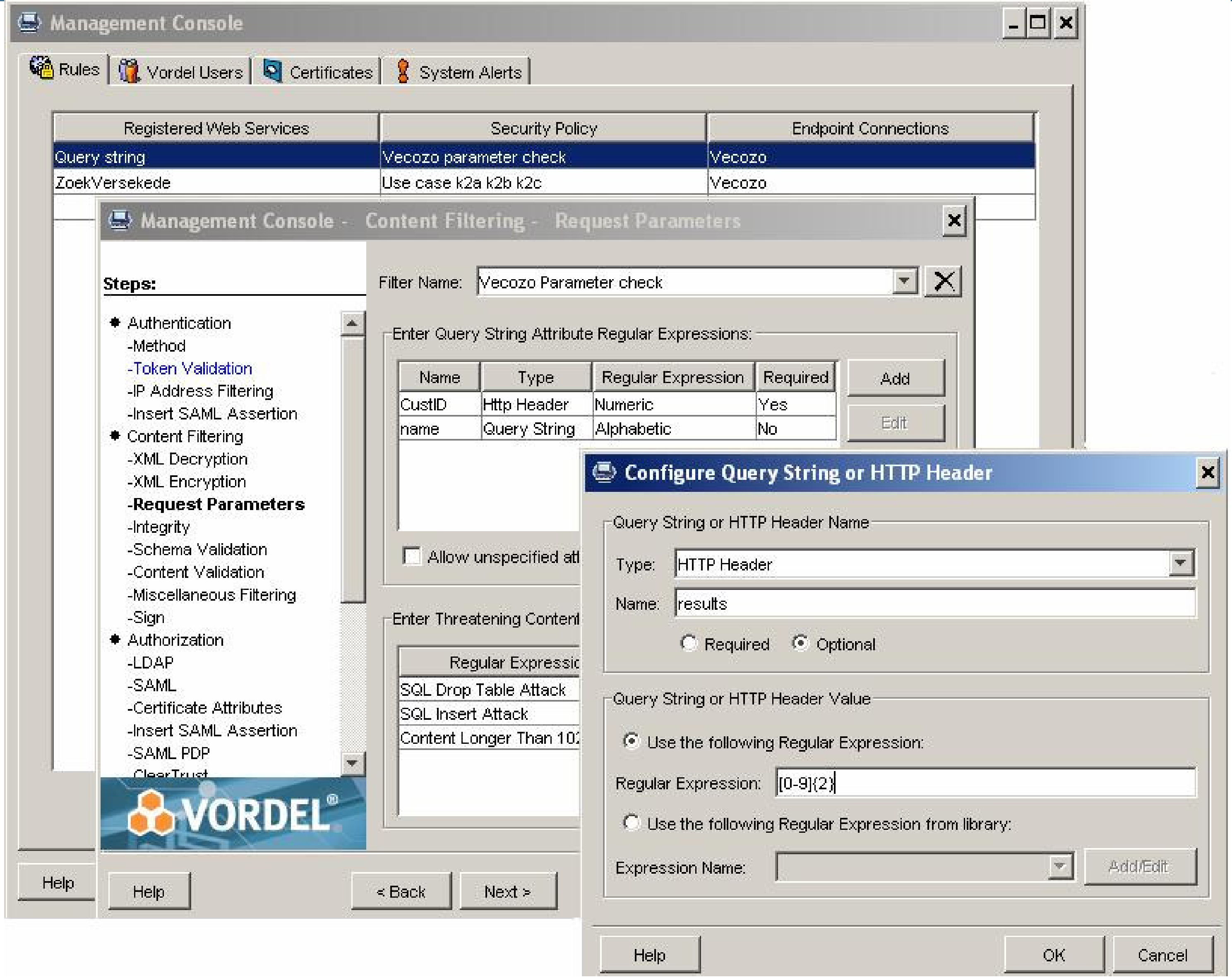
REST has been used for years, it just never had a name

REST is built into SOAP 1.2

Puts the “Web” back into Web Services

Security model not as finely developed as SOAP

Securing Request parameters



Registered Web Services	Security Policy	Endpoint Connections
Query string	Vecozo parameter check	Vecozo
ZoekVersekede	Use case k2a k2b k2c	Vecozo

Filter Name: Vecozo Parameter check

Enter Query String Attribute Regular Expressions:

Name	Type	Regular Expression	Required
CustID	Http Header	Numeric	Yes
name	Query String	Alphabetic	No

☐ Allow unspecified att

Enter Threatening Content

Regular Expressio
SQL Drop Table Attack
SQL Insert Attack
Content Longer Than 10

Query String or HTTP Header Name

Type: HTTP Header

Name: results

Required [✓ Optional]

Query String or HTTP Header Value

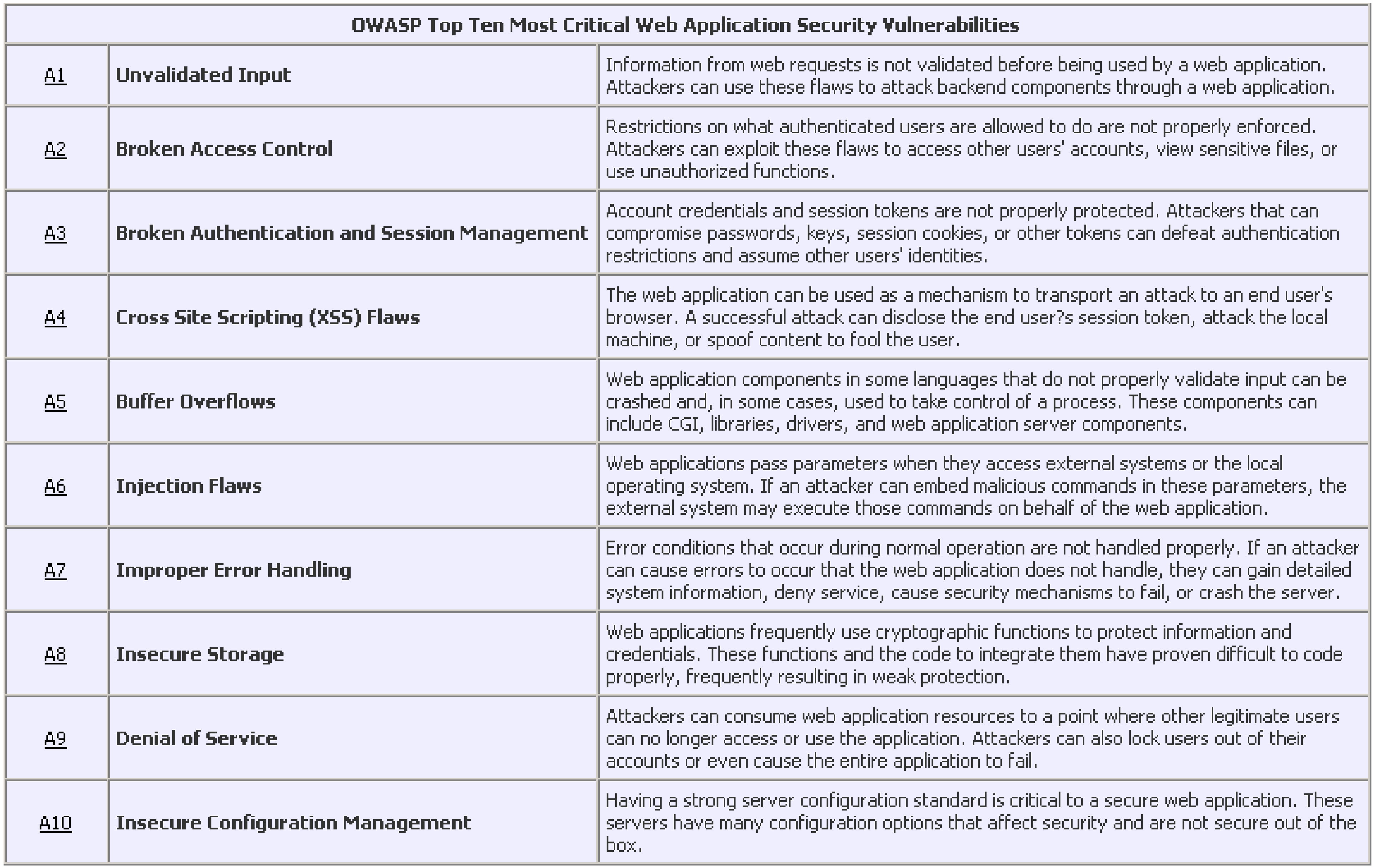
[✓ Use the following Regular Expression:]

Regular Expression: [0-9]{2}

Use the following Regular Expression from library:

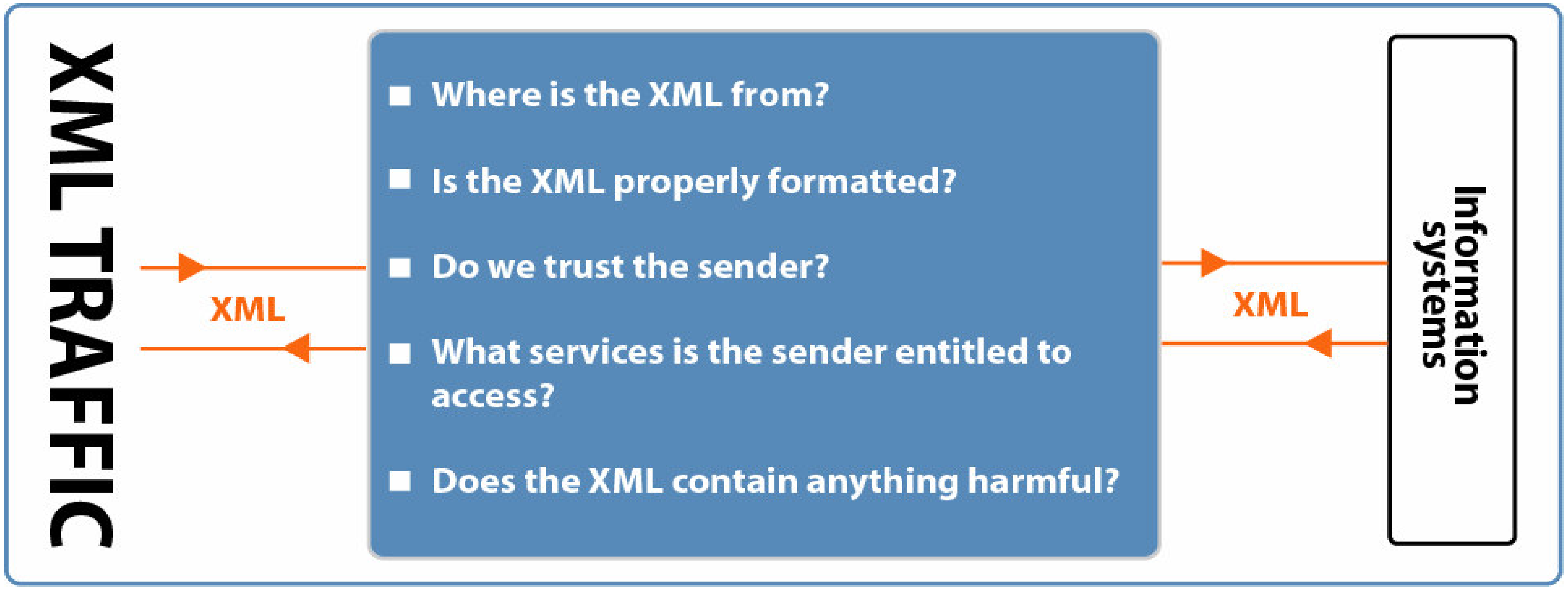
Expression Name:

OWASP Top ten



OWASP Top Ten Most Critical Web Application Security Vulnerabilities		
A1	**Unvalidated Input**	Information from web requests is not validated before being used by a web application. Attackers can use these flaws to attack backend components through a web application.
A2	**Broken Access Control**	Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access other users' accounts, view sensitive files, or use unauthorized functions.
A3	**Broken Authentication and Session Management**	Account credentials and session tokens are not properly protected. Attackers that can compromise passwords, keys, session cookies, or other tokens can defeat authentication restrictions and assume other users' identities.
A4	**Cross Site Scripting (XSS) Flaws**	The web application can be used as a mechanism to transport an attack to an end user's browser. A successful attack can disclose the end user?s session token, attack the local machine, or spoof content to fool the user.
A5	**Buffer Overflows**	Web application components in some languages that do not properly validate input can be crashed and, in some cases, used to take control of a process. These components can include CGI, libraries, drivers, and web application server components.
A6	**Injection Flaws**	Web applications pass parameters when they access external systems or the local operating system. If an attacker can embed malicious commands in these parameters, the external system may execute those commands on behalf of the web application.
A7	**Improper Error Handling**	Error conditions that occur during normal operation are not handled properly. If an attacker can cause errors to occur that the web application does not handle, they can gain detailed system information, deny service, cause security mechanisms to fail, or crash the server.
A8	**Insecure Storage**	Web applications frequently use cryptographic functions to protect information and credentials. These functions and the code to integrate them have proven difficult to code properly, frequently resulting in weak protection.
A9	**Denial of Service**	Attackers can consume web application resources to a point where other legitimate users can no longer access or use the application. Attackers can also lock users out of their accounts or even cause the entire application to fail.
A10	**Insecure Configuration Management**	Having a strong server configuration standard is critical to a secure web application. These servers have many configuration options that affect security and are not secure out of the box.

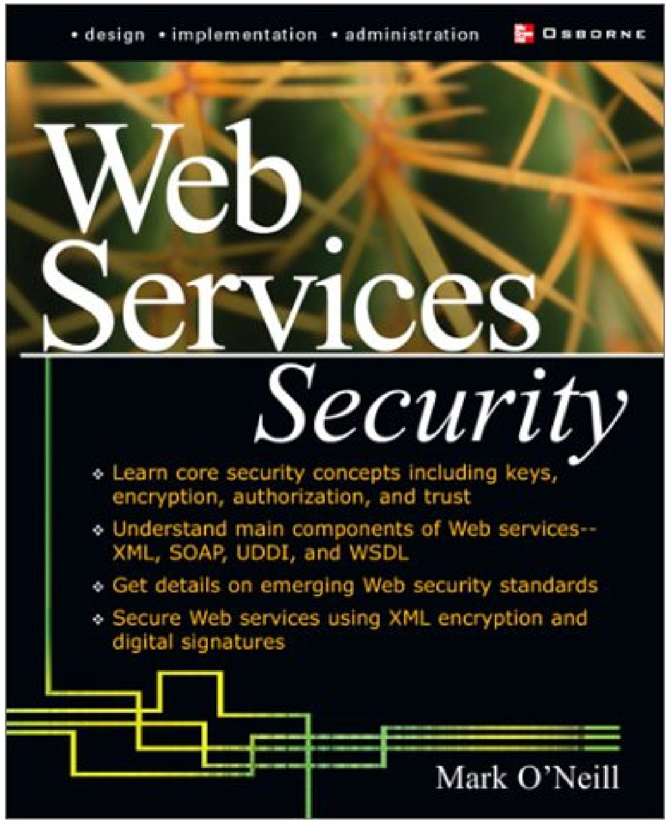
Questions you need to answer before thinking about standards



- Where is the XML from?
- Is the XML properly formatted?
- Do we trust the sender?
- What services is the sender entitled to access?
- Does the XML contain anything harmful?

Read the book!

Vordel CTO, Mark O'Neill author of "Web Services Security"


• design • implementation • administration

Osborne

Web Services *Security*

- Learn core security concepts including keys, encryption, authorization, and trust
- Understand main components of Web services--XML, SOAP, UDDI, and WSDL
- Get details on emerging Web security standards
- Secure Web services using XML encryption and digital signatures

Mark O'Neill

VORDEL®

Thank You

tony.palmer@vordel.com