Web & XML/Web Service Security

Ingeniería de Aplicaciones para la Web Semántica

Guest Lecturer:

Javier Echaiz

jechaiz@cs.uns.edu.ar

JaviZ' Ice breaker! ☺

"On the Internet, nobody knows you're a dog."

El mito

“Nuestro web site/service está a salvo”:

- Tenemos firewalls.
- Firmamos/Encriptamos nuestros datos.
- Autenticamos a nuestros usuarios.
- Tenemos una política de privacidad.

Historias de terror

Qwest Glitch exposes customer data
Securityfocus.com
May 23, 2002

Hackers attack eBay accounts
— ZDNet News
Mar 25,2002

Gov't Web Sites open to Hack Attacks
CBS News, Jan 25, 20

Sites Revealed Passwords For Thousands Of Ameritech Users
— NewsBytes Feb 22,2002

NY Times Internal Network Hacked
— Internet.com, Feb 27, 2002

Hacker Accesses 8 Million Credit Cards
- CNN, Feb 1

hacker theft of sensitive documents
- ComputerWorld A

Vivendi Says Online Shareholder Voting Hacked
— NewsBytes, Apr 29, 2002

Software bug bites U.S. Military
— BBC, Mar 18,2003

Glitch at Fidelity Canada exposes customer information
— ComputerWorld, May 30, 2002

Security worries hold back UK online tax returns
— TheRegister, Jan 29,2003

Hackers steal student soc.sec. numbers
— ABCNews, Mar 6,2003

FTD.com hole leaks personal information
— CNet, Feb 13, 200

Requerimientos de seguridad

Autenticación: ¿sos quién decís ser?

Autorización: ¿tenés permitido tenerlo?

Confianza: ¿acepté trabajar con vos?

Integridad: ¿fue alterado antes de que yo lo obtuviese?

Confidencialidad: ¿puede un “extra” verlo?

Auditoría: ¿puedo probar que pasó?

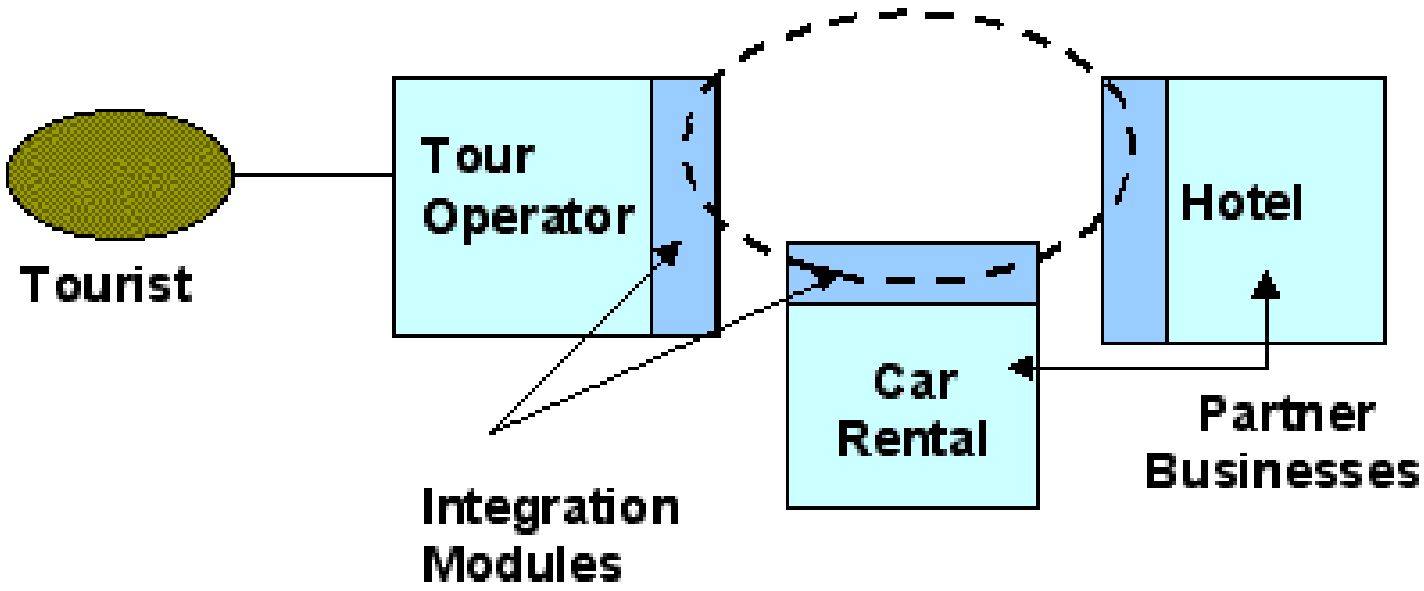
No-repudio: ¿podés argumentar que no lo enviaste/recibiste cuando en realidad si lo hiciste?

Soluciones generales

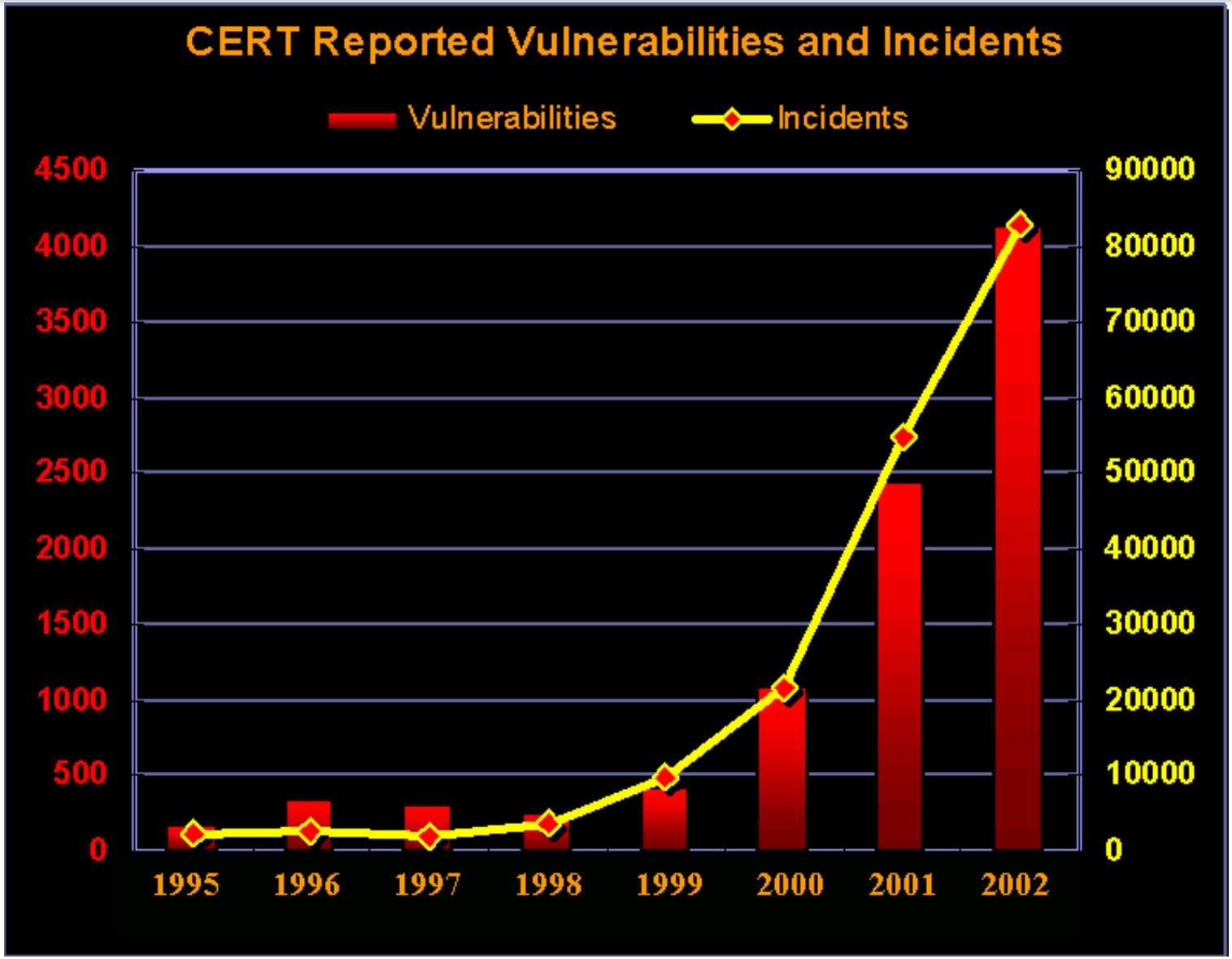
- **Autenticación:** usuario/password, firma digital basada en password y verificación de firma, *challenge-response*, biométrica, smart cards, etc.
- **Autorización:** aplicación de políticas, control de acceso, capacidades, gestión de derechos digitales.
- **Confianza:** a partir de la verificación de la firma digital.
- **Integridad:** Message Digest, autenticado mediante firma digital.
- **Confidencialidad:** encripción/desencripción mediante claves.
- **Auditoría:** logueos encriptados para evitar el *tampering*.
- **No-repudio:** firmado/verificación con firma digital, confiabilidad del mensaje.

Here Comes the Web Services

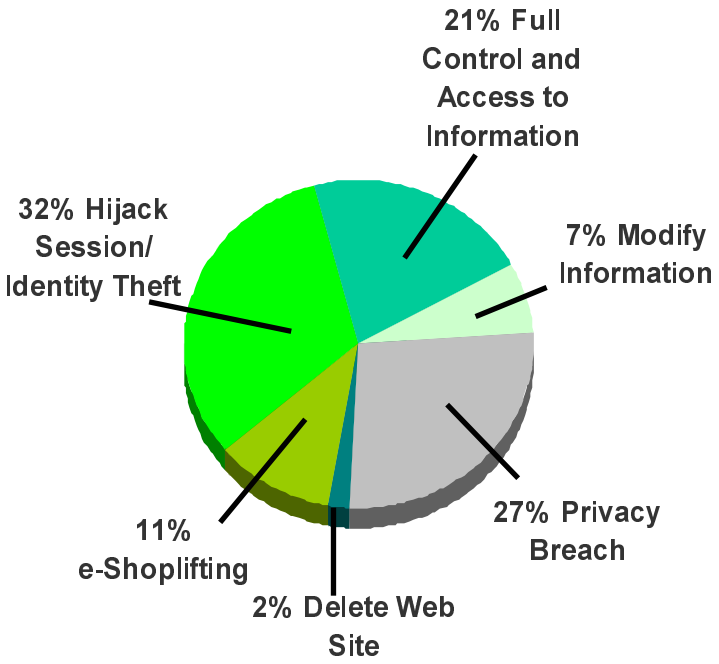
Web Services provides cross-enterprise integration



Cyber Crime & Incidents on the Rise



Why Application Security Defects Matter



>1000 application 'Healthchecks' with AppScan – 98% vulnerable: all had firewalls and encryption solutions in place…

Frequent

- 3 out of 4 business websites are vulnerable to attack (Gartner)

Pervasive

- 75% of hacks occur at the Application level (Gartner)

Undetected

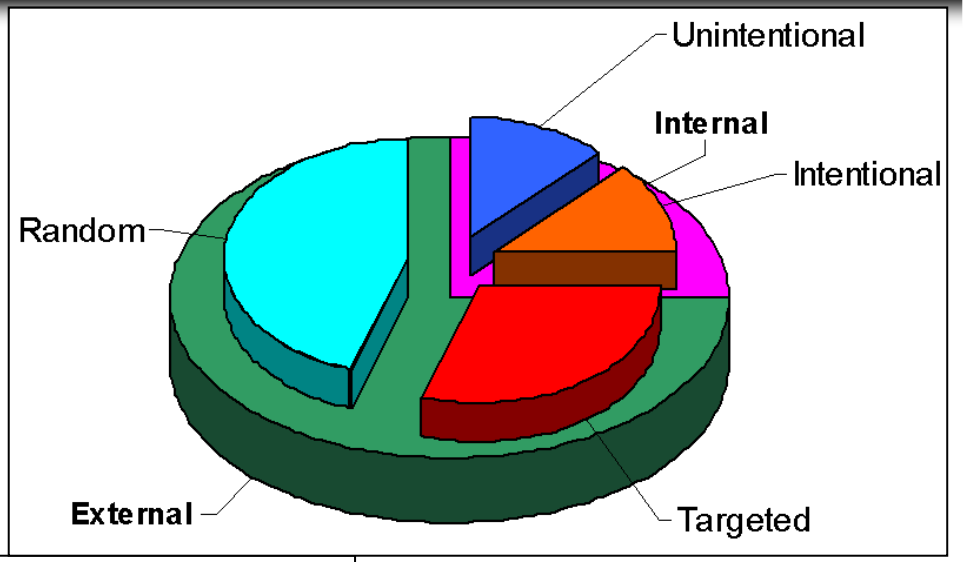
- QA testing tools not designed to detect security defects in applications
- Manual patching - reactive, never ending, time consuming and *expensive*

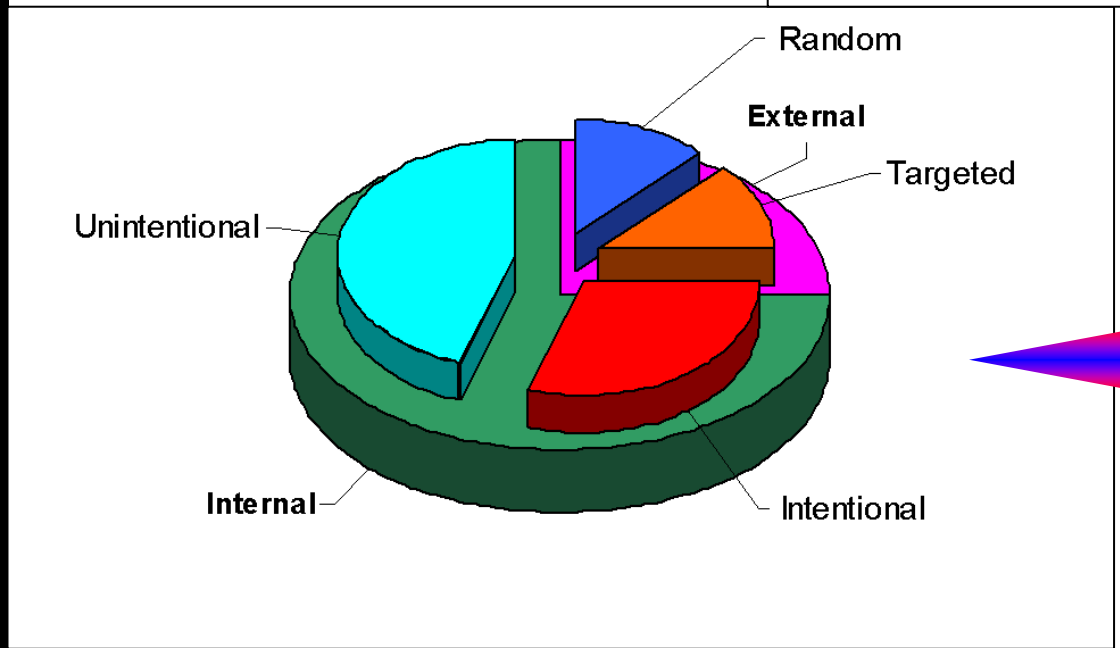
Dangerous

- When exploited, security defects destroy company value and customer trust

The Threat is Real

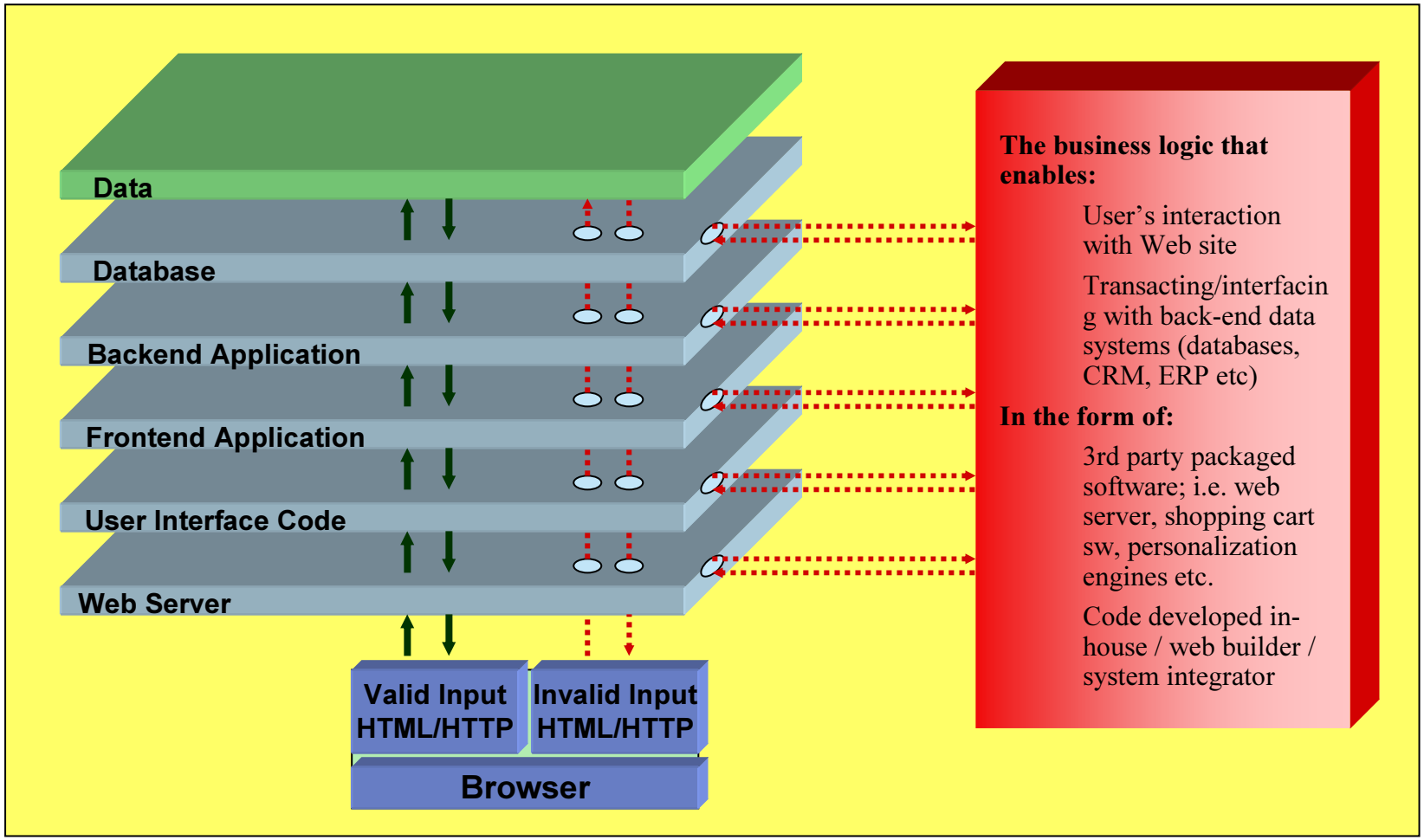
Security Threat





Privacy Threat

Web Application Vulnerabilities



Without any protection, holes and backdoors exist at every layer waiting to be exploited

CWVs vs. ASVs

Common Web Vulnerabilities (CWVs): vulnerabilities found in the site's technical building blocks including, CGI scripts, Web Servers, Application Servers, or Database Servers.

In general, a CWV is:

- an unintended consequence of either flawed design or development of the web application technology
- a misconfiguration of the 3rd party software

Application Specific Vulnerabilities (ASVs): unique to a specific application and native to the specific programming and configuration of the application itself - not the underlying technologies

- exploits a software bug at the *business logic layer* of a specific application

Ten Types of Application Hacks

Through a browser, a hacker can use even the smallest bug or backdoor to change, or pervert, the *intent* of the application

Threat	Application	Negative Outcome
Buffer overflow	Form field: collect data	Crash Server
Cookie poisoning	Customer account	Session Hijacking
Hidden fields	Online shopping	Alter prices, Defacement
Debug options	Any Unsanitized code	Admin Access
Cross Site scripting	Text Field: collect data	Identity Theft
Stealth Commanding	CGI, Backend	Direct O/S/Application Access
Parameter Tampering	Data Fields	Fraud, Data Theft, D/L DB
Forceful Browsing	Web Server	Unauthorized Site/Data Access
3rd Party Misconfiguration	Front/Back end Apps	Admin Access
Published Vulnerabilities	All tools	Admin Access, Crash Server

Commonly known attacks will fall under one or more of these categories.

I.E. SQL Injection – a type of parameter tampering with stealth commanding.

Application Layer Threats

	User Interface	Web Server	Front end Application	Backend Application	Database
HIDDEN MANIPULATION		●	●	●	
COOKIE POISONING		●	●	●	
BACKDOOR & DEBUG OPTIONS		●	●	●	●
BUFFER OVERFLOW		●	●	●	●
STEALTH COMMANDING		●	●	●	●
3RD PARTY MISCONFIGURATION	●	●	●	●	●
KNOWN VULNERABILITIES	●	●	●	●	●
PARAMETER TAMPERING		●	●	●	●
CROSS SITE SCRIPTING	●				
FORCEFUL BROWSING		●			

Everyone Contributes across the Application Lifecycle

- **Develop** (*Developer):*
  - Construct application
  - Unit test application components

- **Test** (*Tester/ QA Engineer):*
  - Create test plan
  - Create, run & manage test scripts
  - Defect assignment & tracking
  - Delta and results analysis
  - Approve release to production

- **Audit** (*Ops & Security Auditor):*
  - Create operations plan
  - Deploy & maintain business compliance
  - Scheduled (or not!) application audits

Web Services Security Drivers

Web Services Security Drivers

Enabling E-Business

Availability:
Can you do what you need to do when you want to do it?

Authorization:
What are you allowed (wanted) to do?

Authentication:
Who are you?

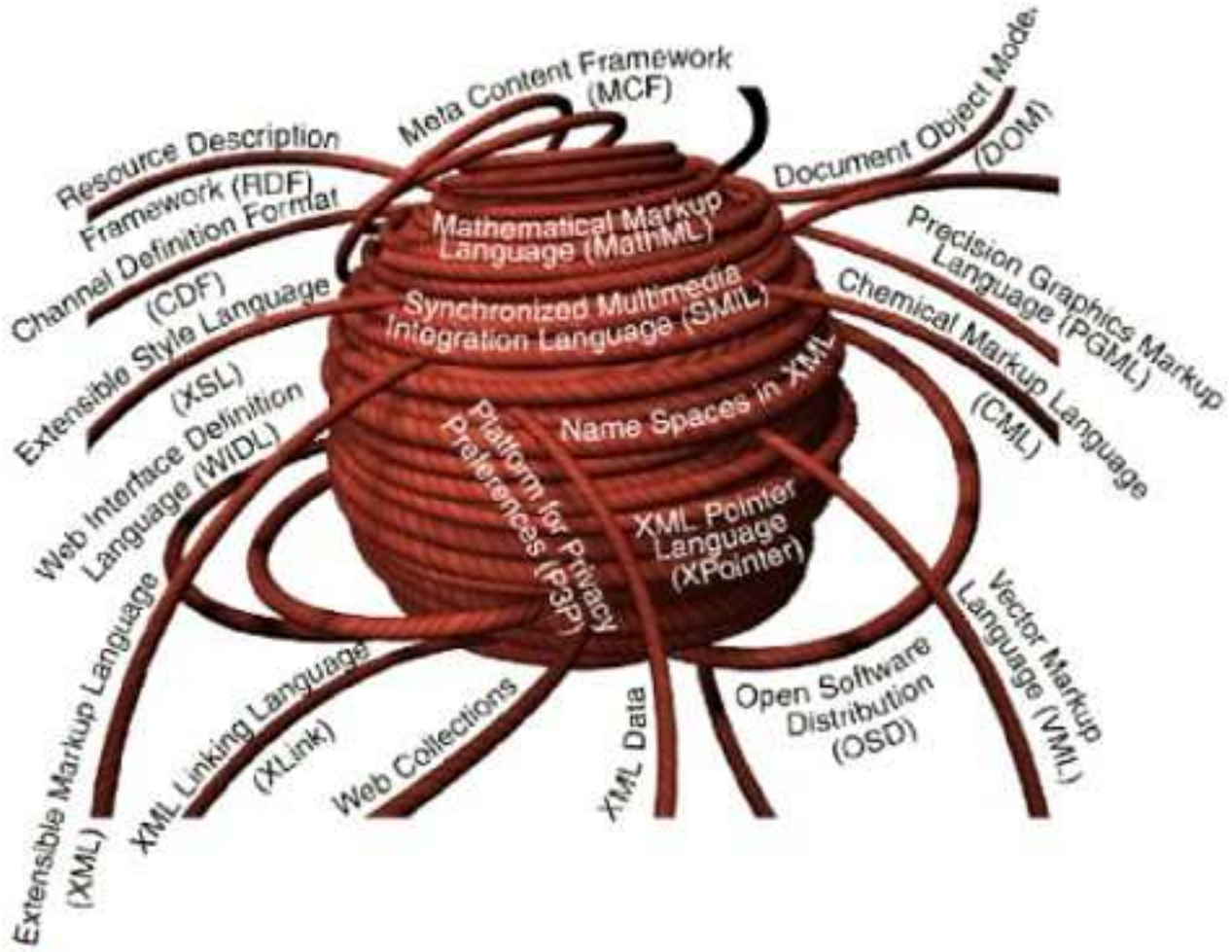
Attack Prevention
How do we protect against malicious users?
How do we prevent attacks?

Privacy:
Can we protect your data?

Nonrepudiation:
Can we prove you did it?

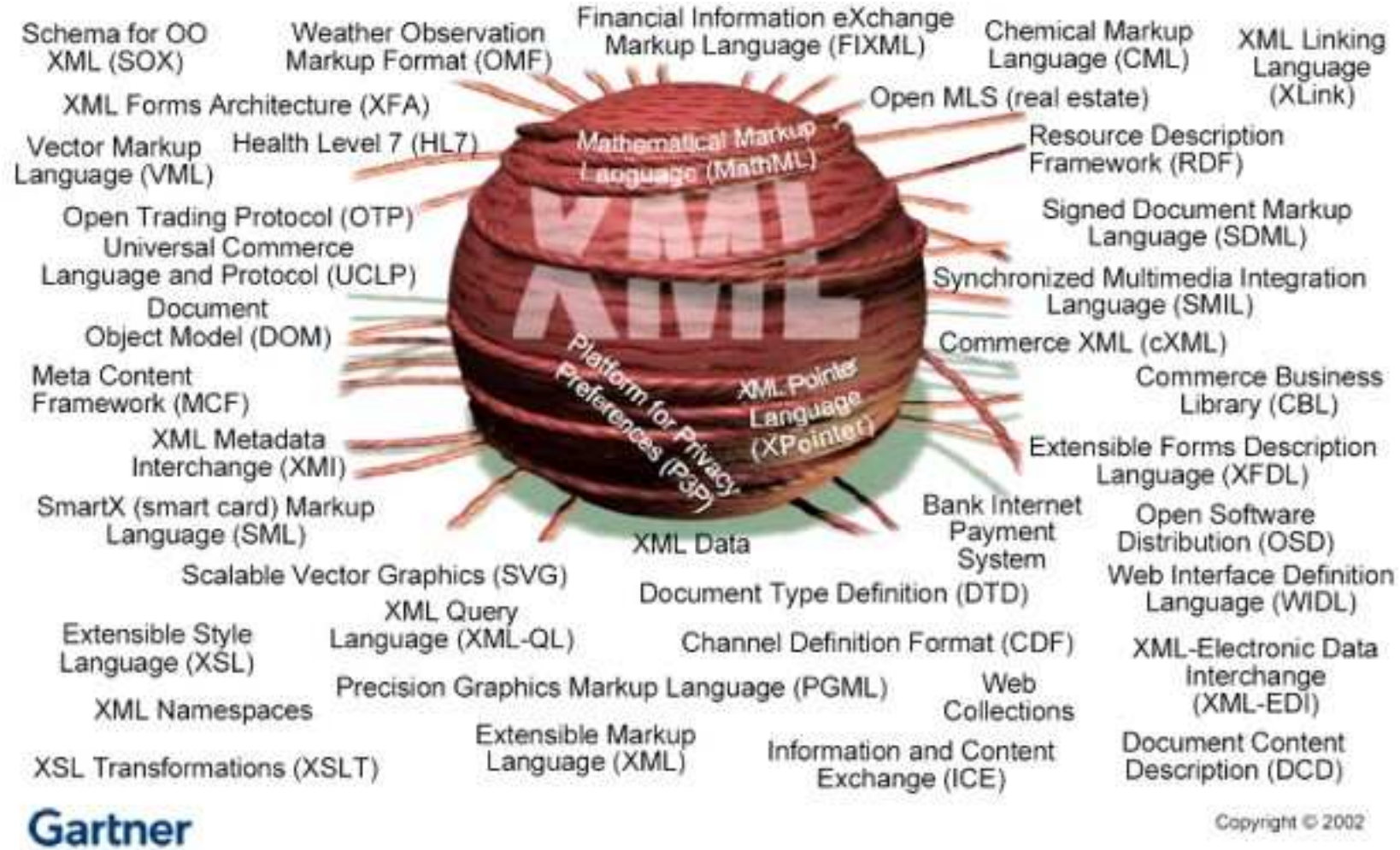
Security Foundation:
Speed, flexibility and integration

All Tied Up With XML



Gartner

All Tied Up With XML



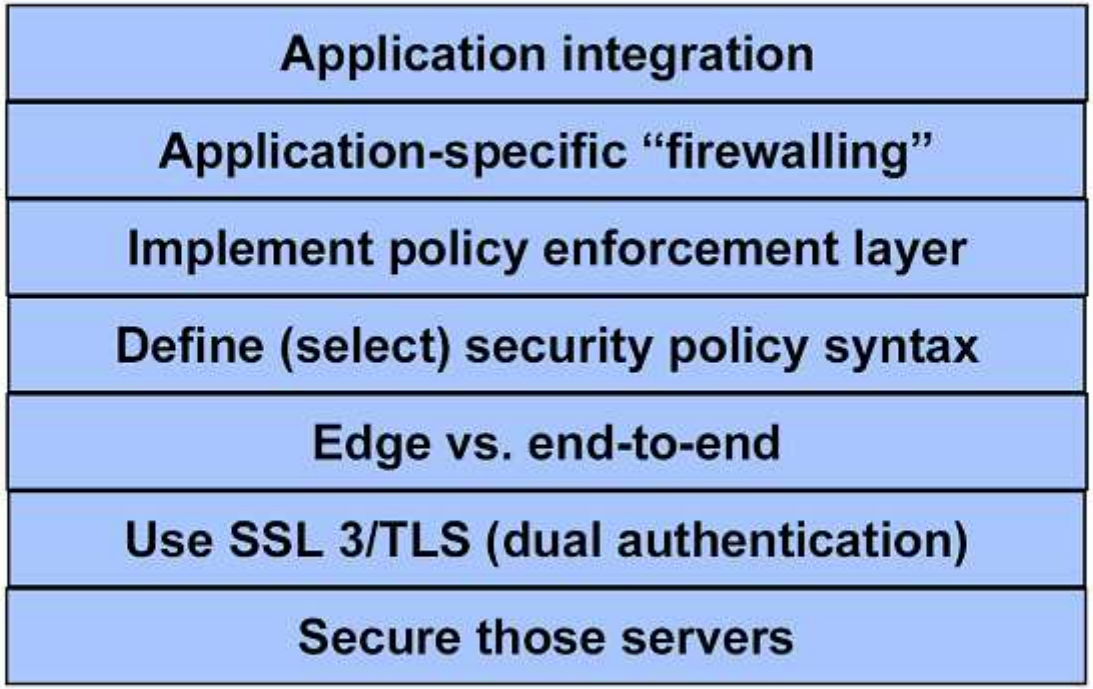
All Tied Up With XML (STILL!!!!!)

All Tied Up With XML (STILL!!!!!)

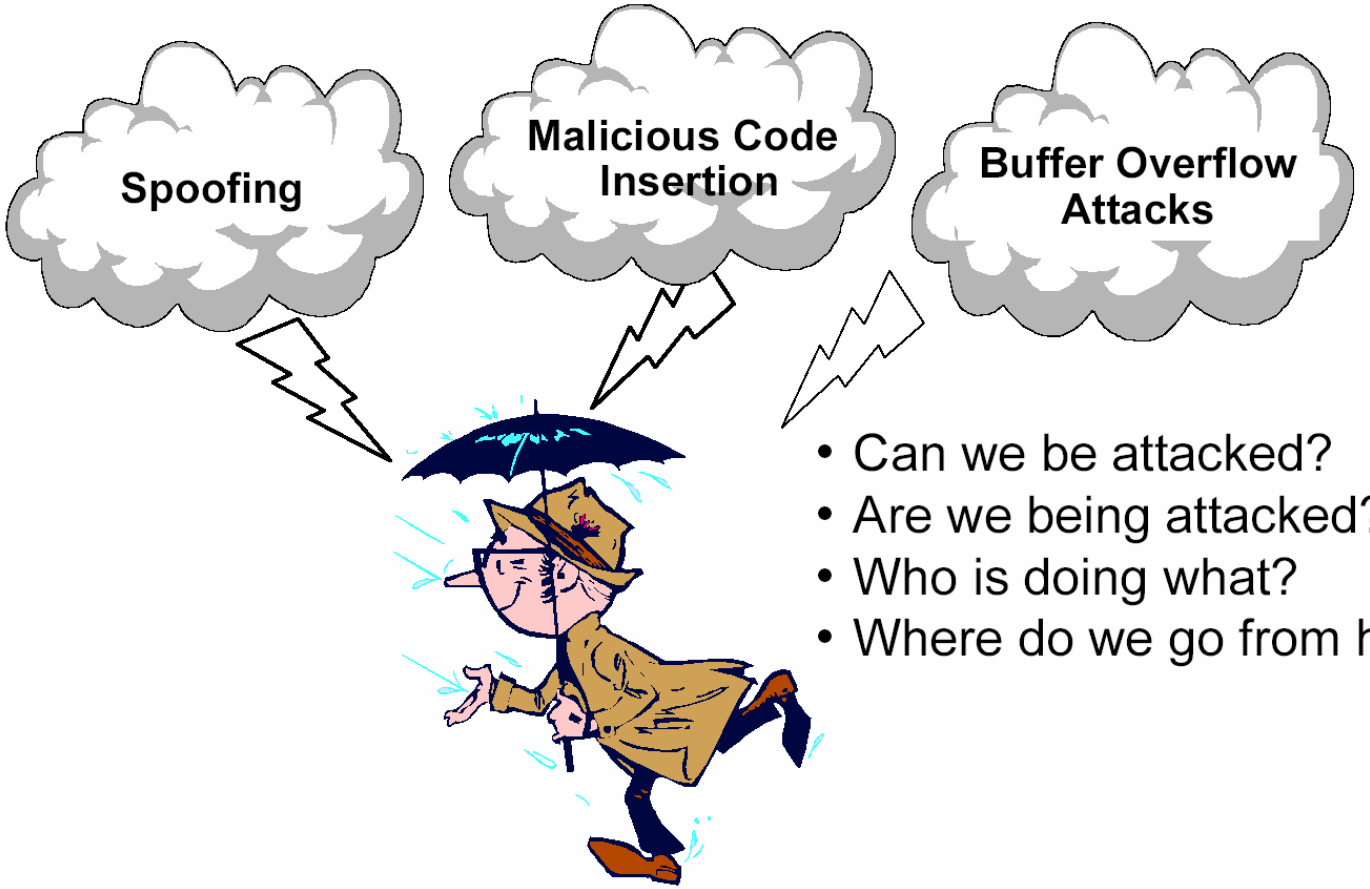
suicides allowed!!!

Gartner

Web Services Security Layers



Web Services Threats



- Can we be attacked?
- Are we being attacked?
- Who is doing what?
- Where do we go from here?

XML/Web Services Attack Vectors

Old Attacks still valid

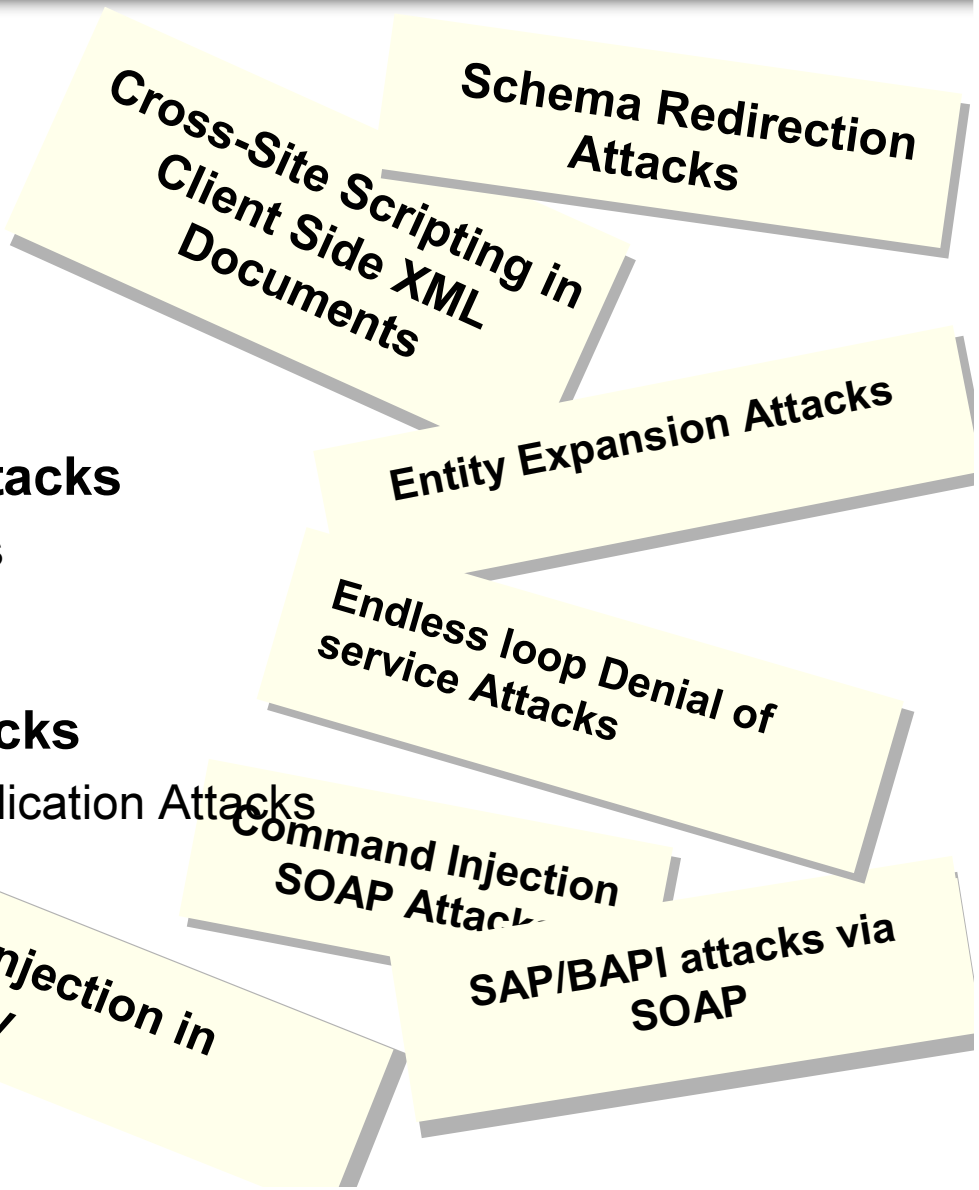
- CWV's
- Injection Attacks
- Buffer Overflow
- Denial of Service

The New Manipulation Attacks

- Entity and Referral Attacks
- DTD and Schema Attacks

The Next Generation Attacks

- Web Service Enabled Application Attacks
- Multi-Phase Attacks



XML Attack Example (Entity Expansion)

An attack on **XXX** Application Server

1. Find a web service which echoes back user data such as the parameter "in"
2. Use the following SOAP request
3. And you'll get C:\WinNT\Win.ini in the response (!!!)

```
...
<!DOCTYPE root [
    <!ENTITY foo SYSTEM
"file:///c:/winnt/win.ini">
]>
...
<in>&foo;</in>
```

How it works:

A. XXX App Server expands the entity “foo” into full text, gotten from the entity definition URL - the actual attack takes place at this phase (by **XXX** Application Server itself)

B. **XXX** App Server feeds input to the web service

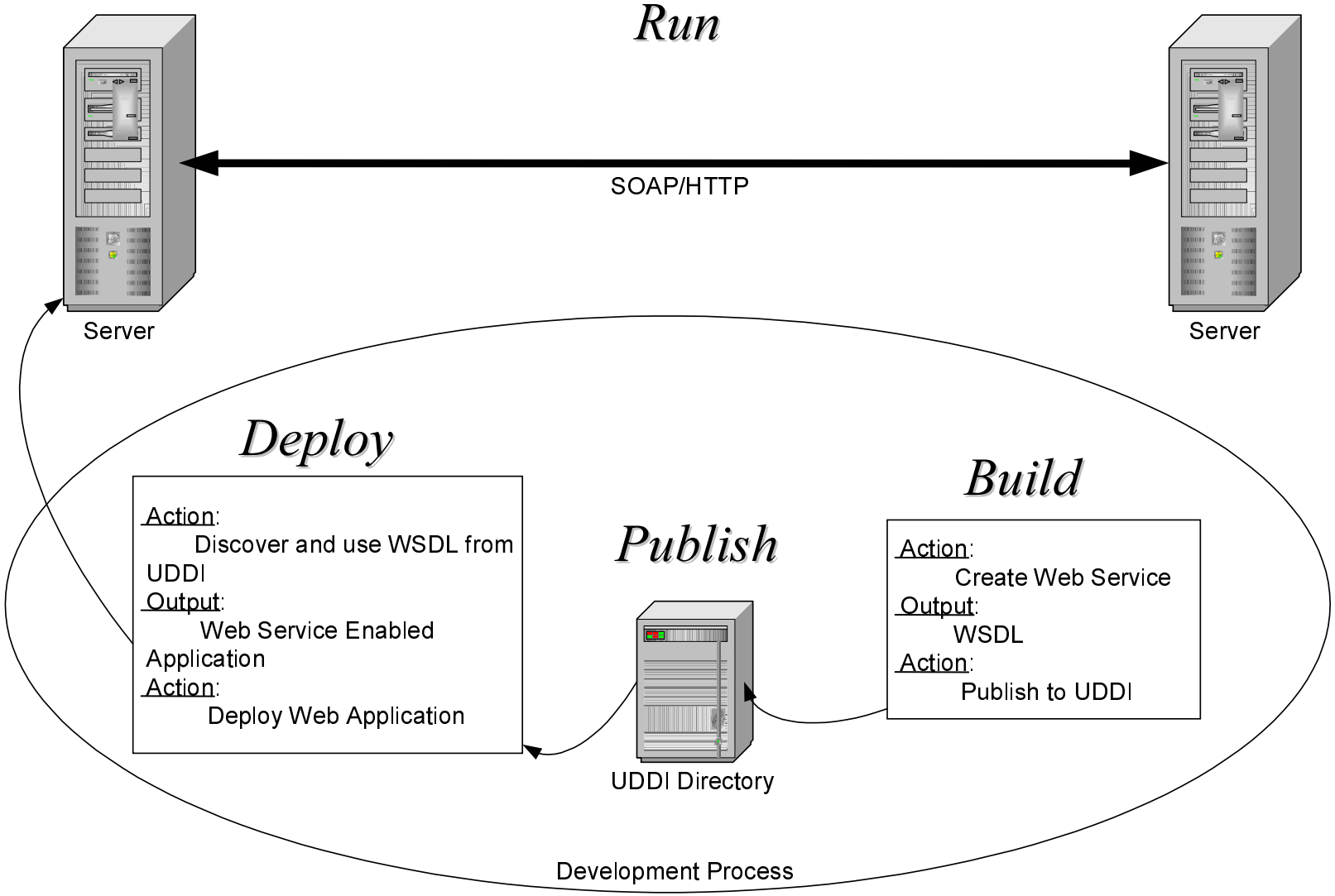
C. The web service echoes back the data

Note: if the file contains "<" or "&" then this method may not work. The file must either be well formed XML document or contain no tags/special characters. But /etc/passwd does not contain these.

Phases of Implementing Web Services

- ***Build***
  - New and existing applications are described in WSDL or as a XML Schema and a SOAP engine is deployed in front of the application
- ***Publish***
  - The WSDL is published to UDDI or the schema/WSDL is published directly to the relevant developers
- ***Deploy***
  - Web Application developers search for the Web Services available to them
  - A Web Application is created using the Web Service(s)
  - The Web Application is deployed to the Internet
- ***Run***
  - The Web Application calls the Web Service via SOAP over HTTP(S)

Web Services Lifecycle Architecture



Security for each Phase

- ***Build***
  - Secure Coding Practices
  - Secure Development Processes
- ***Publish***
  - UDDI Security
  - Publish Security measures taken
- ***Deploy***
  - Secure Coding Practices
  - Secure Development Processes
- ***Run***
  - Combination of Network and Application level security measures

When to Apply Security

- ***Pre-Deployment***
  - Identify and Fix security related defects early in the lifecycle
  - Control Access to Web Services
- ***Post Deployment***
  - Implement common best practices
    - Access control, Authentication and Authorization
    - Encryption
    - Intrusion/Attack prevention
    - Audit

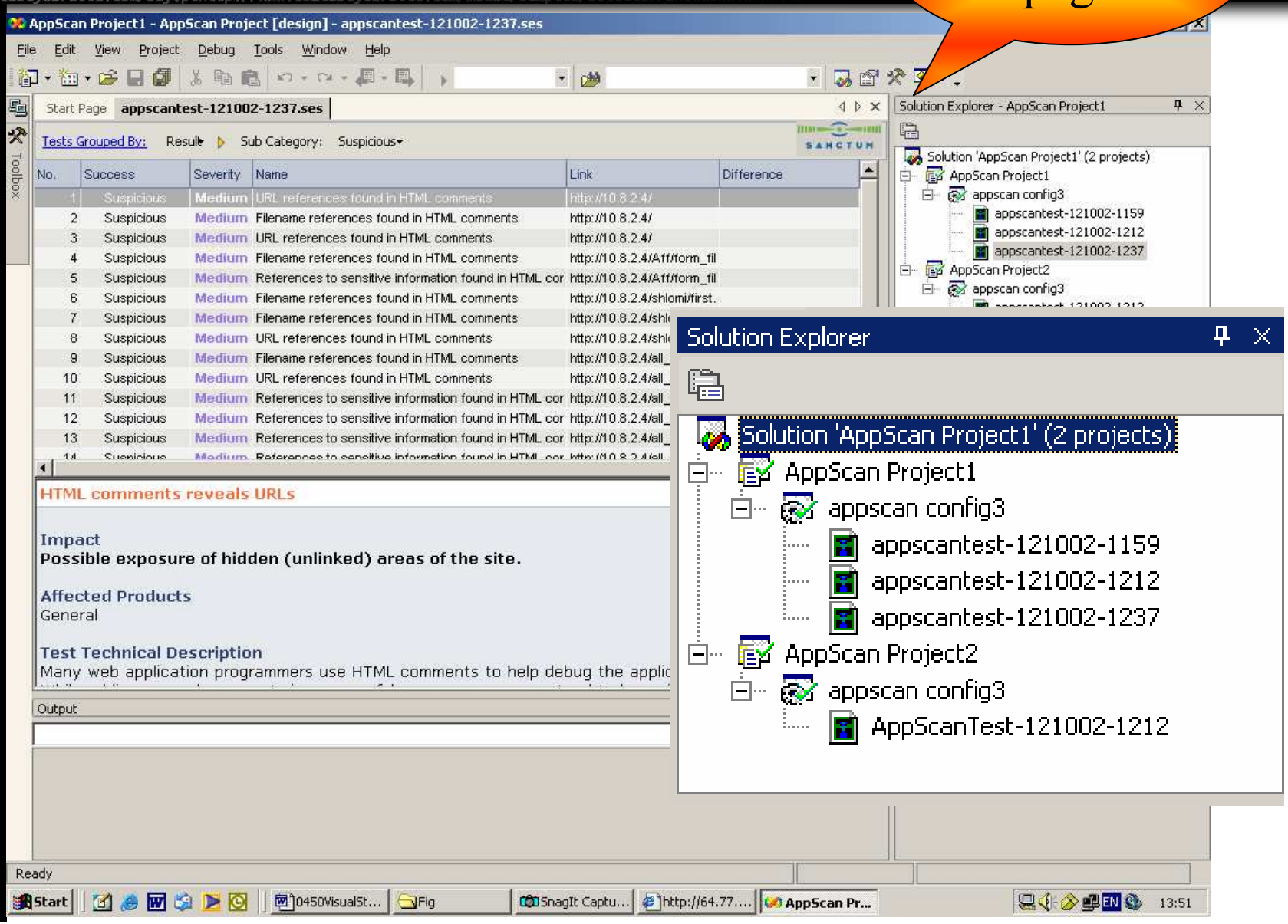
Where to apply Web Services Security

- ***Perimeter***
  - The “Swiss Cheese Firewall”
  - Web Services take advantage of existing ‘holes’ in the perimeter
    - Port 80/443 HTTP(s)
  - Lack of Application layer / Layer 7 awareness in the network
  - The Business Units, NOT IT, are responsible for the creation and deployment of and Access to Web services
    - IT manages the production environment

- ***Application Layer Security for Web Services***
  - Allows for tight adherence to business logic
  - Allows for granular Access Control and Authentication
  - Supports the ad hoc and aggressively evolving nature of Web Services enabled applications

AppScan DE 1.7: Visual Studio .NET Integration

M$ Propaganda

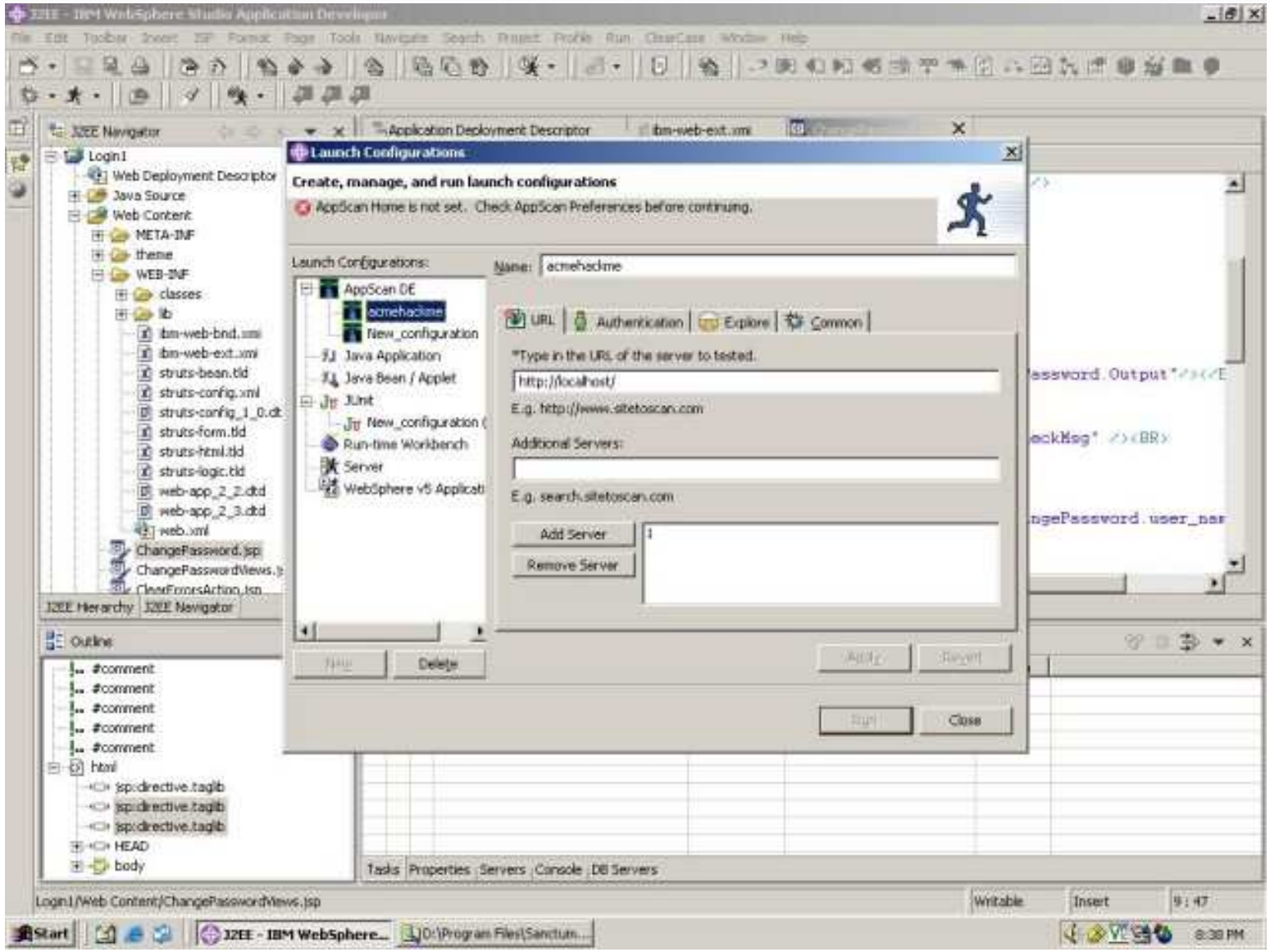


AppScan DE 1.7 Features for Visual Studio .NET Developers

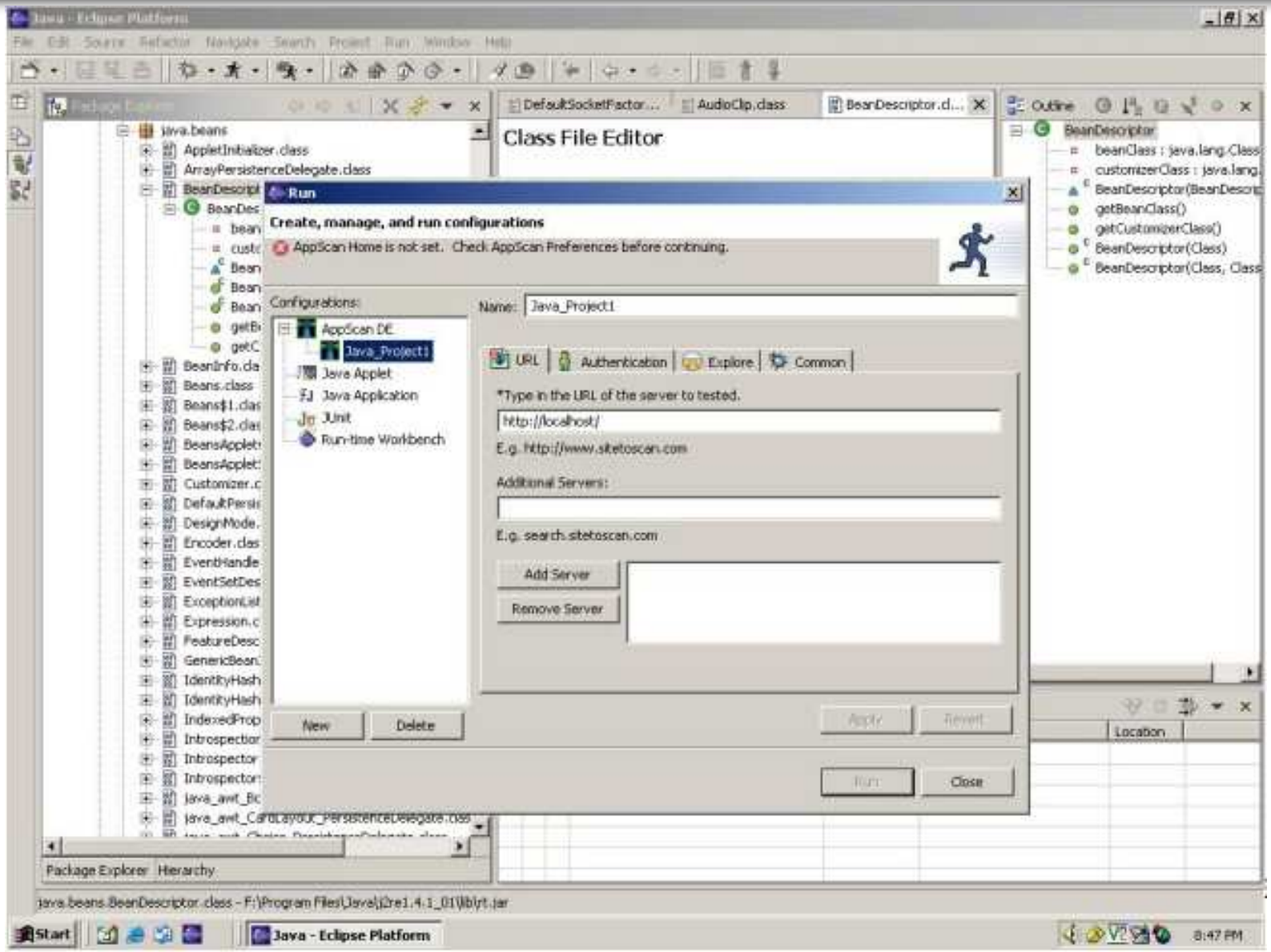
Complete Integrated Development Environment with AppScan DE

- Integrated into Visual Studio . NET project hierarchy as AppScan Projects, Configurations & Test Runs
- Logical organization of all security unit testing projects and configurations for Visual Studio .NET solutions
- AppScan Configured and Launched from within Visual Studio .NET as part of normal workflow
- Single click scan automatically tests web applications written in any language supported by Visual Studio .NET Including VB, C#, C++, and J#
- Provides customizable configuration settings to enable efficient security testing as part of the development cycle
- Review ‘developer centric’ test results and specific inline real time fix recommendations directly from within the Visual Studio .NET development environment

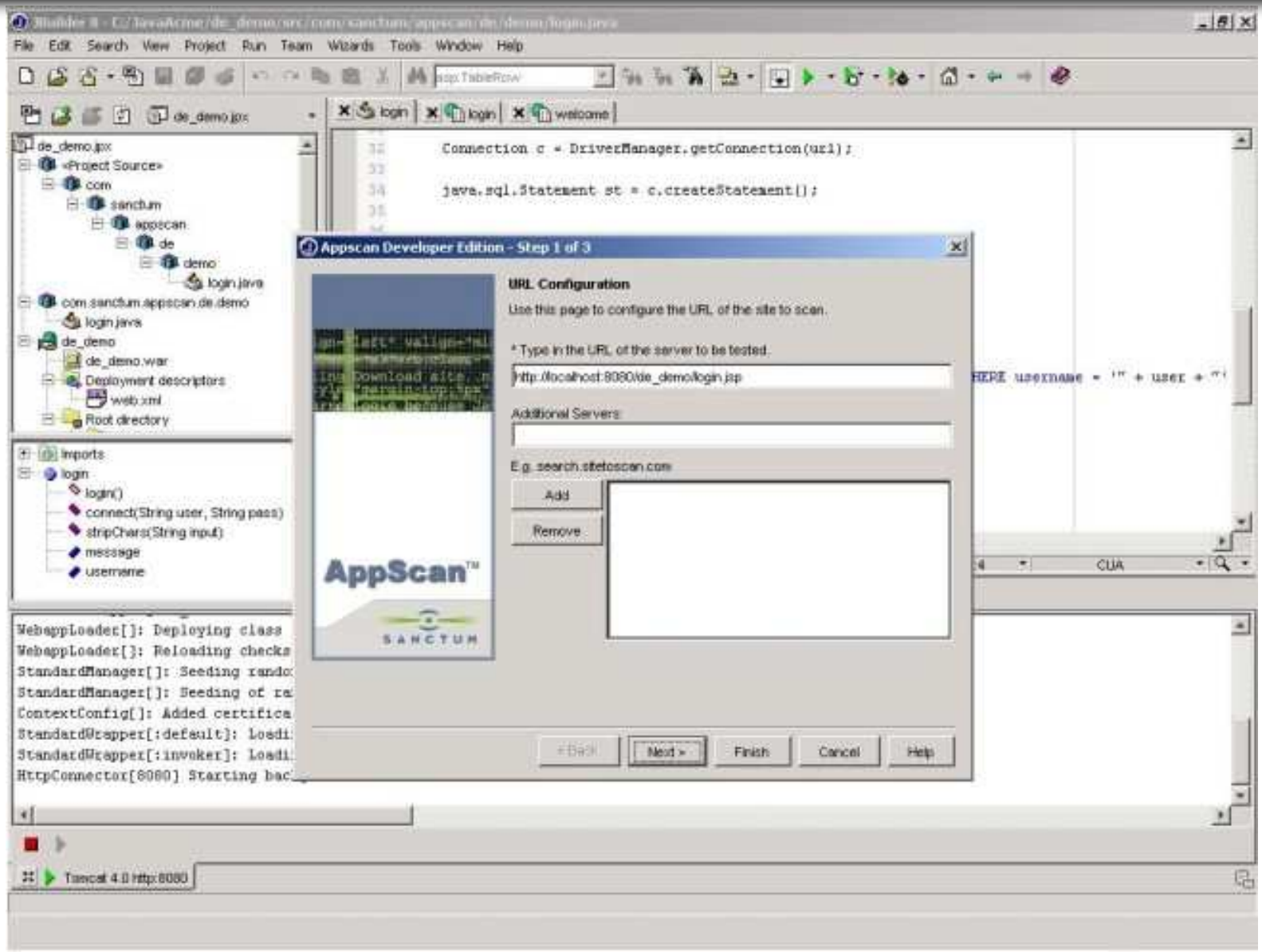
AppScan DE 1.7: WebSphere Studio 5.0 Plug-in



AppScan DE 1.7: Eclipse 2.0/2.1 Plug-in



AppScan DE 1.7: JBuilder v8/9 Plug-in



AppScan DE 1.7 Features for Java and VB6.0 environments

Complete Integrated Development Environment with AppScan DE for:

WebSphere Studio 5.0, Eclipse 2.0/2.1, JBuilder v8/9, and Visual Basic 6.0

- ✓ Streamlined security testing - AppScan is configured and launched as normal part of workflow from within IDE using native IDE Plug-in
- ✓ User can set default values for the scan properties, or change them on the fly for every scan.
- ✓ Single click scan automatically tests web applications written in any language/environment supported by the IDE including Java, EJB, Servlets JSP, HTML, etc.
- ✓ Provides customizable configuration settings to enable efficient security testing as part of the development cycle
- ✓ Review 'developer centric' test results and specific inline real time fix recommendations

This Slide Intentionally Left Blank

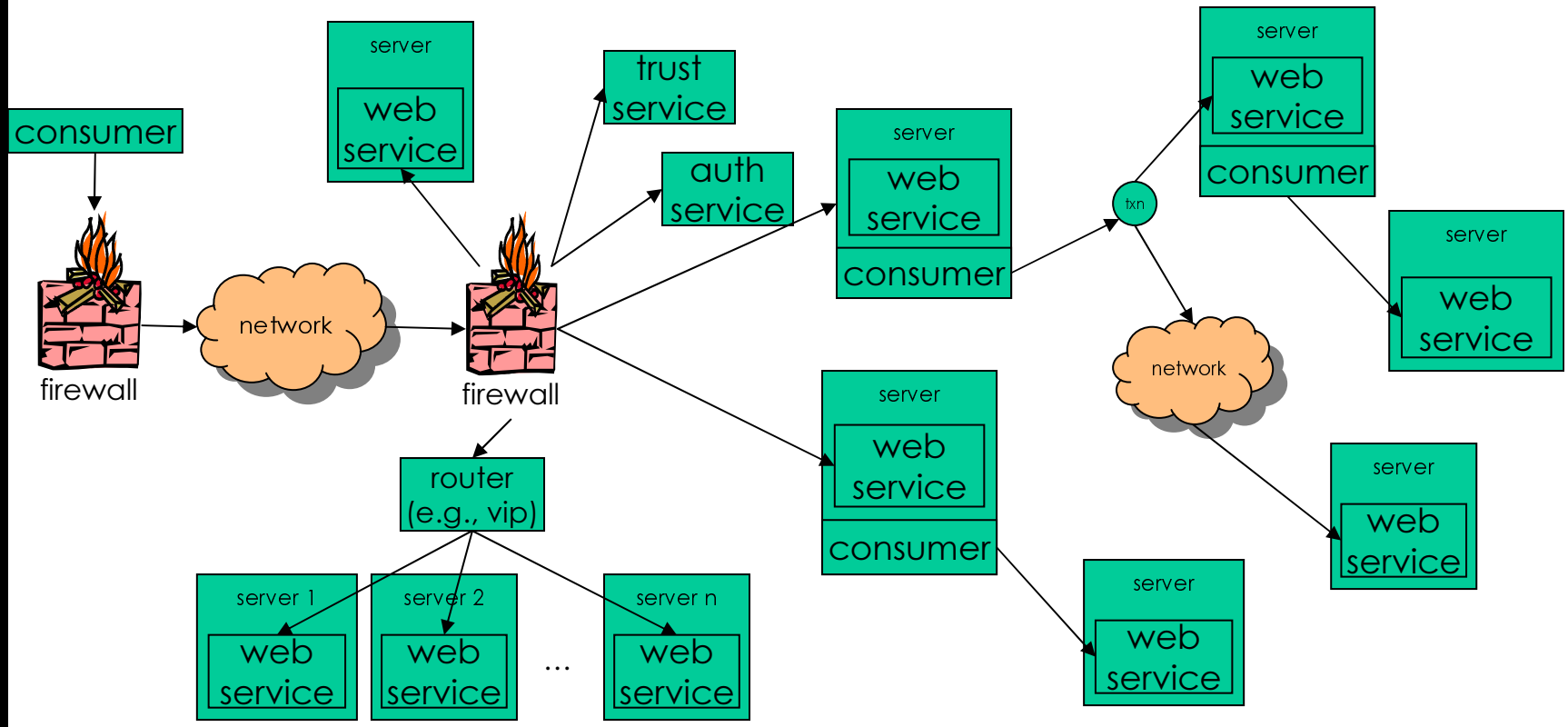
Someone's still with me???

Simple web services

A web service invocation embodies a client/server interaction over open, free and readily available technologies

- the request and response are encoded (marshaled) in XML using the Simple Object Access Protocol (SOAP)
- service references are encoded in XML using the Web Services Definition Language (WSDL)
- SOAP may be implemented over any transport protocol, but HTTP is most common

Many configurations



left alone, all of the traffic through this diagram is XML (plaintext) over HTTP (plaintext) on port 80

Web services and security

What is the right mix between price, performance, robustness, flexibility to an agile enterprise, complexity and exposure to risk?

- under web services, we have to become comfortable with our decisions all over again

the dangers

- becoming overconfident in the face of unacceptable exposure
- locking down our systems at the expense of adaptability

Web service security positions

1. web services are really only useful internally, so security is not a concern
2. web services cannot be secured, and pose a significant threat to the security of an otherwise robust enterprise
3. web services can be secured using SSL and password authentication, just like e-commerce sites on the web
4. SSL is not sufficient to secure web services, but I do not have a basis for figuring out just what level of security I need or what options I have

Security domains of concern

- authentication
  - ensures that we know and approve access for the identity of a party in a given security domain
- authorization
  - ensures that an authorized entity has access to a controlled subset of all available secured resources
- confidentiality
  - ensures that only authorized parties can understand a secured message
- integrity
  - ensures that a message arrives at its destination unaltered from the point it left its sender
- non-repudiation
  - ensures that a sender cannot deny that he/she sent a given message; binds a transaction to a non-refutable identity

Impact on web services

Questions arise because of the plaintext concerns over a simple WS architecture

- how to perform authentication/authorization?
- how to guarantee integrity?
- how to enforce confidentiality and non-repudiation?

A starting point for authentication

Let's start with the simplistic authentication provided by many e-commerce web sites

- HTTP BASIC-AUTH
- user name and password are encoded in the HTTP stream as Base64 encoded plaintext
  - stored in an HTTP header
  - Authorization: Basic U2thdGVib2FyZdhcmVo…
- in this mode, simple Base64 decoding reveals the credentials
  - there is no encryption involved

Moving beyond BASIC-AUTH

- While BASIC-AUTH is pervasive, it is not secure
- for many, the next step is to secure the BASIC-AUTH transmission using HTTPS
  - HTTP is secured using the Secured Sockets Layer (SSL)
    - SSL encrypts the messages passed back and forth in the HTTP conversation, including the BASIC-AUTH header
    - however, we mentioned earlier that SSL was not sufficient to secure web services
      - *let's talk about what is missing…*

Why SSL is sufficient in e-commerce applications

1. transactions are generally conducted within the web application context at the e-commerce site
   - there are no intermediaries or multi-party transactions
2. SSL conversations are conducted point-to-point
3. as long as the consumer can remit payment, user credentials are “good enough” to authenticate and authorize their transaction
   - meanwhile, e-commerce sites cannot generally do anything to gain non-repudiation with their customers
4. individual transactions are relatively small and will not “break the bank,” when compared with total throughput

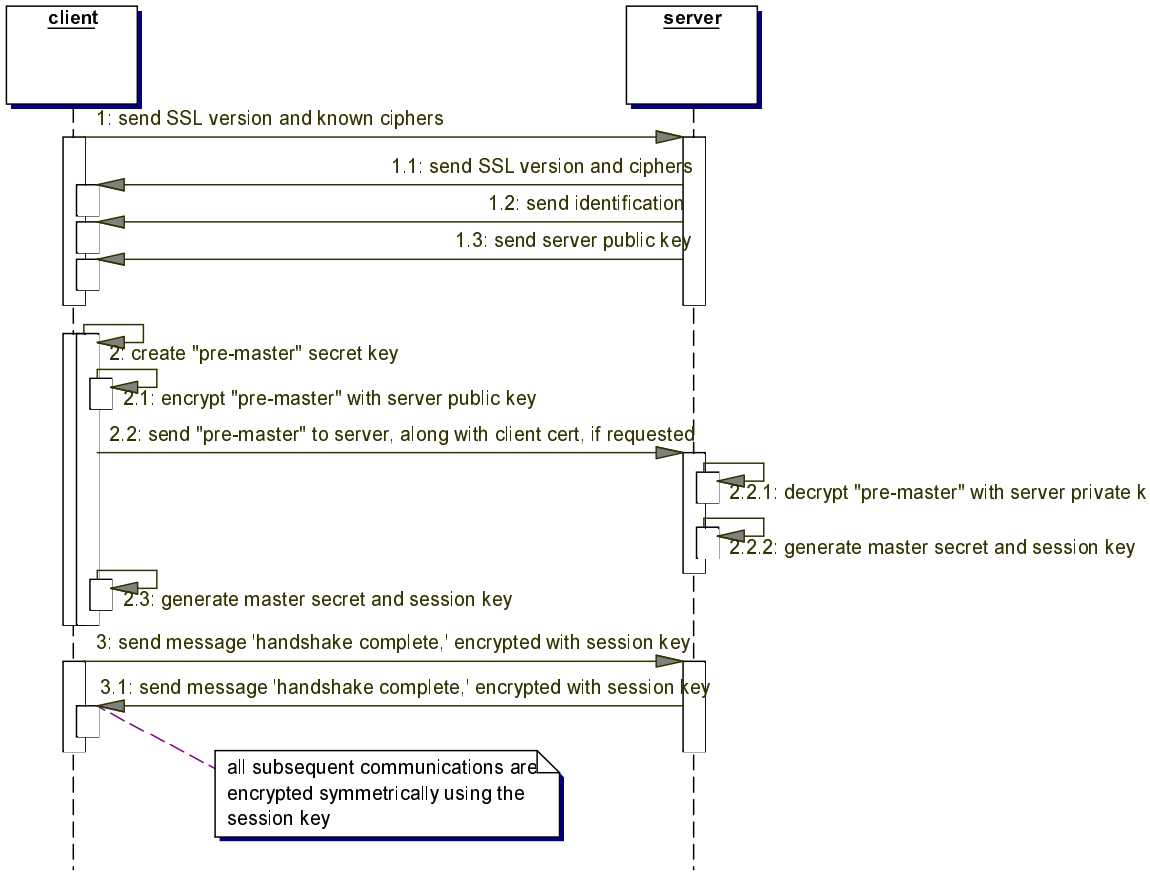
on the other hand…

- an individual web service transaction can involve literally millions of dollars of potential risk exposure, versus a shopping experience at amazon.com
- remember that web services are *systems* transacting with *systems*
  - an open communication channel could be the conduit for a large volume of transacting data

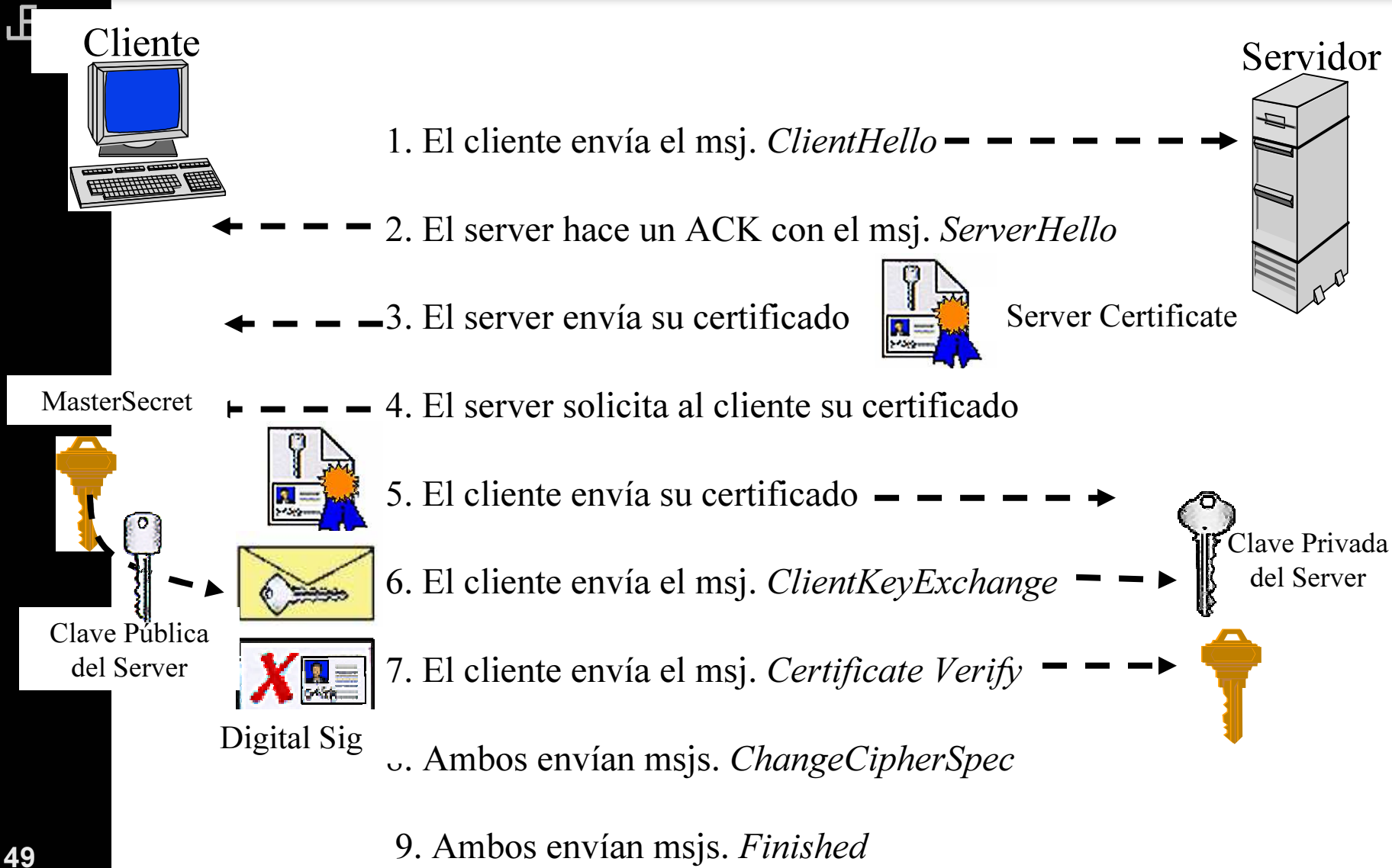
What about client certificates?

- about this time, someone asks the question above…
  - the basic mechanism for authentication breaks down when we start asking a *system* to supply a user name and password anyway
    - have you ever seen a user name and password coded into system algorithms???
    - have you ever abused a user name and password that you learned from application code???
  - client certificates are one analogous, but more secure, means for authentication
  - a *certificate* is a token that contains credentials for asynchronous encryption that remain **confidential** to its owner

the SSL handshake



Handshake Protocol (2)



The benefit of client certificates

Client certificates allow us to create a secured SSL channel that guarantees non-repudiation

…so if we secure BASIC-AUTH over HTTPS using client certificates, is that enough???

The essential issue re: SSL

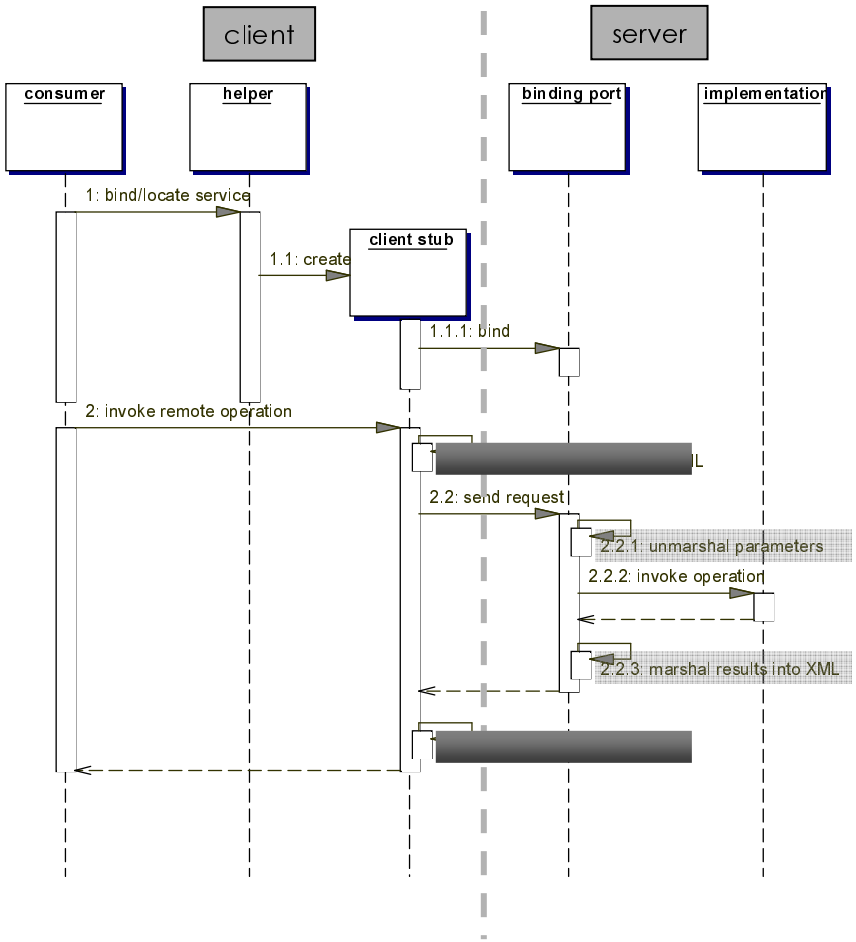
SSL encrypts the conversation between a single client and server, including authentication credentials

- however, there is no guarantee of non-repudiation without a client certificate
- more importantly, you lose confidentiality and non-repudiation in the presence of ANY intermediaries or multi-party transactions

Beneath the transport layer

- Since we cannot do much to secure the transport layer when it involves a single link in an arbitrary chain, what is left?
  - we have to secure the message itself
  - that requires us to take a look into SOAP and a few security standards for web services

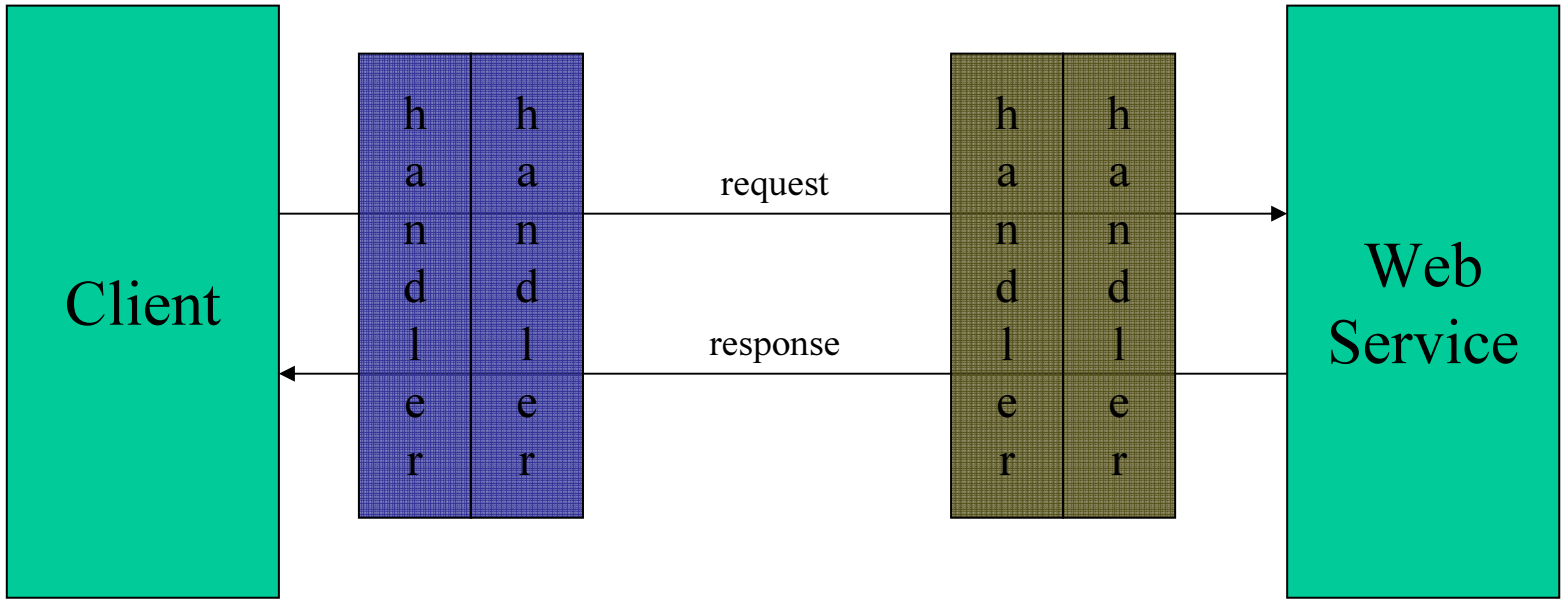
simple web service interaction



- the sequence to the left might be generated from JBuilder using Apache Axis
  - straight JAX-RPC and MS .NET code will differ
  - the ideas are somewhat consistent across implementations

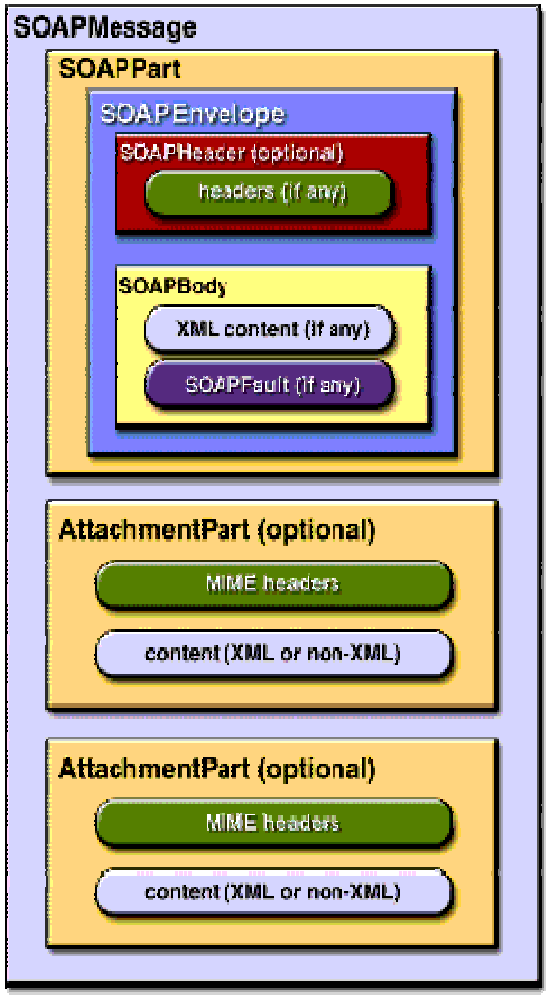
- to add security features to the XML communications, we can intercept the process of marshaling and unmarshaling the request and response

Intercepting the SOAP request and response



Anatomy of a SOAP message

- To add security to the content in the SOAP body, we will be altering the received message
  - for the receiver to get back to the original message, we must add processing instructions
  - those processing instructions are added to the SOAP header



A note on the structure of SOAP messages

Standard web service requests take one of two forms

- RPC, where the SOAP body is like a function call with parameters
- Document, where the "request" is a header, and the SOAP body is an XML document

In next slides will use the Document form

Relevant security specifications

- XML Signature
  - for signing all or part of an XML document
  - provides integrity and non-repudiation, regardless of intermediaries
- XML Encryption
  - for encrypting portions of an XML document
  - provides confidentiality, regardless of intermediaries

by adding these to the authentication capabilities of BASIC-AUTH and SSL, the security picture is more complete

- there are other ways to authenticate as well
- authorization is all that is left
  - often that requires additional effort on your part…
  - we will get back to this

Canonicalization

- Before looking deeper into XML Signature and XML Encryption, we must define XML-C14N (Canonical XML)
  - essentially, this allows two XML documents that have dissimilar whitespace to be compared
  - this is relevant because different XML processors may respond differently to whitespace
    - each whitespace character could alter a signature or resulting ciphertext

C14N processing steps, in case you don't trust it

1. the document is encoded in UTF-8 (RFC 3629)
2. line breaks are normalized to #xA before parsing
3. attribute values are normalized as if by XML validation rules
4. character and parsed entity references are replaced
5. CDATA sections are replaced with their character content
6. the XML declaration and DTD are removed
7. empty elements are replaced with start/end tag pairs
8. whitespace outside the document element and within start/end tag pairs is normalized
9. all whitespace in character content is retained
10. attribute value delimiters are set to quotation marks
11. special characters in attribute values and character content are replaced by character references
12. superfluous namespace declarations are removed
13. default attribute values are explicitly added to elements
14. lexicographic order is imposed on namespace declarations and attributes for each element

Fundamentals of signatures

A signature is a special form of *digest* computed on a relevant block of data

- a hash code is computed on the data block using a well-known algorithm
  - the sender computes the initial hash and adds it to the transmission
  - the receiver computes the hash on the data and checks that both hash codes match
  - this ensures the digest and the data block have integrity (they are unaltered from sender to receiver)
- to prevent hacking, the digest is hashed a second time and then encrypted
  - the hashed and encrypted digest is called a signature
  - private-key encryption provides non-repudiation

A sample of XML Signature

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header>
        <SOAP-SEC:Signature SOAP-ENV:actor="" SOAP-ENV:mustUnderstand="1" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                            xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/2000-12">
            <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
                <dsig:SignedInfo>
                    <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
                    <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
                    <dsig:Reference URI="#43871">
                        <dsig:Transforms>
                            <dsig:Transform Algorithm="http://www.w3.org/TR/2000/REC-xml-c14n-20001026" />
                        </dsig:Transforms>
                        <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                        <dsig:DigestValue>... Base64-encoded Digest Value ...</dsig:DigestValue>
                        <!-- the digest is computed on the referenced element -->
                    </dsig:Reference>
                </dsig:SignedInfo>

                <dsig:SignatureValue>... Base64-encoded Signature Value ...</dsig:SignatureValue>

                <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                    <KeyValue>
                        <RSAKeyValue>
                            <Modulus>... Base64-encoded Modulus ...</Modulus>
                            <Exponent>AQAB</Exponent>
                        </RSAKeyValue>
                    </KeyValue>
                    <X509Data>
                        <X509IssuerSerial>
                            <X509IssuerName>CN=Borcon 2004 Security Demo, OU=DevRel, O=Borland Software Corp, ST=CA,
C=US</X509IssuerName>
                            <X509SerialNumber>4045383e</X509SerialNumber>
                        </X509IssuerSerial>
                        <X509SubjectName>CN=Borcon 2004 Security Demo, OU=DevRel, O=Borland Software Corp, ST=CA, C=US</X509SubjectName>
                        <X509Certificate>... Base64-encoded Certificate ...</X509Certificate>
                    </X509Data>
                </KeyInfo>
            </dsig:Signature>
        </SOAP-SEC:Signature>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <request xmlns="http://localhost:7001/sec-web-service/BorconDemo" id="43871" submitted="2004-03-08">
            ... the rest of the SOAP body goes here
        </request>    </SOAP-ENV:Body> </SOAP-ENV:Envelope>
```

Implementations of XML Signature

The following vendor libraries abstract XML Signature processing in your SOAP handlers

- HP Web Services Platform 2.0
- IAIK XML Signature Library
- IBM XML Security Suite
- Infomosaic SecureXML Digital Signature
- Phaos XML
- RSA BSAFE Cert-J
- Verisign XML Signature SDK

The role of encryption

To this point we have discussed authentication, authorization, integrity and non-repudiation

- the role of encryption is to provide *confidentiality*
- it is the process of converting *plaintext* into *ciphertext*
- we will go into the mechanics more in the second part
- for now, consider that using XML Encryption, we can selectively encrypt any portion of the SOAP body

A sample document for encryption

```
<?xml version="1.0" encoding="UTF-8"?>
<purchase-order>
    <customer>
        <account-number>ABC-12345</account-number>
        <name>ABC Company</name>
        <line1>123 Main St.</line1>
        <city>Boston</city>
        <state>MA</state>
        <postal-code>02134</postal-code>
    </customer>
    <order-date>2004-03-08</order-date>
    <shipvia mode="USPS Standard"/>
    <items>
        <item quantity="4" sku="AB431"/>
        <item quantity="8" sku="AA781"/>
        <item quantity="1" sku="ZD550"/>
        <item quantity="15" sku="CA112"/>
    </items>
    <promotion>111-0110</promotion>
</purchase-order>
```

let's say we encrypt the account-number element

a sample of XML Encryption

```
 <EncryptedData Id="ed1" Type="http://www.w3.org/2001/04/xmlenc#Element"
xmlns="http://www.w3.org/2001/04/xmlenc#">
     <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
     <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
         <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
             <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
             <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                 <KeyName>Borcon</KeyName>
             </KeyInfo>
             <CipherData>
                 <CipherValue>Jpa0fhVTFwjMtP5dPdsoMRZo1yDuDmNCR5mro75IY42erCiPFgFIDtHeaphz+OO+J/mbMO2zeuGaEW
                              2I85pye/YlkKhS/fxosmGsOXH9Fl+wt1N9YNWju+rsERf9d0qpjn5bJaU4gAkGy7jVzJ+PaLLBL8Ka
                              ruVD9SddtFvhGCs=</CipherValue>
             </CipherData>
         </EncryptedKey>
     </KeyInfo>
     <CipherData>
         <CipherValue>aimNgaCFUlwKwiYzZz/Pb32sCcaHEzYoJRx1l13TlRtIX9jbaTq6b0RueknguO9czdi2zHsdE20=</CipherValue>
     </CipherData>
 </EncryptedData>
</purchase-order>
```

XML Encryption considerations

- You can encrypt multiple blocks using different keys
  - perhaps intended for use by different parties in the same request
  - encryption is retained through multiple hops
- could we forego SSL completely?
  - XML Encryption cannot secure the entire message, just blocks within the body

Securing web service entry points

- In addition to the security concerns addressed so far, there should be consideration for securing the entry points to web services
  - UDDI registries
  - ebXML registries
  - web application interfaces used for developing and testing
- in most cases we have seen to date, WSDL interfaces are published and directly accessible from unsecured points in the network

Securing UDDI/ebXML registries

- UDDI v3 provides additional support for digitally signing several request elements
  - businessEntity, businessService, bindingTemplate, tModel, publisherAssertion, etc…
  - this allows consumers who look up web services to be identified with non-repudiation
- moreover, authorization is implemented such that publishers can modify only the entries they created.

Additional tactics for securing registries

- Digitally-signed WSDL
- XML Encryption of private request/response elements (recall that registries are also web services)
- reducing authorization to the registry to very short timed intervals to reduce sniffing and replay attacks
- use SAML (described next) to make assertions about the authorization of a party

Security Assertions Markup Language (SAML)

- Provides queries and assertions in XML
  - authentication
  - authorization (decisions)
  - attributes of known security parties

- Open source implementations
  - www.sourceid.org
  - www.openSAML.org
  - etc.
- Commercial implementations
  - SunONE Identity Server
  - Netegrity JSAML Toolkit
  - Baltimore SelectAccess
  - Systinet WASP Card

sample SAML request (query)

```
<?xml version="1.0"?>
<samlp:Request xmlns:samlp="#"
               MajorVersion="1"
               MinorVersion="0"
               RequestID="123.45.678.90.12345678">

    <samlp:AuthenticationQuery>
        <saml:Subject xmlns:saml="#">
            <saml:NameIdentifier
SecurityDomain="pillartechnology.com"
                                 Name="kfaw" />
        </saml:Subject>
    </samlp:AuthenticationQuery>
</samlp:Request>
```

similar requests can make queries or assertions regarding authorization of a party to secured resources, or to query about attributes for a given party within the security domain

sample SAML response

```
<?xml version="1.0"?>
<samlp:Response xmlns:samlp="#" MajorVersion="1" MinorVersion="0"
                RequestID="128.14.234.20.90123456"
InResponseTo="123.45.678.90.12345678"
                StatusCode="Success">

    <saml:Assertion xmlns:saml="#" MajorVersion="1" MinorVersion="0"
                    AssertionID="123.45.678.90.12345678"
                    Issuer="Pillar Technology Group, LLC"
                    IssueInstant="2004-03-08T18:00:03Z">

        <saml:Conditions NotBefore="2004-03-08T18:00:10Z"
                         NotAfter="2004-03-08T18:00:40Z" />

        <saml:AuthenticationStatement AuthenticationMethod="Password"
                                      AuthenticationInstant="2004-03-
08T18:00:00Z">
            <saml:Subject>
                <saml:NameIdentifier SecurityDomain="pillartechnology.com"
Name="kfaw" />
            </saml:Subject>
        </saml:AuthenticationStatement>
    </saml:Assertion>
</samlp:Response>
```

WS-Security

- In light of all these lower-level technologies, what is WS-Security?
  - the simple answer… a specification defining how they apply to SOAP
  - submitted to OASIS (Organization for the Advancement of Structured Information Standards) in 2002 for development as a standard
- WS-Security defines headers for subject authentication, as well as specifications for signing and encrypting that info
- There are also many related specifications that are in various states of acceptance

Standards and standards bodies

- W3C
  - SOAP
  - XML Encryption
  - XML Signature
  - XKMS
    - X-KISS (info system)
    - X-KRSS (reg system)
- OASIS (Organization for the Advancement of Structured Information Standards)
  - ebXML
  - PKI
  - SAML
  - UDDI
  - XACML (access ctrl)

WS-I (IBM, Microsoft, BEA, Verisign)

- WS-Addressing
- WS-Authorization
- WS-Coordination
- WS-Federation
- WS-Inspection
- WS-Notification
- WS-Policy
- WS-Privacy
- WS-ReliableMessaging
- WS-Routing
- WS-SecureConversation
- WS-Security
- WS-Transaction
- WS-Trust


Part II

Roadmap

- Cryptography and configuration
- The public key infrastructure
- Implementing symmetric and asymmetric encryption in code
- Implementing digital signatures in code
- Applying XML Signature and XML Encryption in code
- Putting it all together in a web service invocation and response
- Some more technologies, SAML and XKMS.

Basic cryptography primer

Cryptography involves the mathematical algorithms for performing encryption and for computing hash code values

there are four primary algorithm types in common cryptography

- symmetric, a.k.a. shared secret
- block cipher
- hash function
- asymmetric, a.k.a. public key

symmetric algorithms

The same key, often called a *shared secret*, is used for encryption and decryption

common algorithms

- IDEA – generally considered very secure; 128-bit key; free for non-commercial use
- RC4 – very fast; accepts keys of arbitrary length
  - RC4-40 is the symmetric algorithm used in exportable SSL

block ciphers

- Convert a fixed-size block of data into another block of the same size
- therefore, they can be applied recursively to their own output, with the same or different keys
- common algorithms
  - Blowfish – one of the most secure
  - DES – common, but outdated; easily cracked by today's standards
  - RC5 – also common
  - 3DES – essentially DES applied three times; usually encrypt-decrypt-encrypt with 3 different keys; pretty slow

hash functions

- consistently compute a fixed-length string from an arbitrary block of data
- used to validate that the block it is computed on has not changed
- common algorithms
  - MD5 – in wide use; considered reasonably secure; 128-bit hash
  - SHA – US government algorithm; also considered pretty good; 160-bit hash
  - SHA1 – an extension of SHA, to fix an undisclosed attack point; 160-bit hash

asymmetric algorithms

- a public key is used for encryption, such that decryption is only possible with a private key
- any message sender can encrypt, knowing that only the receiver can decipher the contents
- common algorithms
  - RSA – most common; used for signing and encrypting; longer keys (>1k-bit) make it more secure
  - Diffie-Helman – often used for shared-secret key exchange
  - DSS – used by the US government for signatures; problems have been found with its use

Security providers (JCE)

- as JDBC providers offer drivers to support connecting to various databases…
  - security providers offer libraries of algorithms to support various forms of cryptography
  - although Java Cryptography Extension (JCE) security is integral to JDK 1.4, not all algorithms are provided
    - a notable exception is the RSA algorithm, provided for signing, but not encryption

enumerating JCE providers

```
public class ProviderInformation {
    public static void main(String[] args) {
        Provider[] providers = Security.getProviders();
        for (int i = 0; i < providers.length; i++) {
            Provider provider = providers[i];
            System.out.println("Provider name: " +
                               provider.getName());
            System.out.println(" information: " +
                               provider.getInfo());
            System.out.println(" version: " +
                               provider.getVersion());
            Set entries = provider.entrySet();
            Iterator iter = entries.iterator();
            while (iter.hasNext()) {
                System.out.println(" - " + iter.next());
            }
        }
    }
}
```

configuring additional providers

- append additional lines to the *java.security* file in jre/lib/security/
- add your security libraries to the Java classpath
  - typically place them in jre/lib/ext/
- to enable higher encryption, download an unrestricted version of local_policy.jar and US_export_policy.jar from the Sun web site
  - these are also stored in jre/lib/security/

.net security framework

- The security framework in .net is contained in the assemblies under System.Security
  - cryptography is under System.Security.Cryptography
  - algorithms supported out-of-the-box include
    - symmetric – DES, 3DES, RC2, Rijndael
    - asymmetric – RSA, DSA
    - hash – MD5, SHA1
  - in older versions of M$ Windows, support for strong encryption is in the High Encryption Pack

Simple code examples – symmetric encryption

- Remember that the same key is used for both encryption and decryption
  - this is fast…
  - however, it requires both sides to know the "shared secret"
    - how many secrets do you have to share if you need confidentiality with N other machines???
    - how easy do you think it will be to keep the shared secret with all those machines???

In practice – basic JCE symmetric encryption

```
public class SymmetricEncryption {
    public static void main(String[] args) {
        try {
            final String algorithmName = "Blowfish";
            KeyGenerator keyGen = KeyGenerator.getInstance(algorithmName);
            Key key = keyGen.generateKey();
            Cipher cipher = Cipher.getInstance(algorithmName);
            cipher.init(Cipher.ENCRYPT_MODE, key);
            System.out.println("Key and cipher generated by: " +
cipher.getProvider());
            System.out.println("Algorithm: " + cipher.getAlgorithm());

            byte[] plaintext = "Hello, everyone".getBytes();
            System.out.println("Original data: " + new String(plaintext));

            byte[] ciphertext = cipher.doFinal(plaintext);
            System.out.println("Encrypted data: " + new String(ciphertext));

            cipher.init(Cipher.DECRYPT_MODE, key);
            System.out.println("Decrypted data: " +
                               new String(cipher.doFinal(ciphertext)));
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Simple code examples – asymmetric encryption

- in next slide, plaintext is encrypted with the receiver's public key
  - only the receiver can decrypt, because the private key is private
  - anyone can send messages to the receiver confidentially using the same public key
  - key management is much easier in scalable environments
  - algorithms are generally considered more secure, because the private key never leaves the receiver's possession
  - however, this method of encryption is also slower than symmetric encryption

il practice – basic .net asymmetric encryption

```
using System;
using System.Security.Cryptography;
using System.Text;

namespace Encryption {
    class AsymmetricDemo {
        [STAThread]
        static void Main(string[] args) {
            // simply creating the RSA provider generates a key pair by default
            // alternatively, we could then load a key into the created object
            Console.WriteLine("Generating RSA key and cipher");
            RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();

            // convert plaintext to a byte array using UTF-8
            UTF8Encoding utf8 = new UTF8Encoding();
            byte[] plaintext = utf8.GetBytes("Hello, everyone");

            byte[] ciphertext = rsa.Encrypt(plaintext, true);

            Console.WriteLine("Original data: " + utf8.GetString(plaintext));
            Console.WriteLine("Encrypted data: " + utf8.GetString(ciphertext));

            byte[] decryptedText = rsa.Decrypt(ciphertext, true);
            Console.WriteLine("Decrypted data: " + utf8.GetString(decryptedText));

            Console.Write("Press <ENTER> to finish...");
            Console.Read();
        }
    }
}
```

Applying cryptography to SOAP messages

The SOAP specification does not provide provisions for securing messages

- confidentiality of all or part of the transmission
- authenticity through use of a digital signature
- these are provided by XML Encryption and XML Signature, respectively

XML Signature

- XML Signature (xml-dsig) allows us to add signing information to an XML document
  - compute a digest on a block of XML data
  - add the digest to a SOAP header element
  - optionally include our certificate, to aid the receiver in validating the digest
- in Java, XML Signature must be hand-coded, or use a utility library like wss4j
- the .net framework includes xml-dsig classes in System.Security.Cryptography.XML

In practice – performing XML Signature in .net

```
XmlDocument doc = new XmlDocument();
doc.PreserveWhitespace = false;
doc.Load(new XmlTextReader("initial-response.xml"));
XmlNodeList list = doc.GetElementsByTagName("env:Body");
XmlElement body = (XmlElement)list[0];
Console.WriteLine(body.FirstChild.OuterXml);

// Create a reference to the data to be canonicalized and signed.
Reference reference = new Reference();
reference.AddTransform(new XmlDsigC14NTransform());
reference.Uri = "";

// Create a SignedXml object and add the reference.
RSACryptoServiceProvider key = new RSACryptoServiceProvider();
SignedXml signedXml = new SignedXml((XmlElement)body.FirstChild);
signedXml.SigningKey = key;
signedXml.AddReference(reference);

// Add the key so the receiver can validate our signature.
KeyInfo keyInfo = new KeyInfo();
keyInfo.AddClause(new RSAKeyValue((RSA)key));
signedXml.KeyInfo = keyInfo;

// Compute the signature and store it in the SOAP header.
signedXml.ComputeSignature();
XmlElement xmlDigitalSignature = signedXml.GetXml();
doc.DocumentElement.FirstChild.AppendChild(doc.ImportNode(xmlDigitalSignature, true));

// Save the signed XML document to a file so we can prove we did it.
XmlTextWriter xmltw = new XmlTextWriter("initial-response-signed.xml", new
UTF8Encoding(false));
xmltw.Formatting = Formatting.Indented;
doc.WriteTo(xmltw);
xmltw.Close();
```

Results of XML Signature

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env=http://schemas.xmlsoap.org/soap/envelope/
              xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
              xmlns:soapenc=http://schemas.xmlsoap.org/soap/encoding/
              xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <env:Header>
        <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
            <SignedInfo>
                <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
                <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
                <Reference URI="">
                    <Transforms>
                        <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
                    </Transforms>
                    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                    <DigestValue>RjmqTulOiJr+Iu/GDC7CNUEAw9A=</DigestValue>
                </Reference>
            </SignedInfo>
            <SignatureValue>oYtzba8fXmi5TTeqmR2XQVkhtNZrflDNHoDCDJv1JtZDPi1iQcWFvQxQXDVGDRImIgA+JhVNVSpP0wDUAdyKKBr+0SCn
ETkgO7kgxhCeWTZSr
                            hxJwAFMdW818HJaIAe14GPXDuUN7nPWszzmHxGWqcfGzsHlgPec8D+jvstqCkg=</SignatureValue>
            <KeyInfo>
                <KeyValue xmlns="http://www.w3.org/2000/09/xmldsig#">
                    <RSAKeyValue>
                        <Modulus>u0zEjEw9hPw5NmLTT+AkX7DDtn0UJtXnE7S1c2ZN6I/PEnGdbPm/Z72rksGrG3QNoZy7rZlfgPiHfGywjdmpTZN
7ixp5j4MGgBcf/3NJ
                            oBRLsgVihe0x3dYLMlpoWW8pA4DczPU/SybQb4onSba2ub3aR9raefj5bwNJ5+7ajOU=</Modulus>
                        <Exponent>AQAB</Exponent>
                    </RSAKeyValue>
                </KeyValue>
            </KeyInfo>
        </Signature>
    </env:Header>
    <env:Body env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <m:listImagesResponse xmlns:m="http://attachments">
            <result soapenc:arrayType="xsd:string[3]">
                <xsd:string xsi:type="xsd:string">attitude-simple-honest-direct.JPG</xsd:string>
                <xsd:string xsi:type="xsd:string">Cary_Not_Carrry.JPG</xsd:string>
                <xsd:string xsi:type="xsd:string">christmas-stars.jpg</xsd:string>
            </result>
        </m:listImagesResponse>
    </env:Body>
</env:Envelope>
```

XML Encryption

- XML Encryption defines a protocol for encrypting portions of a SOAP transmission, including
  - canonicalization
  - identifying the node to encrypt, perhaps with XPATH
  - producing an encrypted version of the node
  - substituting the encrypted node for the plaintext node
- You could perform all the XML manipulation yourself
  - in .net, this is the only alternative open to you
  - however, in Java there are toolkits to do both encryption and signatures, e.g.,
    - open source – Apache WSS4J (a subproject of WS-FX). See next slide.
    - commercial – IBM WSDK, etc. No slide about this?

Apache WSS4J

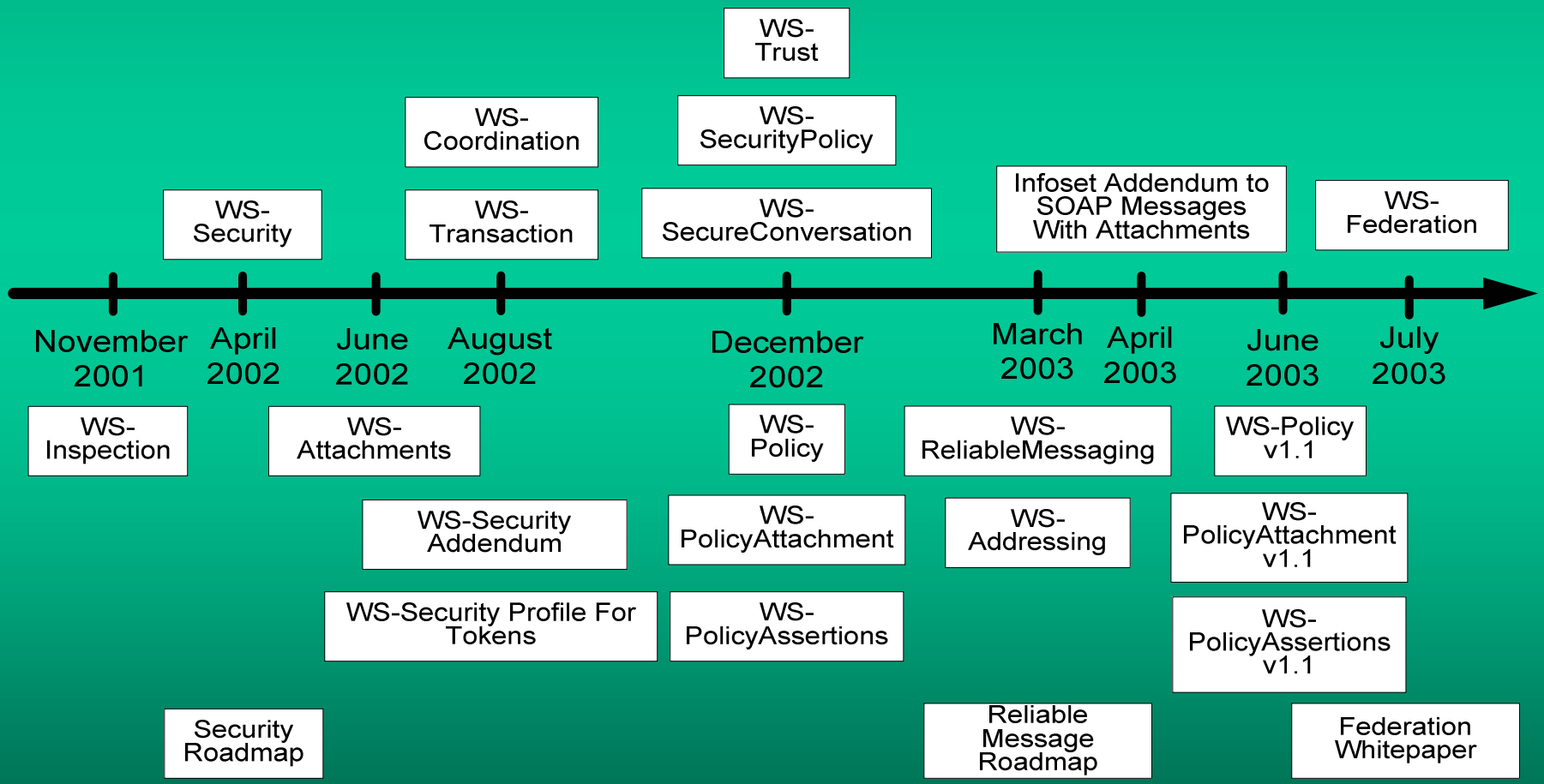
- Apache WSS4J is an implementation of the OASIS Web Services Security (WS-Security) from OASIS Web Services Security TC.
- WSS4J is a primarily a Java library that can be used to sign and verify SOAP Messages with WS-Security information. WSS4J will use Apache Axis and Apache XML-Security projects and will be interoperable with JAX-RPC based server/clients and .NET server/clients.

Additional WS-Security support

- WSS4J also supports
  - SAML, shipping with openSAML out of the box
  - XML Encryption and Signature are implemented as JAX-RPC handlers
    - they plug into web service application configurations declaratively
- other toolkits and commercial implementations may also provide additional support.

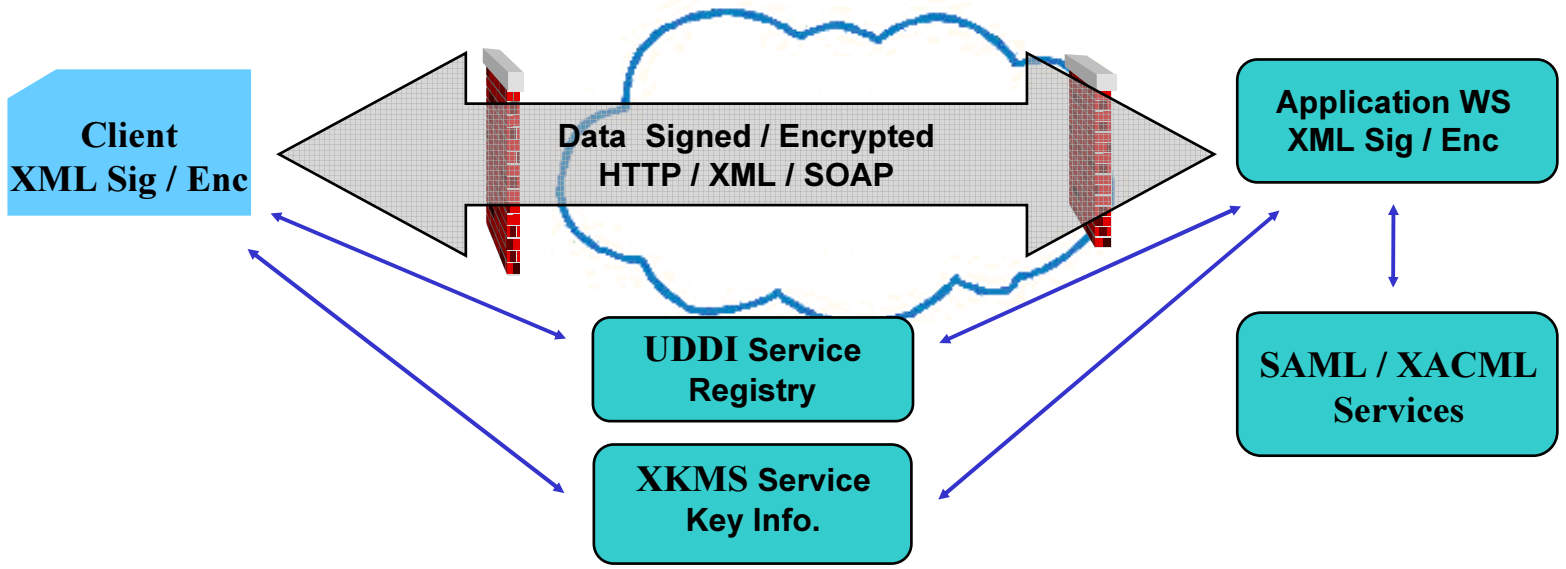
Second Wave Specifications

Historical Timeline of Specifications



XML/PKI Security - Web Services

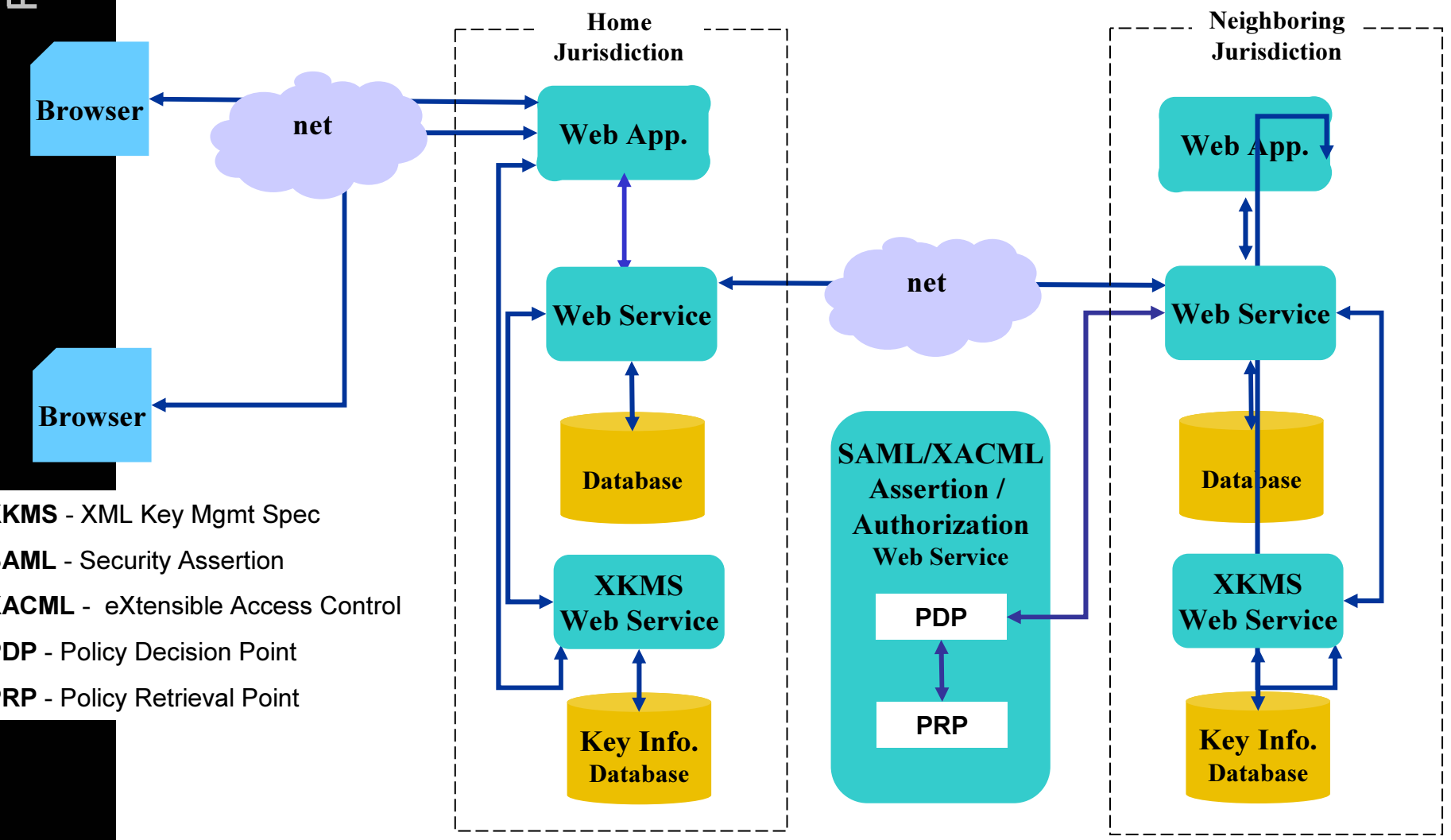
Trust and Interoperability



XML/PKI Security
PKI (Public Key Infrastructure)
XML Signatures / XML Encryption
XKMS (XML Key Management Specification)
SAML (Security Assertion Markup Language)
XACML (eXtensible Access Control Markup Language)

Web Services
UDDI (Universal Description and Discovery)
WSDL (Web Services Description Language)
SOAP (Simple Object Access Protocol)
XML (eXtensible Markup Language)
HTTP (Hyper Text Transfer Protocol)

Scenarios - Conceptual Model



XKMS - XML Key Mgmt Spec

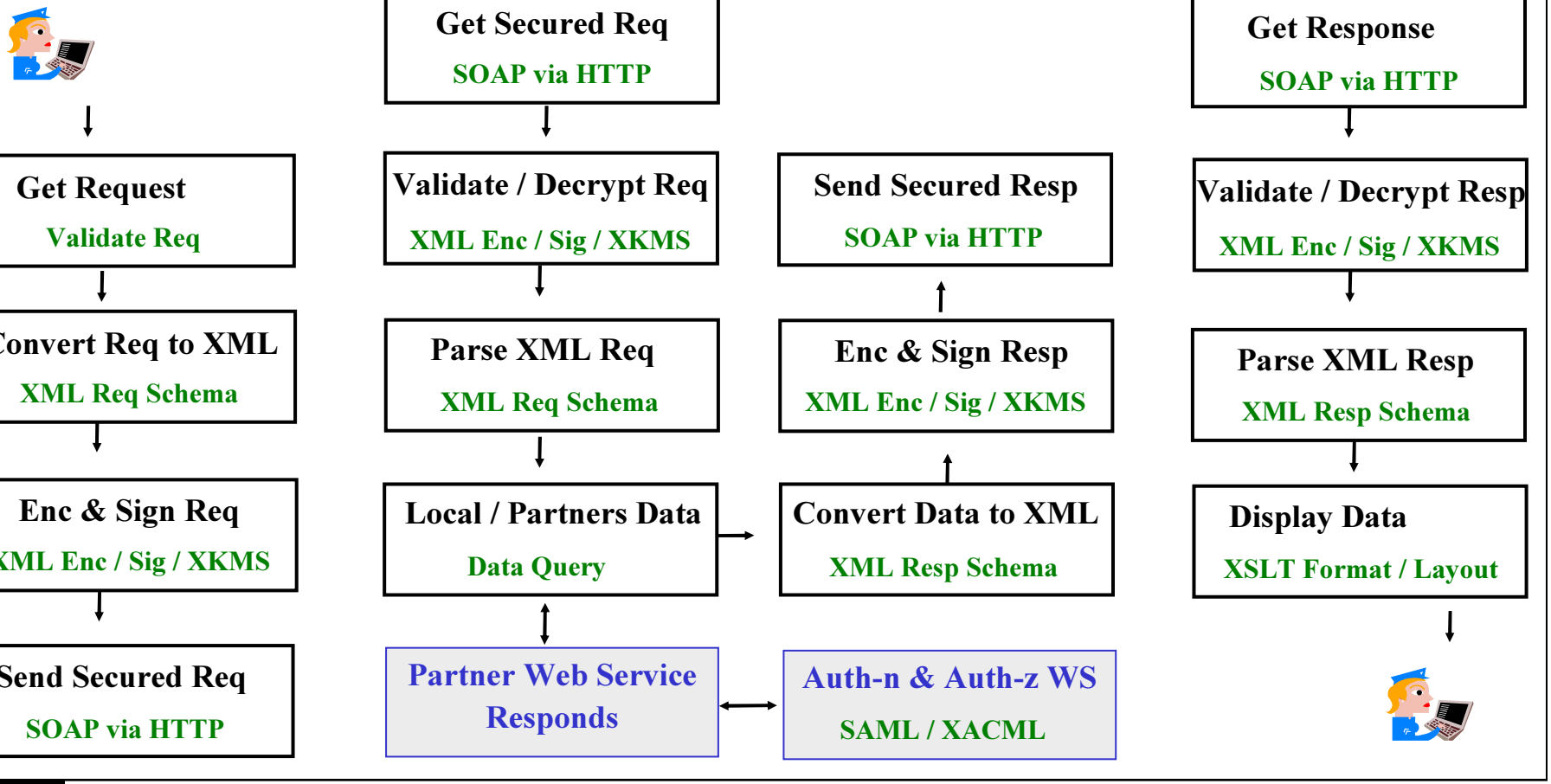
SAML - Security Assertion

XACML - eXtensible Access Control

PDP - Policy Decision Point

PRP - Policy Retrieval Point

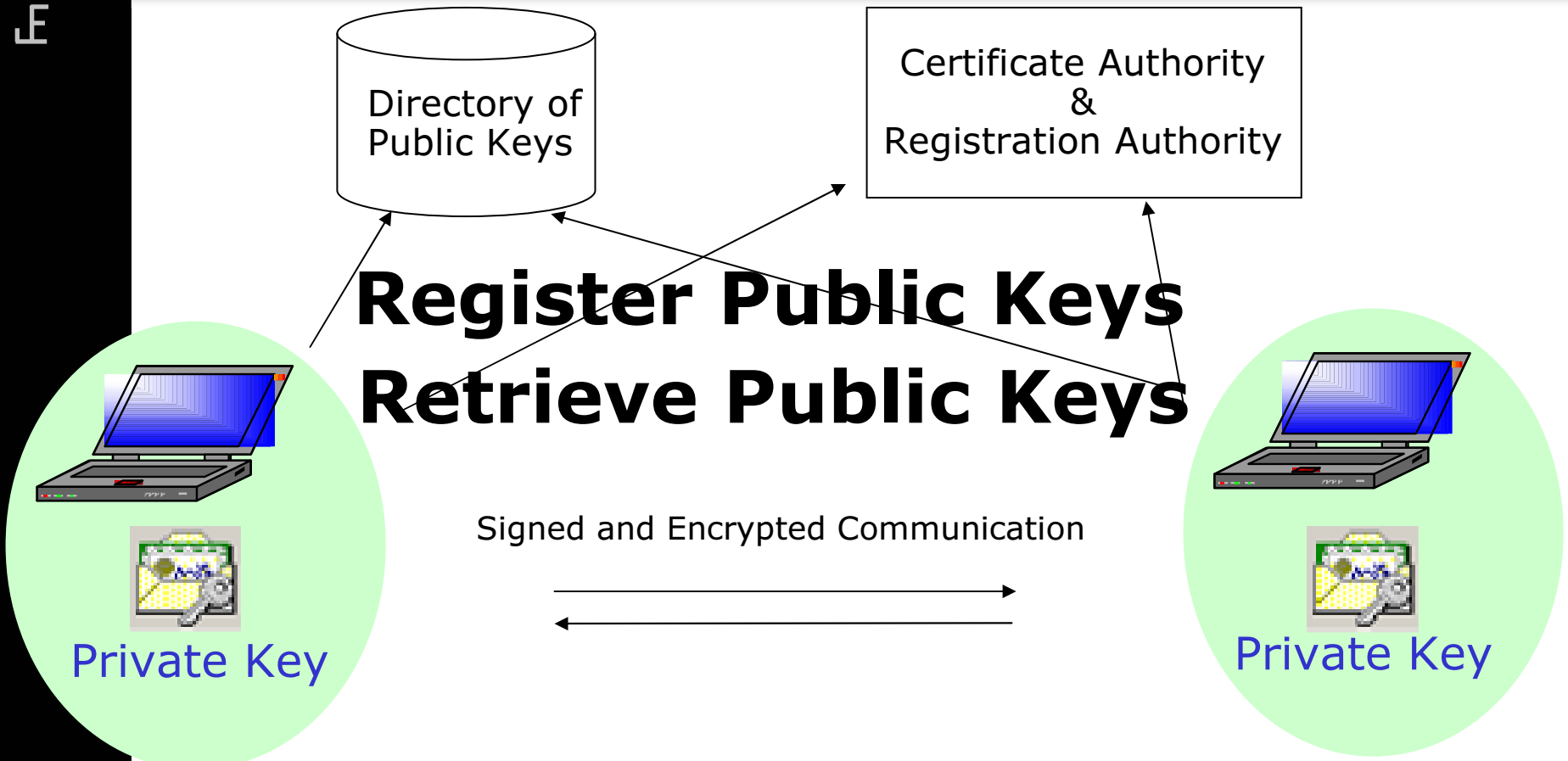
Request – Response: XML/PKI Security Process



Quick introduction to PKI

- PKI stands for “Public Key Infrastructure”
  - Public and Private key cryptography is used for:
    - Digital Signatures
    - Encryption
  - A *PKI* is used for
    - Verifying Digital Signatures
    - Verifying the identity of a signatory
    - Registering a user’s identity with a Trusted Third Party (TTP)
    - Maintaining an online directory of *Digital Certificates*
  - Remember – the “I” in PKI stands for “Infrastructure”
  - We see these PKI features on the next slide…

Four Steps of PKI



Step 1 – Register Public Keys with PKI

Step 2 – Retrieve Public Key in Digital Certificate from PKI Directory

Step 3 – Encrypt document with recipient's public key

Step 4 – Sign document with sender's private key

Step 1 – Register Public Keys with PKI

- XKMS includes X-KRSS (XML Key Registration Service Specification)
- XKMS services can be used to register public/private key pairs
  - For escrow services
  - For revocation
  - For recovery
- Keys can be generated on the client, providing that:
  - A cryptographic engine is present on the client
  - The client is capable of performing computationally-expensive cryptography

  Otherwise, the XKMS service can generate the keys that are subsequently managed through the service.
- With legacy non-XML key request protocols, such as PKCS#10, a client-side toolkit was required to register keys. This could cause “toolkit bloat”
- http://www.w3.org/TR/xkms/

Step 2 – Retrieve Public Key in Digital Certificate from PKI Directory

DSML (Directory Services Markup Language) - OASIS

DSML 1.0 provided a means of representing directory information in XML.

- Allows a directory to describe its schema in XML - a language another directory or an application can understand

The OASIS Directory Services TC is working on DSML 2.0, which adds:

- Support for querying directories
- Support for modifying directories

DSML 2.0 will:

- Allow lightweight devices which don't include an LDAP client (eg PDAs and smart phones) to query a directory
- Bypass firewalls which block LDAP traffic at present

http://www.oasis-open.org/committees/dsml/index.shtml

Step 3 – Encrypt document with recipient's public key

XML Encryption – W3C

Defines:

- A process for encrypting/decrypting digital content (including XML documents and portions thereof)
- An XML syntax used to represent
  - encrypted content
  - information that enables an intended recipient to decrypt it

Allows for *element-wise* encryption – meaning that an document can be encrypted per-element – see example on the next slide

http://www.w3.org/Encryption/2001/

XML encryption – element-wise encryption

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 – element-wise encryption

Scenario 1: Encrypt the entire CreditCard XML block:

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
    <Name>John Smith<Name/>
    <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
    xmlns='http://www.w3.org/2001/04/xmlenc#'>
      <CipherData>
                <CipherValue>A23B45C56</CipherValue>
      </CipherData>
    </EncryptedData>
</PaymentInfo>
```

XML encryption – element-wise encryption

Scenario 2: Encrypt only card's number, issuer, and expiration date

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

XML encryption – element-wise encryption

Scenario 3: 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>
```

Step 4 – Sign document with private key

XML Digital Signature

Joint IETF (Internet Engineering Task Force) and W3C (World-Wide Web Consortium initiative)

Used for signing “Any digital content” – not just XML

XML Digital Signature includes the following component parts:

- **Encrypted Hash of a document**
- **Information on algorithms used**
- **Information on PKI directory (optional)**
- **Public Key Certificate (optional)**

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 XML)

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

XML Digital Signature Example

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  - <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    - <Reference URI="http://www.w3.org/TR/xml-stylesheet">
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <DigestValue>60NvZvtdTB+7UnlLp/H24p7h4bs=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>juS5RhJ884qoFR8flVXd/rbrSDVGn40CapgB7qeQiT+rr0NekEQ6BHhUA8dT3+BC
      TBUQI0dBjlml9lwzENXvS83zRECjzXbMRTUtVZiPZG2pqKPnL2YU3A9645UCjTXU
      +jgFumv7k78hieAGDzNci+PQ9KRmm//icT7JaYztgt4=</SignatureValue>
  - <KeyInfo>
    - <KeyValue>
      - <RSAKeyValue>
          <Modulus>uCiukpgOaOmrq1fPUTH3CAXxuFmPjsmS4jnTKxrv0w1JKcXtJ2M3akaV1d/karvJ
            lmeao20jNy9r+/vKwibjM77F+3bIkeMEGmAIUnFciJkR+ihO7b4cTuYnEi8xHtu4
            iMn6GODBoEzqFQYdd8p4vrZBsvs44nTrS8qyyhba648=</Modulus>
          <Exponent>AQAB</Exponent>
        </RSAKeyValue>
      </KeyValue>
    - <X509Data>
        <X509SubjectName>CN=John Doe,O=Acme\, Inc.,ST=New York,C=US</X509SubjectName>
      - <X509IssuerSerial>
          <X509IssuerName>CN=Corporate CA,O=Acme\, Inc.,ST=New York,C=US</X509IssuerName>
          <X509SerialNumber>970849928</X509SerialNumber>
        </X509IssuerSerial>
        <X509Certificate>MIICeDCCAeGgAwIBAgIEOd3+iDANBgkqhkiG9w0BAQQFADBbMQswCQYDVQQGEwJJ
          RTEPMA0GA1UECBMGRHVibGluMSUwIwYDVQQKExxCYWx0aW1vcmUgVGVjaG5vbG9n
          aWVzLCBMdGQuMRQwEgYDVQQDEwtUZXN0IFJTQSBDQTAeFw0wMDEwMDYxNjMyMDda
          Fw0wMTEwMDYxNjMyMDRaMF0xCzAJBgNVBAYTAklFMQ8wDQYDVQQIEwZEdWJsaW4x
          JTAjBgNVBAoTHEJhbHRpbW9yZSBUZWNobm9sb2dpZXMsIEx0ZC4xFjAUBgNVBAMT
          DU1lcmxpbiBIdWdoZXMwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBALgorpKY
          Dmjpq6tXz1Ex9wgF8bhZj47JkuI50ysa79MNSSnF7SdjN2pGldXf5Gq7yZZnmqNt
          Izcva/v7ysIm4zO+xft2yJHjBBpgCFJxXIiZEfooTu2+HE7mJxIvMR7buIjJ+hjg
          waBM6hUGHXfKeL62QbL7OOJ060vKssoW2uuPAgMBAAGjRzBFMB4GA1UdEQQXMBWB
          E21lcmxpbkBiYWx0aW1vcmUuaWUwDgYDVR0PAQH/BAQDAgeAMBMGA1UdIwQMMAqA
          CEngrZIVgu03MA0GCSqGSIb3DQEBBAUAA4GBAHJu4JVq/WnXK2oqqfLWqes5vHOt
          fX/ZhCjFyDMhzslI8am62gZedwZ9IIZIwlNRMvEDQB2zds/eEBnIAQPl/yRLCLOf
          ZnbA8PXrbFP5igs3qQWScBUjZVjik748HU2sUVZOa90c0mJl2vJs/RwyLW7/uCAf C/I/k9xGr7fneoIW</X509Certificate>
      </X509Data>
    </KeyInfo>
</Signature>
```

Signing Web Resources

- XML Digital Signature be used to sign web resources referenced by URIs
- Signature is invalid if the resource
  - has changed, or
  - is unavailable

```
<?xml version="1.0"?>
<Signature Id="SimpleSignature" xmlns=http://www.w3.org/2000/09/xmldsig#>
<SignedInfo>
<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-
20001026"/>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
<Reference URI="http://www.vordel.com/index.html">
<Transforms>
<Transform Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>t54TgeGErewtWEtwetKYUQWDw</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>DSFrefjk7wdfWER</SignatureValue>
</Signature>
```

Signed URI

Signing XML

Remember – XML Signature is not just for signing XML, but for signing *any digital content*

But there are specific issues when we are digitally signing XML

- The XML itself is typically not seen by the user, the user sees the results after a style-sheet has processed the XML
- The XML rendered as HTML may depend on fonts or inline images
- The user must sign what they see, which may be very different from the underlying XML
- Where the signing entity is a computer, other considerations apply – who initiated the signing process, who is bound to the signing key

XML Digital Signature and SOAP

February 9, 2001 - IBM and Microsoft submit specification for SOAP Security Extensions (SOAP-SEC) to W3C

Proposes a standard way to use XML Digital Signature to sign SOAP 1.1 messages

Defines SOAP header entry <SOAP-SEC:Signature> for this purpose

Imports two optional headers for use in SOAP-SEC

- “actor” to indicate the recipient of a header element
- “mustUnderstand” to indicate whether an application must attempt the validation of the enclosed XML Digital Signature

OASIS – SAML and XACML

SAML - Security Assertion Markup Language

- Allows companies to securely exchange
  - Authentication information
  - Authorization information
  - profile information
- between their customers, partners, or suppliers regardless of the security systems or e-commerce platforms that they have in place today

XACML - XML Access Control Markup Language

- Defines
  - An XML specification for expressing authorization and entitlement policies for information access over the Internet
- For fine-grained access control

http://www.oasis-open.org/committees/security/index.shtml

XACML – Data Protection

eXtensible Access Control Markup Language

Conceived to support the separation of policy from applications, data bases, and operating systems.

Usage oriented doesn't drive policy regarding collection or disclosure.

Facilitates

- more flexible systems
- policy enforcement in heterogeneous, distributed environments
- granular control
- Introspection
- attach to anything that can be referenced from XML
- controls data and procedure access
- more approachable policy management

XACML – Data Protection

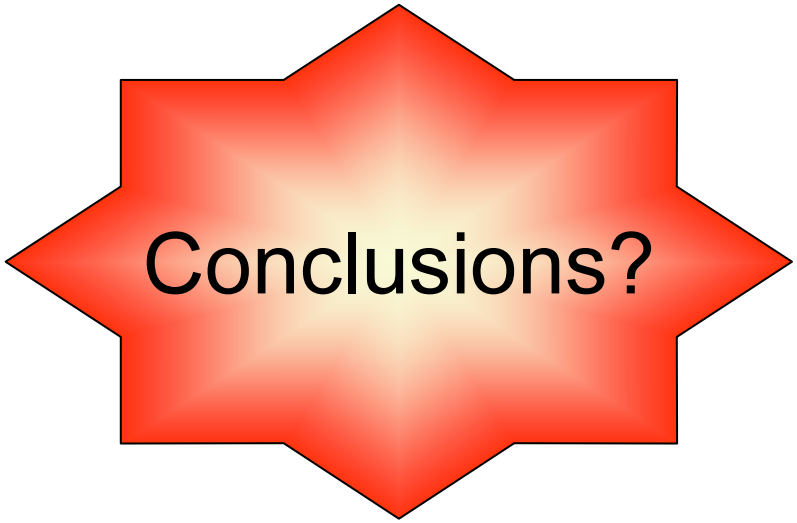
Participants include, but not limited to:

- Sun
- IBM
- Verisign
- Hewlett Packard
- Netegrity
- Cisco
- University Of Milan

Apache WS Project

- http://ws.apache.org/

  Project List:
  - Addressing
  - Axis
  - EWS
  - JaxMe
  - jUDDI
  - Kandula
  - Mirae
  - Muse
  - Pubscribe
  - Sandesha
  - Scout
  - SOAP
  - TSIK
  - Woden
  - WSIF
  - WSRF
  - WSS4J
  - XML-RPC




That's all!
Thanks!!!!