SUN TECH DAYS 2005-2006
A Worldwide Developer Conference

Innovation Happens Here.

Securing Your Enterprise: Web Application and Web Services Security

Rima Patel Sriganesh
Staff Engineer
Sun Microsystems, Inc.

Security is panoptic...

- Applicable at all levels of a solution
  - Hardware
  - OS
  - Network
  - Application / Web server
    - Standalone application OR Web tier
    - Application server tier
      - Component security such as EJB security or Web service security
  - Data
    - Database
    - Other sources of data such as directory services

This is just an example of layers of security for a typical three tier application!

What does “security” imply in the world of computing?

- Security implies ensuring that the following is achieved
  - Confidentiality (of data en route network)
  - Authentication (of subjects)
  - Integrity (of data en route network)
  - Non-repudiation (of origin and destination)
  - Authorization (of subjects)
  - Trust (of subjects)
  - Privacy (of subjects)

Subject = Users (at origin and destination), Code (at origin and destination), any entity that need security assurance

All that can be ensured using various mechanisms...

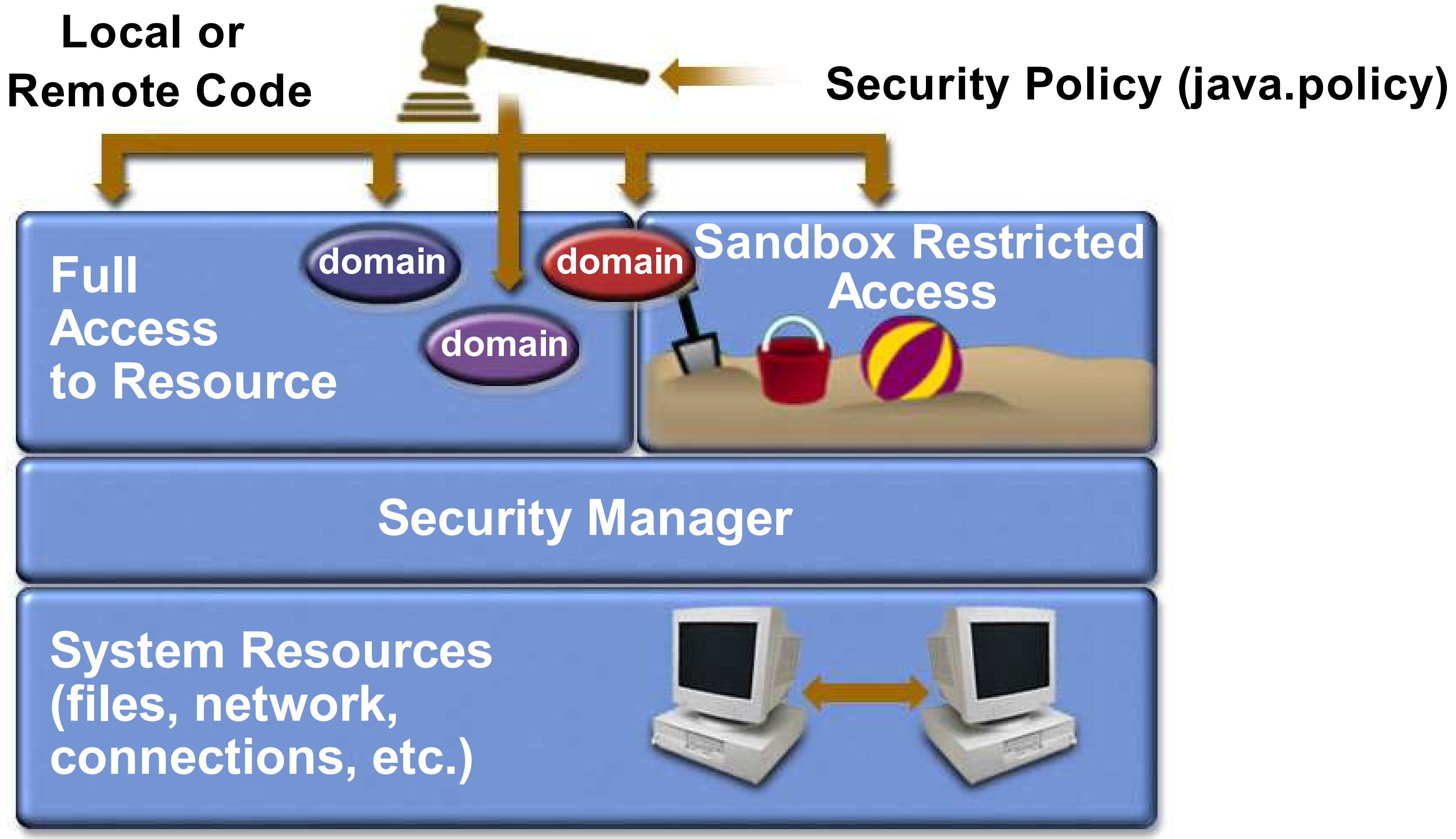
- Encryption / decryption
- Digital signing
- Identity and federation based security mechanisms

What is in our scope today?

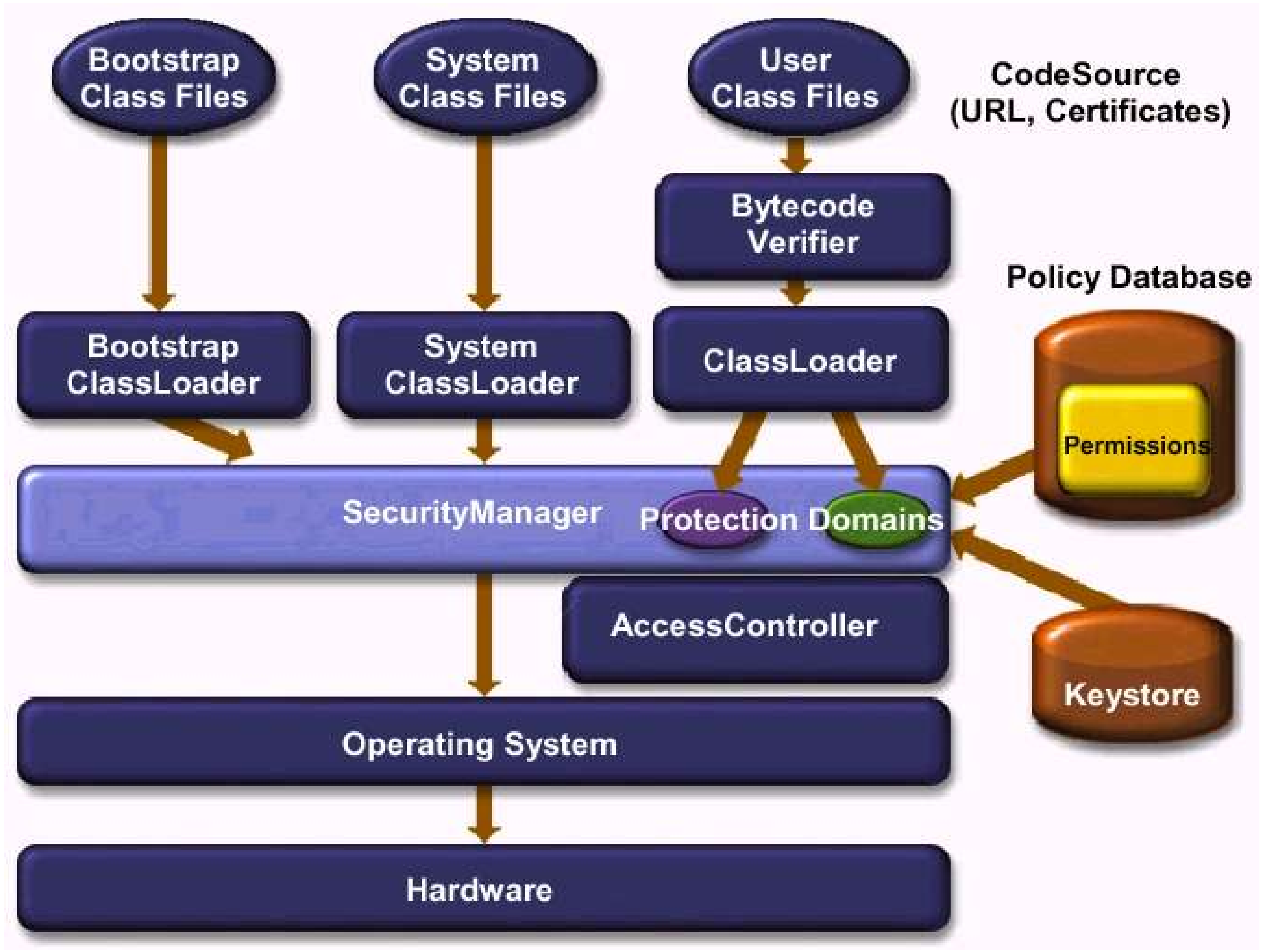
- Web application security
- Web service security
- SSO and Federations
  - SAML 2.0
  - Liberty
  - Sun Java System Identity Management Platform

Java Security 101

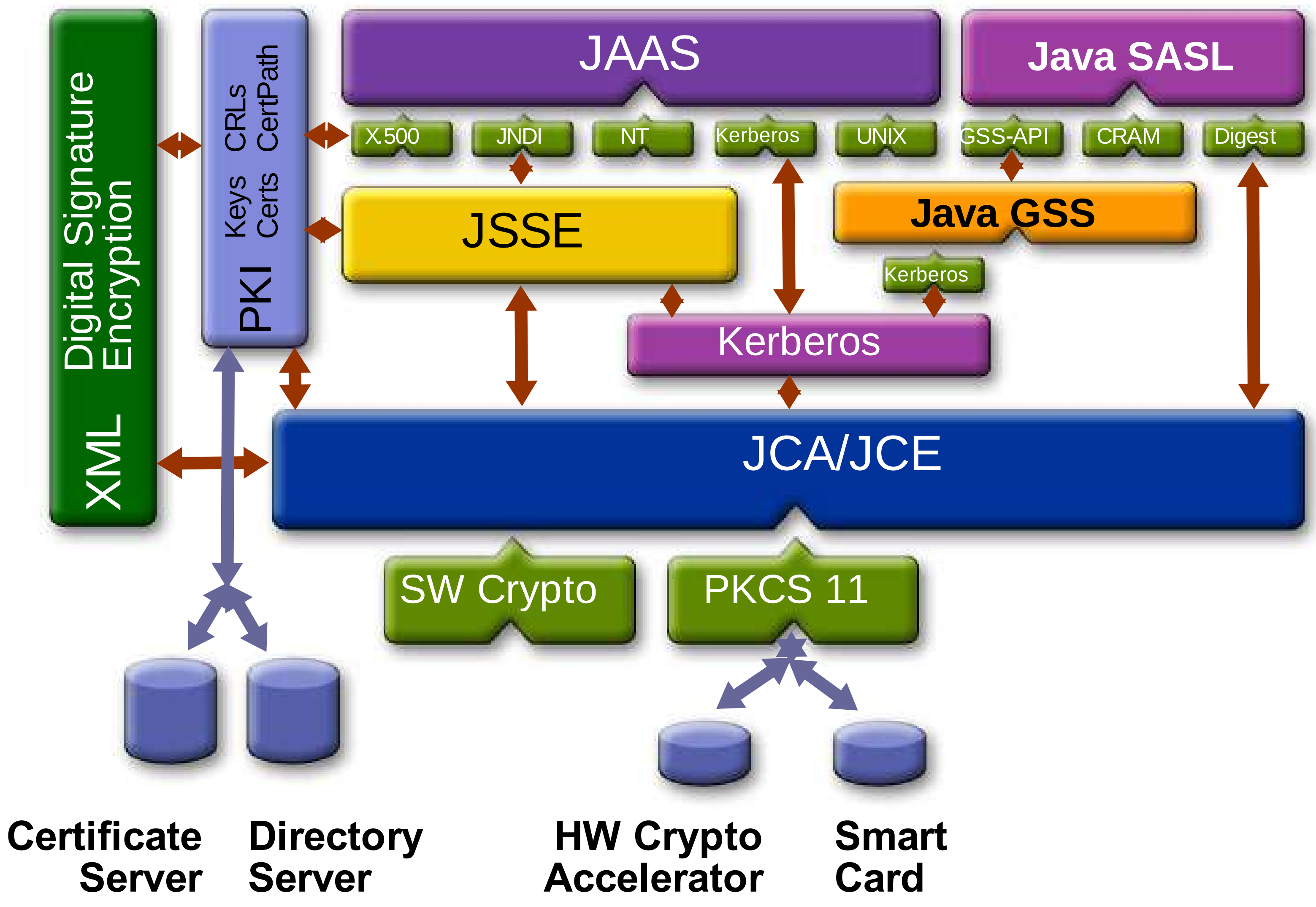
J2SE Platform Security Model



JRE Runtime Security

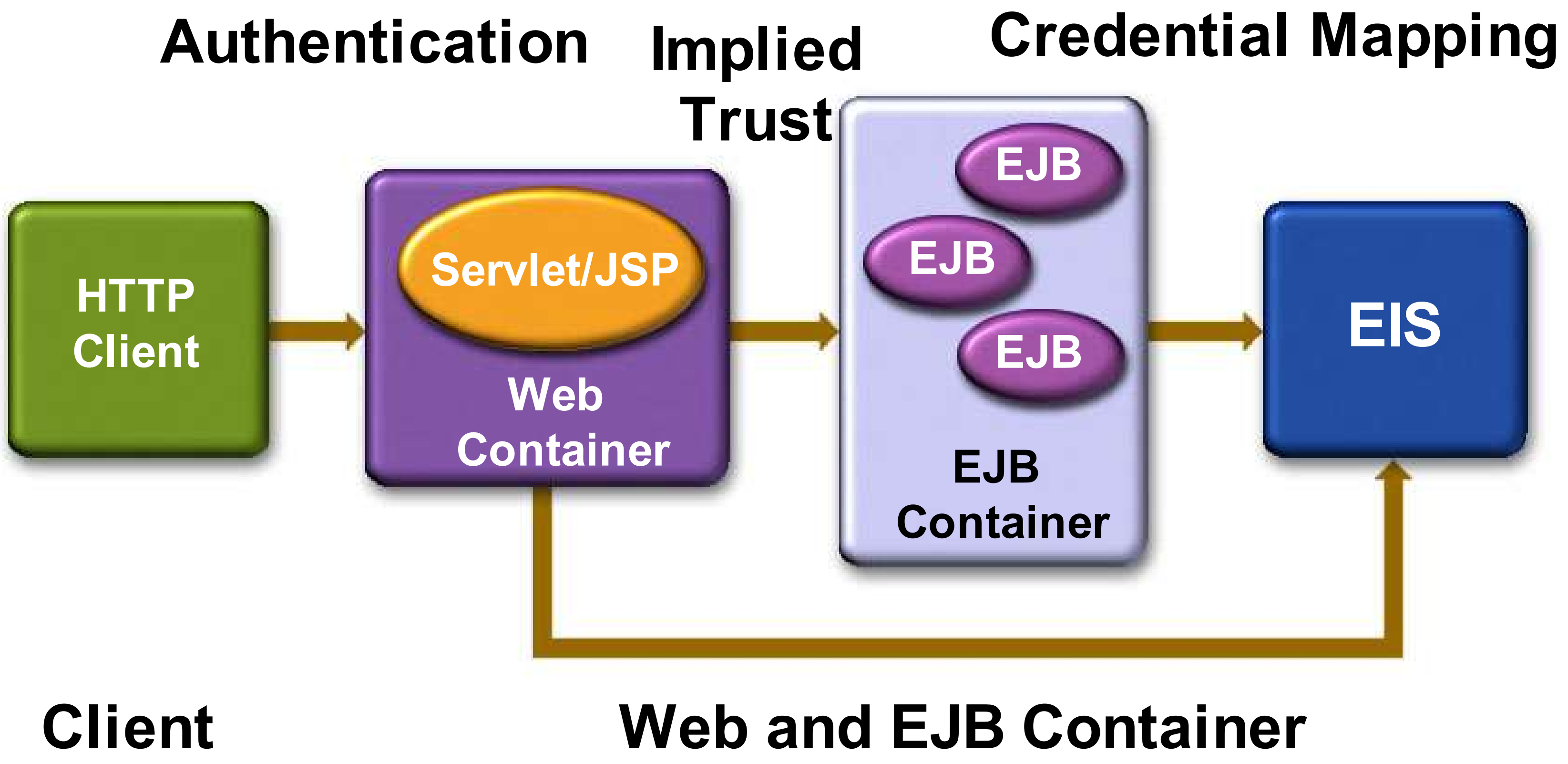


Java Platform Security – APIs



Web Application Security

J2EE Web Application Security Model



The Open Web Application Security Project

- www.owasp.org
  - Non-profit organization funded by a foundation
- Provides minimum standard for Web application security
- Lots of documentation on best practices of security
- Many tools to secure J2EE applications
- We have a lot to learn from them!

OWASP Top 10 Web Security Threats

1. Unvalidated input
2. Broken access control
3. Broken authentication
4. Cross-site scripting (XSS)
5. Buffer overflows
6. Injection flaws
7. Improper error handling
8. Insecure storage
9. Application denial-of-service
10. Insecure configuration management

Source: www.owasp.org

Let's see what we can learn about the security vulnerabilities of a typical Web application!

#1: Unvalidated Input (Description)

- Attacker can easily tamper any part of the HTTP request before submitting
  - URL
  - Cookies
  - Form fields
  - Hidden fields
  - Headers
- Common names for common input tampering attacks
  - forced browsing, command insertion, cross site scripting, buffer overflows, format string attacks, SQL injection, cookie poisoning, and hidden field manipulation

#1: Unvalidated Input (Solutions)

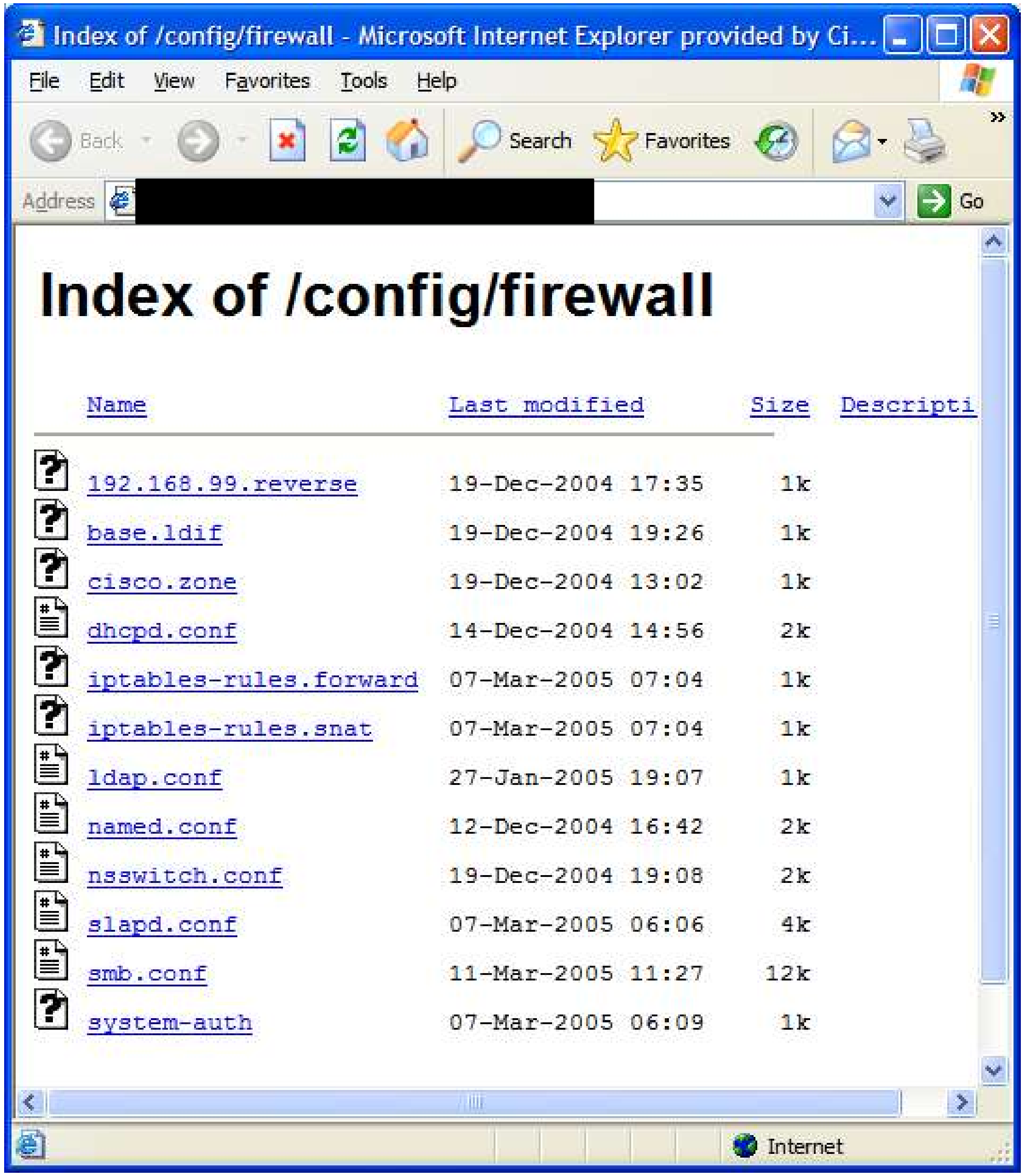
- Do rigorous input data validation
  - > All parameters should be validated before use
- Do server-side validation
  - > Client side validation could be bypassed by the attacker easily
- Do canonicalization of input data
  - > The process of simplifying the encoding
- Stinger Project
  - > Defines validation rules for every part of HTTP request handled by a J2EE Web application

#1: Unvalidated Input (Example)

```
public void doPost(HttpServletRequest req,…) {
  String customerId =
    req.getParameter(“customerId”);
  String sku = req.getParameter(“sku”);
  String stringPrice = req.getParameter(“price”);
  Integer price = Integer.valueOf(stringPrice);
  // Store in the database
  orderManager.submitOrder(sku,customerId,price);
} // end doPost
```

#2: Broken Access Control (Examples)

- Insecure ID's
- Forced browsing past access control checking
- Path traversal
- File permissions
- Client side caching


Index of /config/firewall - Microsoft Internet Explorer provided by Ci...
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
Go
Internet

[redacted]

Index of /config/firewall

```
     Name                        Last modified      Size  Descripti
 ---------------------------------------------------------------
     192.168.99.reverse          19-Dec-2004 17:35    1k
     base.ldif                   19-Dec-2004 19:26    1k
     cisco.zone                  19-Dec-2004 13:02    1k
     dhcpd.conf                  14-Dec-2004 14:56    2k
     iptables-rules.forward      07-Mar-2005 07:04    1k
     iptables-rules.snat         07-Mar-2005 07:04    1k
     ldap.conf                   27-Jan-2005 19:07    1k
     named.conf                  12-Dec-2004 16:42    2k
     nsswitch.conf               19-Dec-2004 19:08    2k
     slapd.conf                  07-Mar-2005 06:06    4k
     smb.conf                    11-Mar-2005 11:27   12k
     system-auth                 07-Mar-2005 06:09    1k
```

#3: Broken Authentication & Session Management

- Includes all aspects of handling user authentication and managing active sessions
- Session hi-jacking
  - If the session tokens are not properly protected, an attacker can hijack an active session and assume the identity of a user

#3: Broken Account/Session Management (Client Example—SSO)

```
public void doGet(HttpServletRequest req,…) {
   // Get user name
   String userId = req.getRemoteUser();
   Cookie ssoCookie = new Cookie("userid",userId);
   ssoCookie.setPath("/");
   ssoCookie.setDomain("cisco.com");
   response.addCookie(ssoCookie);
   …
}
```

#3: Broken Account/Session Management (Server Example—SSO)

```
public void doGet(HttpServletRequest req,…) {
  // Get user name
  Cookie[] cookies = req.Cookies();
  for (i=0; i < cookies.length; i++) {
    Cookie cookie = cookies[i];
    if (cookie.getName().equals(“ssoCookie”)) {
        String userId = cookie.getValue();
        HttpSession session = req.getSession();
        session.setAttribute(“userId”,userId);
    } // end if
  } // end for
} // end doGet
```

#3: Broken Account/Session Management (Client Solution—SSO)

```
public void doGet(HttpServletRequest req,…) {
  // Get user name
  String userId = req.getRemoteUser();
  encryptedUserId = Encrypter.encrypt(userId);
  Cookie ssoCookie =
    new Cookie("userid",encrypteduserId);
  ssoCookie.setPath("/");
  ssoCookie.setDomain("cisco.com");
  response.addCookie(ssoCookie);
  …
}
```

#3: Broken Account/Session Management (Server Solution—SSO)

```
public void doGet(HttpServletRequest req,…) {
  // Get user name
  Cookie[] cookies = req.Cookies();
  for (i=0; i < cookies.length; i++) {
    Cookie cookie = cookies[i];
    if (cookie.getName().equals(“ssoCookie”)) {
          String encryptedUserId = cookie.getValue();
          String userId = Encrypter.decrypt(encryptedUserId);
          if (isValid(userId)) {
                 HttpSession session = req.getSession();
                 session.setAttribute(“userId”,userId);
          } // end if isValid…
    } // end if cookie = ssoCookie…
  } // end for
} // end doGet
```

#4: Cross Site Scripting (Description)

- An attacker can use cross site scripting to send malicious script to an unsuspecting user
- The end user's browser has no way to know that the script should not be trusted, and will execute the script
  - > Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by your browser and used with that site
  - > These scripts can even rewrite the content of the HTML page

#4: Cross Site Scripting (Description)

- XSS attacks usually come in the form of embedded JavaScript
  - However, any embedded active content is a potential source of danger, including: ActiveX (OLE), VBscript, Shockwave, Flash and more

#4: Cross Site Scripting (Examples)

- Disclosure of the user’s session cookie – session high-jacking
- Disclosure of end user files
- Installation of Trojan horse programs
- Redirecting the user to some other page or site
- Modifying presentation of content

#4: Cross Site Scripting (Counter Measures)

- Validate input against a rigorous positive specification of what is expected
  - Validation of all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification of what should be allowed
  - ‘Negative’ or attack signature based policies are difficult to maintain and are likely to be incomplete
  - White-listing: a-z, A-Z, 0-9, etc.
  - Truncate input fields to reasonable length

#4: Cross-Site Scripting (Flaw)

```
protected void doPost(HttpServletRequest req, HttpServletResponse res) {
    String title = req.getParameter("TITLE");
    String message = req.getParameter("MESSAGE");
  try {
      connection = DatabaseUtilities.makeConnection(s);
      PreparedStatement statement =
        connection.prepareStatement
          ("INSERT INTO messages VALUES(?,?)");
      statement.setString(1,title);
      statement.setString(2,message);
      statement.executeUpdate();
    } catch (Exception e) {
      …
    } // end catch
} // end doPost
```

#4: Cross-Site Scripting (Solution)

```
private static String stripEvilChars(String evilInput) {
   Pattern evilChars = Pattern.compile(“[^a-zA-Z0-9]”);
   return evilChars.matcher(evilInput).replaceAll(“”);
}
protected void doPost(HttpServletRequest req, HttpServletResponse res) {
  String title = stripEvilChars(req.getParameter(“TITLE”));
  String message = stripEvilChars(req.getParameter(“MESSAGE”));
  try {
     connection = DatabaseUtilities.makeConnection(s);
     PreparedStatement statement =
       connection.prepareStatement
        (“INSERT INTO messages VALUES(?,?)”);
     statement.setString(1,title);
     statement.setString(2,message);
     statement.executeUpdate();
   } catch (Exception e) {
     …
   } // end catch
} // end doPost
```

#6: Injection Flaws (Description)

- Injection flaws allow attackers to relay malicious code through a web application to another system
  - > Calls to the operating system via system calls
  - > The use of external programs via shell commands
  - > Calls to backend databases via SQL (i.e., SQL injection)
- Any time a web application uses an interpreter of any type there is a danger of an injection attack

#6: Injection Flaws (Description)

- Many web applications use operating system features and external programs to perform their functions
  - Runtime.exec() to external programs (like sendmail)
- When a web application passes information from an HTTP request through as part of an external request, the attacker can inject special (meta) characters, malicious commands, or command modifiers into the information

#6: Injection Flaws (Example)

- SQL injection is a particularly widespread and dangerous form of injection
  - To exploit a SQL injection flaw, the attacker must find a parameter that the web application passes through to a database
  - By carefully embedding malicious SQL commands into the content of the parameter, the attacker can trick the web application into forwarding a malicious query to the database

#6: Injection Flaws (Examples)

- Path traversal
  - > "../" characters as part of a filename request
- Additional commands could be tacked on to the end of a parameter that is passed to a shell script to execute an additional shell command
  - > "; rm –r *"
- SQL queries could be modified by adding additional 'constraints' to a where clause
  - > "OR 1=1"

#6: Injection Flaws (How to find them)

- Search the source code for all calls to external resources
  - e.g., system, exec, fork, Runtime.exec, SQL queries, or whatever the syntax is for making requests to interpreters in your environment

#6: Injection Flaws (Counter Measures)

- Avoid accessing external interpreters wherever possible
  - Use library API's instead
- Structure many requests in a manner that ensures that all supplied parameters are treated as data, rather than potentially executable content
  - For SQL, use PreparedStatement or Stored procedures
- Ensure that the web application runs with only the privileges it absolutely needs to perform its function

#6: SQL Injection (Counter Measures)

- When making calls to backend databases, carefully validate the data provided to ensure that it does not contain any malicious content
- Use PreparedStatement or Stored procedures

#9: Application DOS (Description)

- Types of resources
  - bandwidth, database connections, disk storage, CPU, memory, threads, or application specific resources
- Application level resources
  - Heavy object allocation/reclamation
  - Overuse of logging
  - Unhandled exceptions
  - Unresolved dependencies on other systems
    - Web services
    - Databases

#9: Application DOS (How to determine you vulnerability)

- Load testing tools, such as JMeter can generate web traffic so that you can test certain aspects of how your site performs under heavy load
  - > Certainly one important test is how many requests per second your application can field
  - > Testing from a single IP address is useful as it will give you an idea of how many requests an attacker will have to generate in order to damage your site
- To determine if any resources can be used to create a denial of service, you should analyze each one to see if there is a way to exhaust it

#9: Application DOS (Counter Measures)

- Limit the resources allocated to any user to a bare minimum
- For authenticated users
  - Establish quotas so that you can limit the amount of load a particular user can put on your system
  - Consider only handling one request per user at a time by synchronizing on the user’s session
  - Consider dropping any requests that you are currently processing for a user when another request from that user arrives

#9: Application DOS (Counter Measures)

- For un-authenticated users
  - Avoid any unnecessary access to databases or other expensive resources
  - Caching the content received by un-authenticated users instead of generating it or accessing databases to retrieve it
- Check your error handling scheme to ensure that an error cannot affect the overall operation of the application

Security Principles

1. Minimize attack surface area
2. Secure defaults
3. Principle of least privilege
4. Principle of defense in depth
5. Fail securely
6. External systems are insecure
7. Separation of duties
8. Do not trust security through obscurity
9. Simplicity
10. Fix security issues correctly

Minimize Attack Surface Area

- The aim for secure development is to reduce the overall risk by reducing the attack surface area
- Every feature that is added to an application adds a certain amount of risk to the overall application

Secure Defaults

- There are many ways to deliver an “out of the box” experience for users. However, by default, the experience should be secure, and it should be up to the user to reduce their security – if they are allowed
- Example:
  - By default, password aging and complexity should be enabled
  - Users might be allowed to turn these two features off to simplify their use of the application and increase their risk.

Principle of Least Privilege

- Accounts have the least amount of privilege required to perform their business processes.
  - > This encompasses user rights, resource permissions such as CPU limits, memory, network, and file system permissions
- Example
  - > If a middleware server only requires access to the network, read access to a database table, and the ability to write to a log, this describes all the permissions that should be granted

Principle of Defense In Depth

- Controls, when used in depth, can make severe vulnerabilities extraordinarily difficult to exploit and thus unlikely to occur.
  - > With secure coding, this may take the form of tier-based validation, centralized auditing controls, and requiring users to be logged on all pages

Fail Safely

- Applications regularly fail to process transactions for many reasons. How they fail can determine if an application is secure or not
- Example: In the code below, if codeWhichMayFail() fails, the attacker gets an admin priviledge

```
isAdmin = true;
try {
     codeWhichMayFail();
     isAdmin = isUserInRole( “Administrator” );
}
catch (Exception ex) {
     log.write(ex.toString());
}
```

External Systems Are Insecure

- Implicit trust of externally run systems is not warranted
  - > All external systems should be treated in a similar fashion
- Example:
  - > A loyalty program provider provides data that is used by Internet Banking, providing the number of reward points and a small list of potential redemption items
  - > However, the data should be checked to ensure that it is safe to display to end users, and that the reward points are a positive number, and not improbably large

Separation of Duties

- A key fraud control is separation of duties
- Certain roles have different levels of trust than normal users
  - > In particular, Administrators are different to normal users. In general, administrators should not be users of the application
- Example
  - > An administrator should be able to turn the system on or off, set password policy but shouldn’t be able to log on to the storefront as a super privileged user, such as being able to “buy” goods on behalf of other users.

Do Not Trust Security Through Obscurity

- Security through obscurity is a weak security control, and nearly always fails when it is the only control
  - This is not to say that keeping secrets is a bad idea, it simply means that the security of key systems should not be reliant upon keeping details hidden
- Example
  - The security of an application should not rely upon only on knowledge of the source code being kept secret
  - The security of an application should rely upon many other factors, including reasonable password policies, defense in depth, business transaction limits, solid network architecture, and fraud and audit controls

Tools

- WebScarab - a web application vulnerability assessment suite including proxy tools
- Validation Filters – (Stinger for J2EE, filters for PHP) generic security boundary filters that developers can use in their own applications
- CodeSpy – look for security issues using reflection in J2EE apps

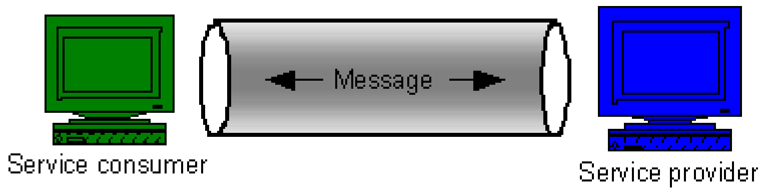
Tools

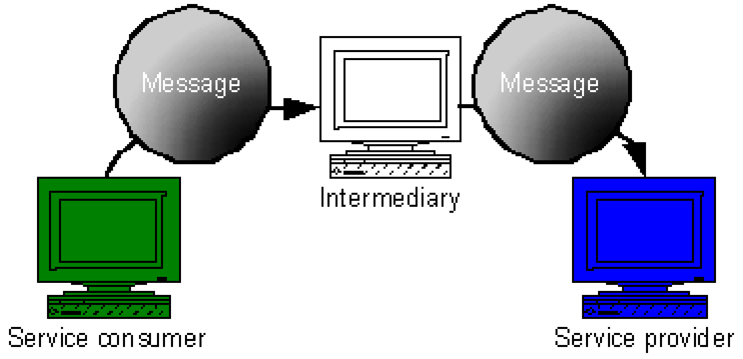
- CodeSeeker - an commercial quality application level firewall and Intrusion Detection System that runs on Windows and Linux and supports IIS, Apache and iPlanet web servers,
- WebGoat - an interactive training and benchmarking tool that users can learn about web application security in a safe and legal environment
- WebSphinx – web crawler looking for security issues in web applications
- OWASP Portal - our own Java based portal code designed with security as a prime concern

Web Service Security

Approaches to Web Services Security

- Most messaging still uses transport-layer security between services and consumers
  - SSL/TLS with mutual authentication
  - VPNs with IPsec
- Some use of message-layer security technologies
  - SOAP-aware
  - Non-SOAP-aware





Transport Security: Pros and Cons

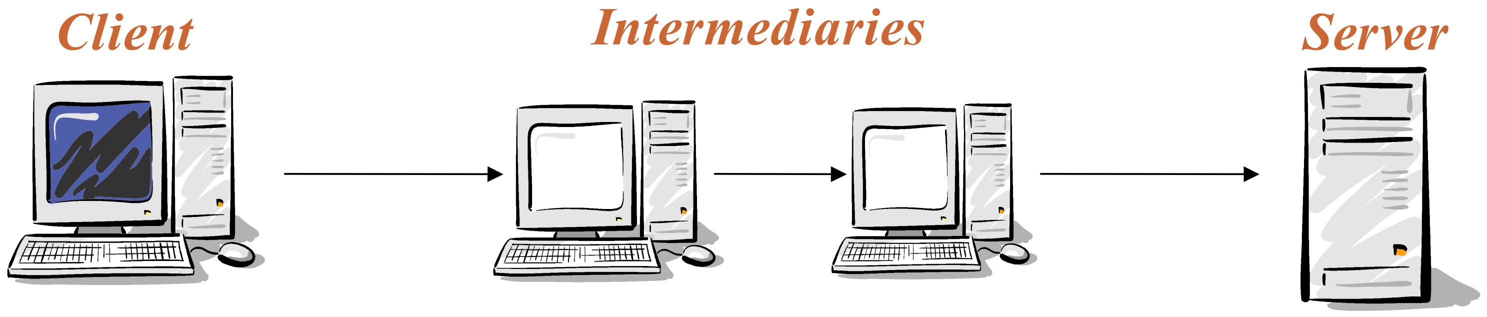
- ✓ Well-understood, standard technology
- ✓ Built-in way to do mutual authentication
- ✓ Few moving parts
- ✓ Captures message attachments as well as the main message body

- ✗ Unaware of the “XML inside”
  - - No selective encryption, auditing, or routing
  - - Can't sign the data
- ✗ Protection is transient
  - - No protection at OS layer or across system tiers
- ✗ Not end-to-end; can be used for only one hop
  - - The consumer and provider might never talk directly

Message Security: Pros and Cons

- ✓ Secures the message itself
- ✓ 'Sticks' to the message when it is stored or transmitted
  - ✓ Can be validated on multiple occasions
  - ✓ Over a wide span of time
  - ✓ Remains secure even after initial access
- ✓ Fine grained
  - ✓ Different pieces of the message can be operated on separately
- ✓ Intermediaries aren't a problem
- ✓ Makes possible selective encryption, auditing, routing, and signing
- ✓ Scales better to arbitrary partners
- ✗ **More moving parts**
- ✗ **More complex key management**

Transport Security via TLS/SSL



- SSL is good only for POINT-TO-POINT communication.
- Authentication becomes difficult on SSL
- SSL is fasten to HTTP protocol whereas WSS can bind to HTTP, SMTP, FTP and JMS transports as the technology matures.
- Encrypting/Signing partly in the Message not possible with SSL
- Security only when data is on the wire, does not secure data off the wire so doesn't support non-repudiation.

Web Service Security Technologies

Standard	Venue	Status
XML Signature	W3C, IETF	Recommendation
XML Encryption	W3C	Recommendation
Web Service Security	OASIS	Standard
X.509	ITU, IETF	Well established authentication technology using public/private keys

Web Services Security Technologies
XML Encryption

- W3C started XML Encryption activity in late January 2001
- Defines the process of encrypting and decrypting XML content
- Java Specification Request (JSR) 106

  http://www.jcp.org/jsr/detail/106.jsp

Web Services Security Technologies

XML Encryption (Contd.)

- XML vocabulary and process for encrypting and decrypting a whole or partial XML document
- Also uses XML for representing the information that enables recipients of the encrypted content to decrypt it
- XML Encryption is not a replacement to SSL/TLS
- Allows encryption of data at different granularities

```
<InventoryData>
 <Date>…</Date>
 <Products>
  <Product Amt="3">
  Chia Head</Product>
  <Product …>…</Product>
  …
 </Products>
</InventoryData>
```

```
<InventoryData>
 <Date>…</Date>
 <EncryptedData …>
  <CipherData>
   <CipherValue>
   …SKFWw5F34=…</CipherValue>
  </CipherData>
 </EncryptedData>
</InventoryData>
```

Web Services Security Technologies

XML Encryption on the Wire

```
<SOAP-ENV:Envelope   SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
                            xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
     <SOAP-ENV:Header>
        <Security SOAP-ENV:mustUnderstand='1'
xmlns='http://schemas.xmlsoap.org/ws/2003/06/secext'>
           <ReferenceList xmlns='http://www.w3.org/2001/04/xmlenc#'>
                        <DataReference URI='#ed1'/>
            </ReferenceList>
         </Security>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <PaymentInfo xmlns='http://example.org/paymentv2'>
                <Name>John Smith</Name>
                <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/200
                                 <KeyName>John Smith</KeyName>
                                </KeyInfo>
                                <CipherData>
                                <CipherValue>ydUNqHkMrD...</CipherValue
                                </CipherData>
                </EncryptedData>
         </PaymentInfo>
     </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
<SOAP-ENV:Envelope
 SOAP-ENV:enc='http://schemas.xmlsoap.org/soap/enc...'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/.../'>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <CreditCard Limit='5,000' Currency='USD'>
        <Number>4019 2445 0277 5567</Number>
        <Issuer>Example Bank</Issuer>
        <Expiration>04/02</Expiration>
      </CreditCard>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Web Services Security Technologies
XML Signature

- Authentication, data integrity (tamper-proofing), non-repudiation
- Joint W3C/IETF effort
  - XML syntax for representing signature of web resources and portions thereof
  - Procedures for computing and verifying such signatures
  - Canonicalization of XML data
- JSR-105

Web Services Security Technologies
XML Signature

- XML vocabulary and process for digitally signing whole or partial XML documents (or non-XML content) and validating signatures
- Provides a mechanism for applying digital signatures to XML documents
- A signature can be enveloped in the signed XML content, can envelope that content, or can be detached
- Its `<KeyInfo>` element identifies the relevant key; it's reused (profiled) in many other specs
- Provide strong integrity for
  - Message authentication
  - Signer authentication
  - Non-repudiation
- http://www.w3.org/Signature/

Web Services Security Technologies

XML Signature on the Wire

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <SOAP-SEC:Signature xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/2000-12"
              SOAP-ENV:mustUnderstand="1">
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
       <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <ds:Reference URI="#Body">
          <ds:Transforms>
                   <ds:Transform Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
          </ds:Transforms>
         <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:DigestValue>
        </ds:Reference>
       </ds:SignedInfo>
       <ds:SignatureValue>MC0CFFrVLtRlk=...</ds:SignatureValue>
       <ds:KeyInfo>
        <ds:KeyName>Michael</ds:KeyName>
       </ds:KeyInfo>
      </ds:Signature>
     </SOAP-SEC:Signature>
  </SOAP-ENV:Header>
   <SOAP-ENV:Body xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/2000-12" SOAP-
SEC:id="Body">
     <order:buy xmlns:order="http://www.onlinetrade.com/order">
      <order:ticker-symbol>SUNW</order:ticker-symbol>
      <order:quantity>1000</order:quantity>
      <order:market>New York</order:market>
     </order:buy>
  </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

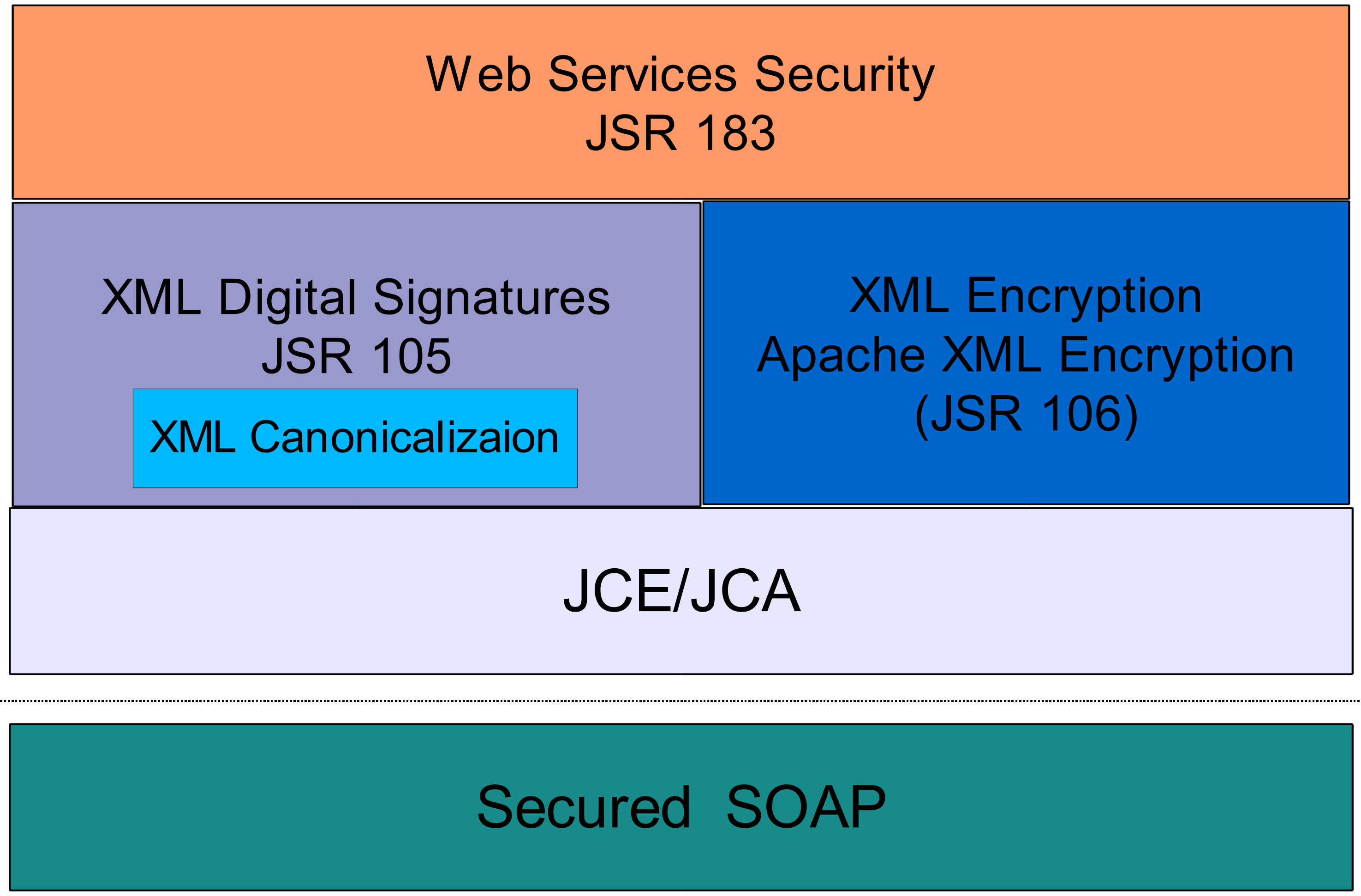
Web Services Security Technologies
Web Services Security

- OASIS standard that profiles XML Encryption and XML Signatures, mainly, to protect SOAP communication
- Defines token profiles to exchange security tokens across Web services
- Most widely adopted standard for security Web services

Web Services Security Technologies
WS-I Basic Security Profile

- WS-I focuses on interoperability at various layers of web services
- Profiled
  - SOAP, WSDL, UDDI
  - SOAP w/ Attachments
  - Security

Web Service Security in the Java Land



Web Services Security Support in Java WSDP 1.6

- Implements OASIS WSS 1.0
  - Encryption & Signatures
  - Authentication - X509, Username Token Profile
- Ease of Use
  - Declarative: Security configuration files
  - Configurable at Service, Port and Operation
- Interoperable: 100% with all vendors

Simple HelloWorld Example on JWSDP

- Client signs the SOAP request message body and Server verifies signature
- Server signs the SOAP response message body and the client verifies signature
- No encryption, uses plain HTTP transport
- Security properties
  - Mutual Client/Server authentication using X.509 PKI

Steps

- Programmatic security
  - Start with unsecured JAX-RPC server code
  - Add JAX-RPC ServiceLifecycle.init() method
    - Container is required to call init() if endpoint implements ServiceLifecycle interface
  - Method init() creates a ServerHelper to
    - Bind ServerHelper to endpoint
    - Configure security policy for endpoint
- Alternatively
  - Use Wscompile tool with -security option to generate security code
    - > Need to supply a XWS-Security configuration file to wscompile tool
    - > Security enabled JAXRPC artifacts (stubs, ties etc.) are generated

Server

```
public class HelloImpl implements HelloIF,
 ServiceLifecycle {
  public String sayHello(String s) {
      return prompt + s;
  }
  public void init(Object context) throws
      ServiceException {
      // Configure security for endpoint
      //  1) Bind context to ServerHelper
      ServerHelper sh = ServerHelper.createFor(context);
      //  2) Config server security actions
     sh.addVerifyRequest().addSignResponse();
  }
}
```

The ServerHelper API

- ServerHelper integrated with JAAS (Java™ Authorization and Authentication Service)
- JAAS is used to locate credentials
  - Server private key
  - Server X509 certificate containing public key
  - Trusted client certificates and Certification Authorities (Cas)

Client

```
Remote proxy = (Remote) createProxy();

// Following steps are to enable security
//   1) Create and bind ClientHelper to proxy
CertificateClientHelper cch =
  CertificateClientHelper.createFor(proxy);

//   2) Config security actions
cch.addSignRequest().addVerifyResponse();

HelloIF hello = (HelloIF) proxy;
hello.sayHello("Duke!");
```

The ClientHelper API

- Similar to server-side, uses ClientHelpers
- Different kinds of ClientHelpers
  - ClientHelper has no client credential
  - CertificateClientHelper has client certificate credential
  - UsernameClientHelper has username password client credential
- ClientHelpers integrated with JAAS

Client side security config file

```
<xwss:Service>
    <xwss:SecurityConfiguration dumpMessage=”true”>
        <xwss:Sign>
            <xwss:X509Token certificateAlias=”my_cert”
                    keyReferenceType=”Direct”/>
        </xwss:Sign>
    </xwss:SecurityConfiguration>
    <xwss:Port name="{http://myserver.com/hello}HelloWorldIF">
            <xwss:Operation name="{...}sayHello”>
        <xwss:SecurityConfiguration>
            <xwss:UsernameToken useNonce=”true” id=”login”/>
            <xwss:Encrypt certificateAlias=”server_cert”>
                <xwss:Target type=”uri”>#login</xwss:Target>
                <xwss:Target type=”qname”>
                    {http://myserver.com/hello}name
                </xwss:Target>
            </xwss:Encrypt>
        </xwss:Operation>
    </xwss:Port>
```

Sign the enire message

Login to use the service

Encrypt the login name and the company name

Server side security config file

```
<xwss:Service>
    <xwss:SecurityConfiguration dumpMessage=”true”>
    </xwss:SecurityConfiguration>
    <xwss:Port name="{http://myserver.com/hello}HelloWorldIF">
                <xwss:Operation name="{...}GetCreditScore”>
        <xwss:SecurityConfiguration>
            <xwss:RequireSignature>
                <xwss:Target type=”qname”>
                    {http://schemas.xmlsoap.org/.../envelop/}Body
                </xwss:Target>
            </xwss:RequireSignature>
            <xwss:RequireEncryption>
                <xwss:Target type=”uri”>#login</xwss:Target>
                <xwss:Target type=”qname”>
                    {http://myserver.com/rate}name
                </xwss:Target>
            </xwss:RequireEncryption>
            <xwss:RequireUsernameToken id=”login”/>
        </xwss:SecurityConfiguration>
      </xwss:Operation>
    </xwss:Port>
```

Verify signature

Decrypt the login/password and the company name

Login to the service

WSS Security Samples in JWSDP 1.6

- Under <JWSDP_HOME>/xws-security/samples/
  - Simple sample
    - > Lets you plug in different client and server side config
    - > Support for digital signatures, username token authentication, encryption
  - Jaas-sample
    - > Uses the Java Authentication and Authorization service
    - > Obtain username and password at run-time
    - > Use JAAS to authenticate username and password
    - > http://java.sun.com/products/jaas

Web Services Security Roadmap for the JWSDP and SJAS Platforms

- WSS 1.1 support
- JSR 183 support
- SAML2.0 support
- Internal use of JSR 106 for Encryption
- Support for Policy Standards
- Integrate XWS-Security into JAXRPC 2.0
    - JAXRPC 2.0 defines Security APIs and Annotations
- Enhancements to JSR 196 Security Provider
  - Java Authentication SPI for Containers
    - Defines a way for authentication modules to integrate with containers so that container's identities can be used by providers

SAML, Liberty, Java Identity Management Suite

SSO

- Sign in once, don't sign in again to use other services secured through that security domain
- Simple SSO
- Cross-domain SSO
- Provides ease of use and better experience for end users

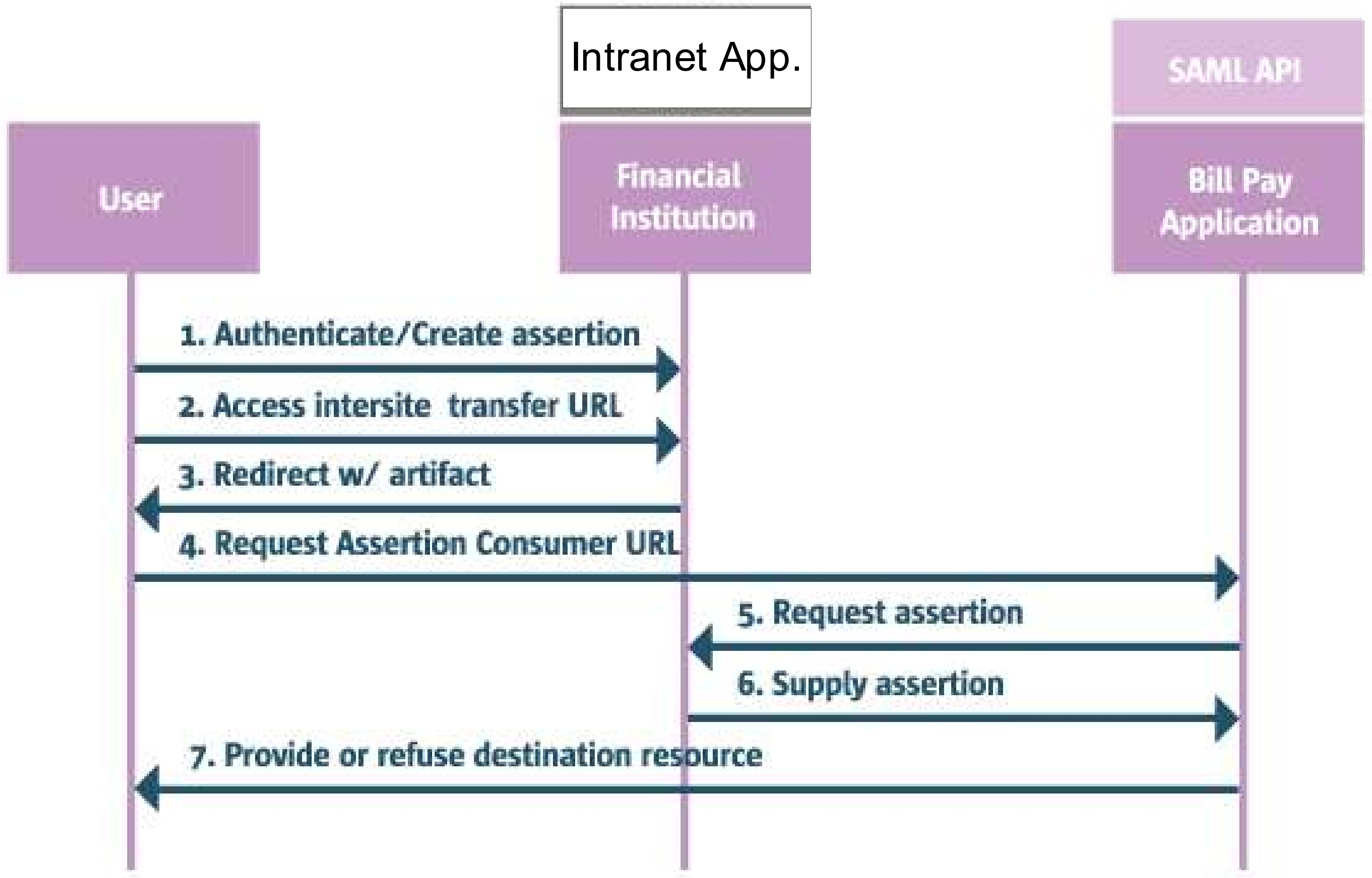
The concept of Federation

- Federation is a collection of entities that have formed a liasion for a purpose
- Identity federation
  - Federation wherein identity owning entities (companies) can come together and share identities of employees, customers, etc.
- Identity federation serves a lot of business goals
  - Better end user experience
  - Enable ways to form new partnerships and drive revenue

SAML

- XML-based framework for marshaling security and identity information and exchanging it across domain boundaries
  - Wraps existing security technologies rather than inventing new ones
  - Its **profiles** offer interop for a variety of use cases, but you can extend and profile it further
- At SAML's core: **assertions** about subjects
  - Assertions contain statements: authentication, attribute, entitlement, or roll-your-own

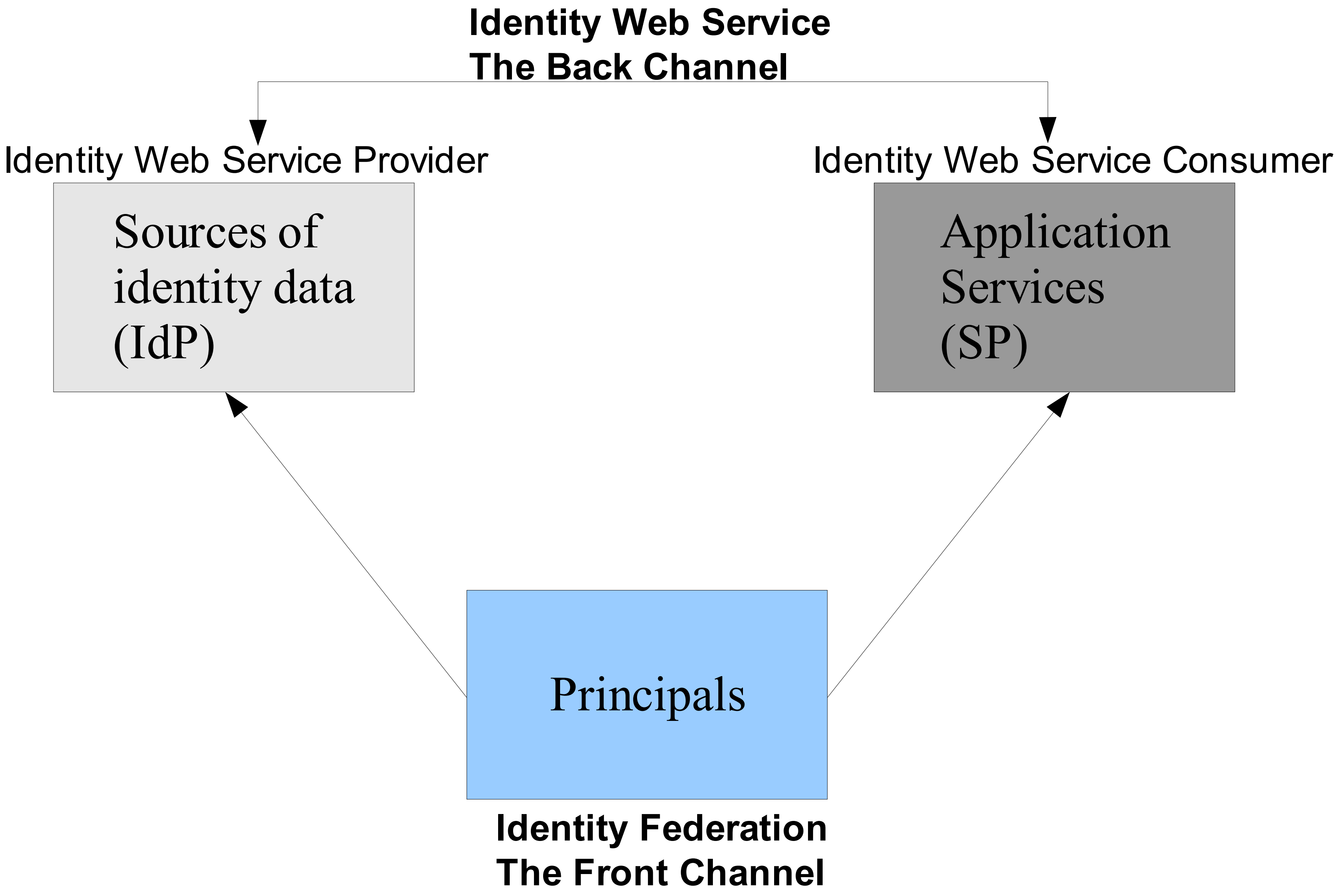
SSO Using SAML


Intranet App.
SAML API
User
Financial Institution
Bill Pay Application
1. Authenticate/Create assertion
2. Access intersite transfer URL
3. Redirect w/ artifact
4. Request Assertion Consumer URL
5. Request assertion
6. Supply assertion
7. Provide or refuse destination resource

Liberty

- Complete set of technical solutions for secure network identity management and usage in web applications and web services
- **Circles of trust** use a hub-and-spoke model
  - One (or a few) identity providers (IdPs), potentially many service providers (SPs)
- Supports many different devices and systems
  - Mobile clients and gateways are given special attention
- Privacy and permission-based attribute sharing are enabled throughout
  - Individual deployments might not need these features

Architecture


Identity Web Service
The Back Channel
Identity Web Service Provider
Identity Web Service Consumer
Sources of identity data (IdP)
Application Services (SP)
Principals
Identity Federation
The Front Channel

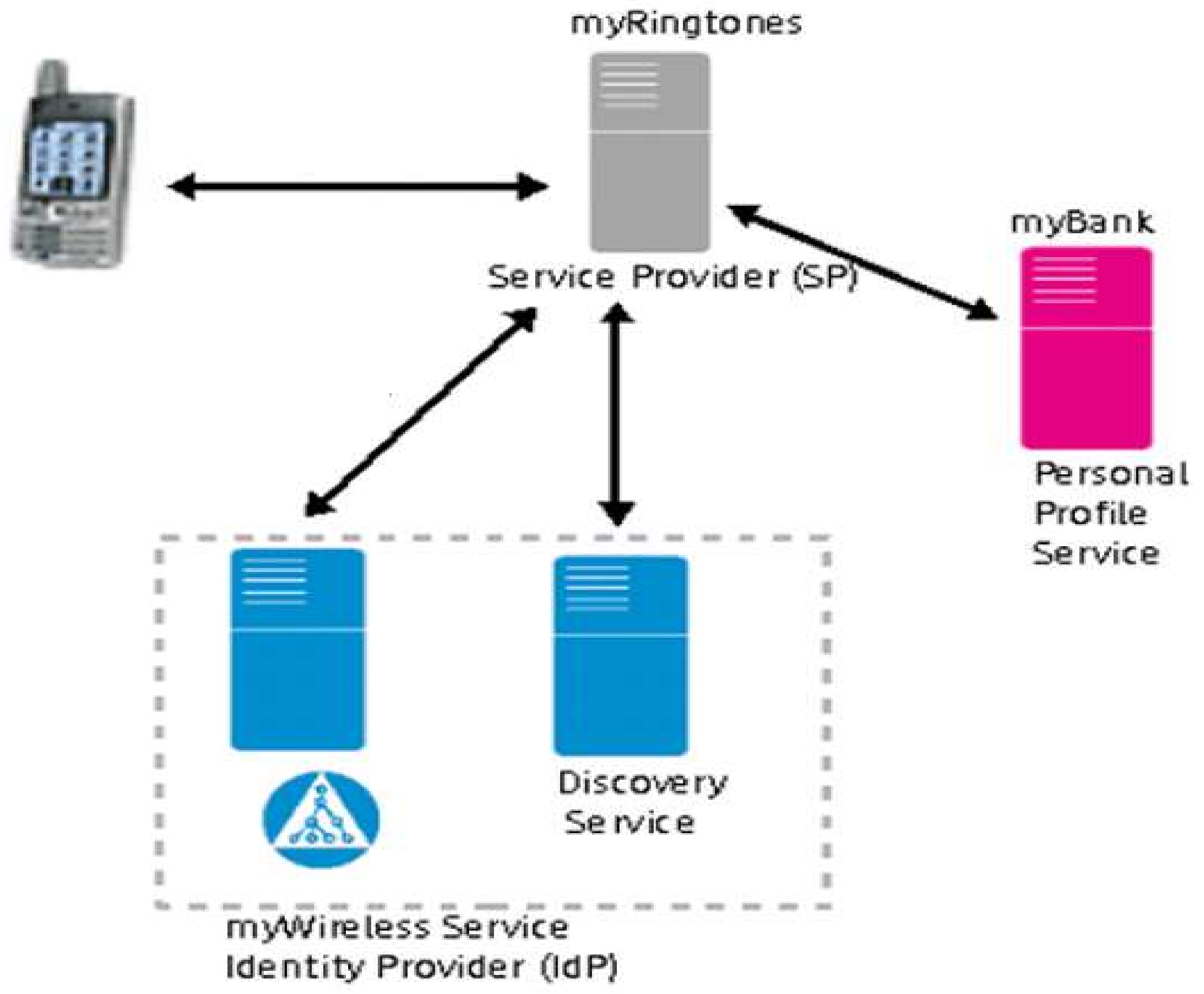
Liberty Architectural Modules

- Liberty Identity Federation Framework (ID-FF)
  - Enables identity federation and management
    - Identity and account linkage
    - Simplified SSO
    - Simple session management
- Liberty Identity Web Services Framework (ID-WSF)
  - Framework for building Web services that can
    - Do permission based attribute sharing with IdP
    - Describe and discover user's IdP
    - Associate security profiles with identity services

Liberty Architectural Modules (Contd.)

- Liberty Identity Services Interface Specification (ID-SIS)
  - Collection of specifications of interoperable services built on top of ID-WSF
    - Services such as identity registration, contact book, calendar, geo-location, presence or alerts
  - First ID-SIS: Personal Profile Identiy Service
    - Defines schemas for basic profile information of a user
    - Enables organizations to share a common dictionary or vocabulary of identity

Example of Liberty In Action



Sun Java System Identity Management Suite

- Access Manager
  - Access control, SSO (simple, federation, Windows Desktop)
- Federation Manager
  - Extends trust domain to include service providers as part of hub-and-spoke architecture
  - Provides secure federated services to let spokes leverage the core security and identity infrastructure services of the hub
- Identity Manager
  - User provisioning and identity synchronization (with active directory as well)

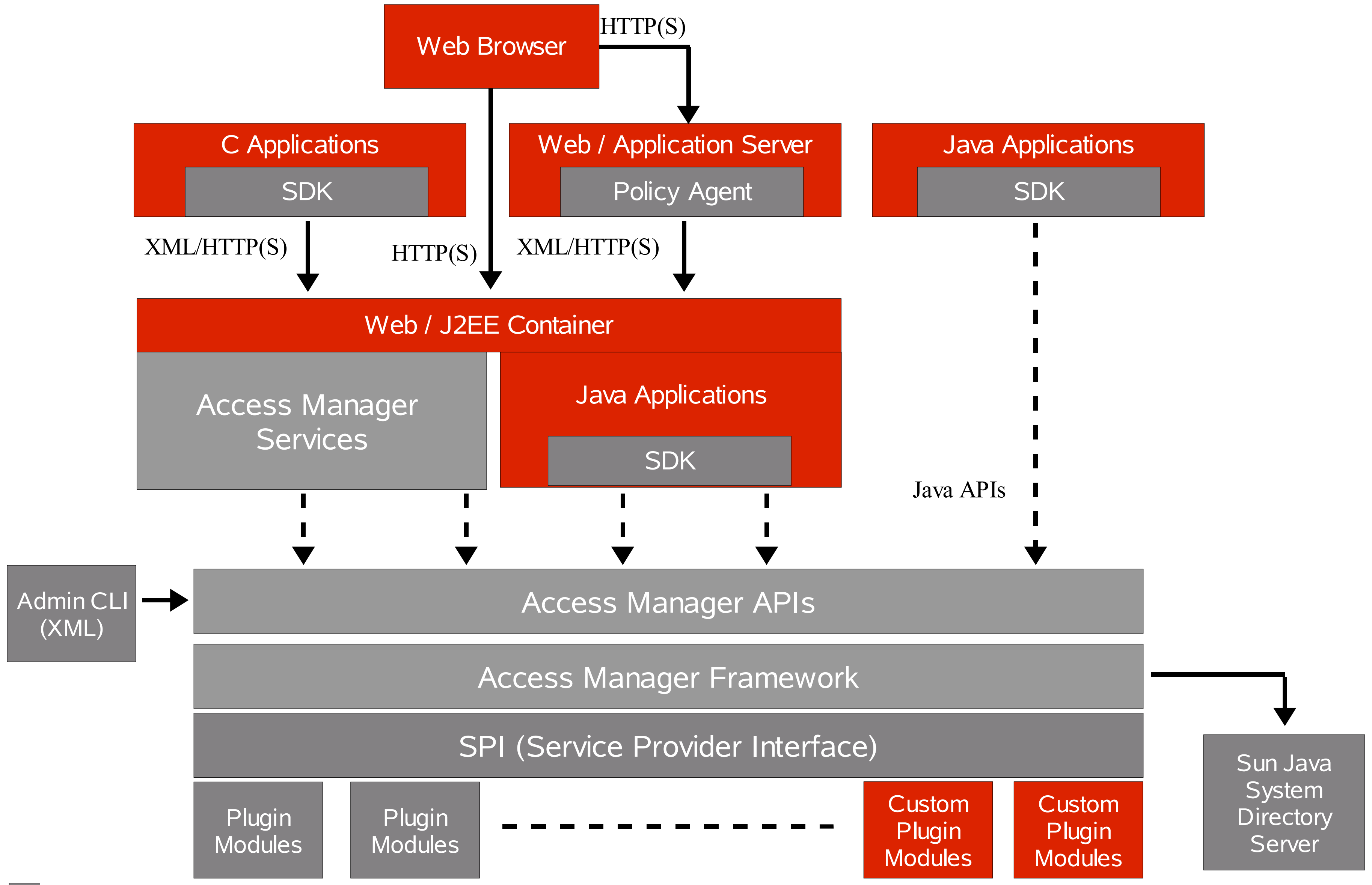
Sun Java System Identity Management Suite (Contd.)

- Identity Auditor
  - Provides a review of identity controls to help compliance and improve audit performance
    - Packaged audit policies for SOX and HIPAA
- Identity Manager Service Provider Edition
- Directory Server EE
  - Directory infrastructure for identities, policies, etc.

How Access Manager Works

- Intercept access to resource
- Authenticate user
- Issue token
- Repeat
  - Intercept access to resource
  - Use token to authorize access depending on policy
  - Provide identity data to resource
  - Log everything that happens
- Until session expires

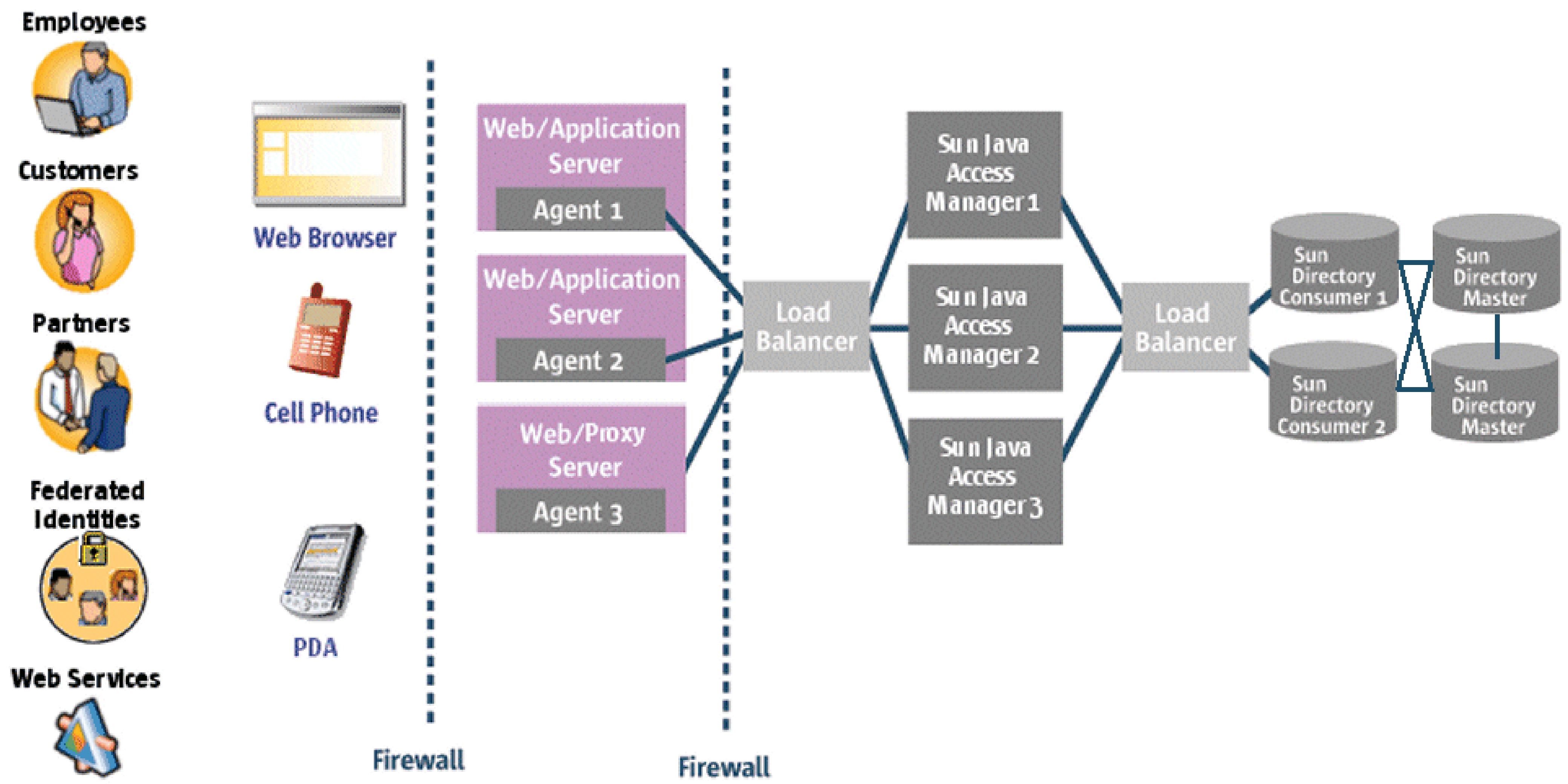
Access Manager Architecture



▨ Provided by Sun Java System Access Manager

- - ▶ Java APIs

Deployment Architecture



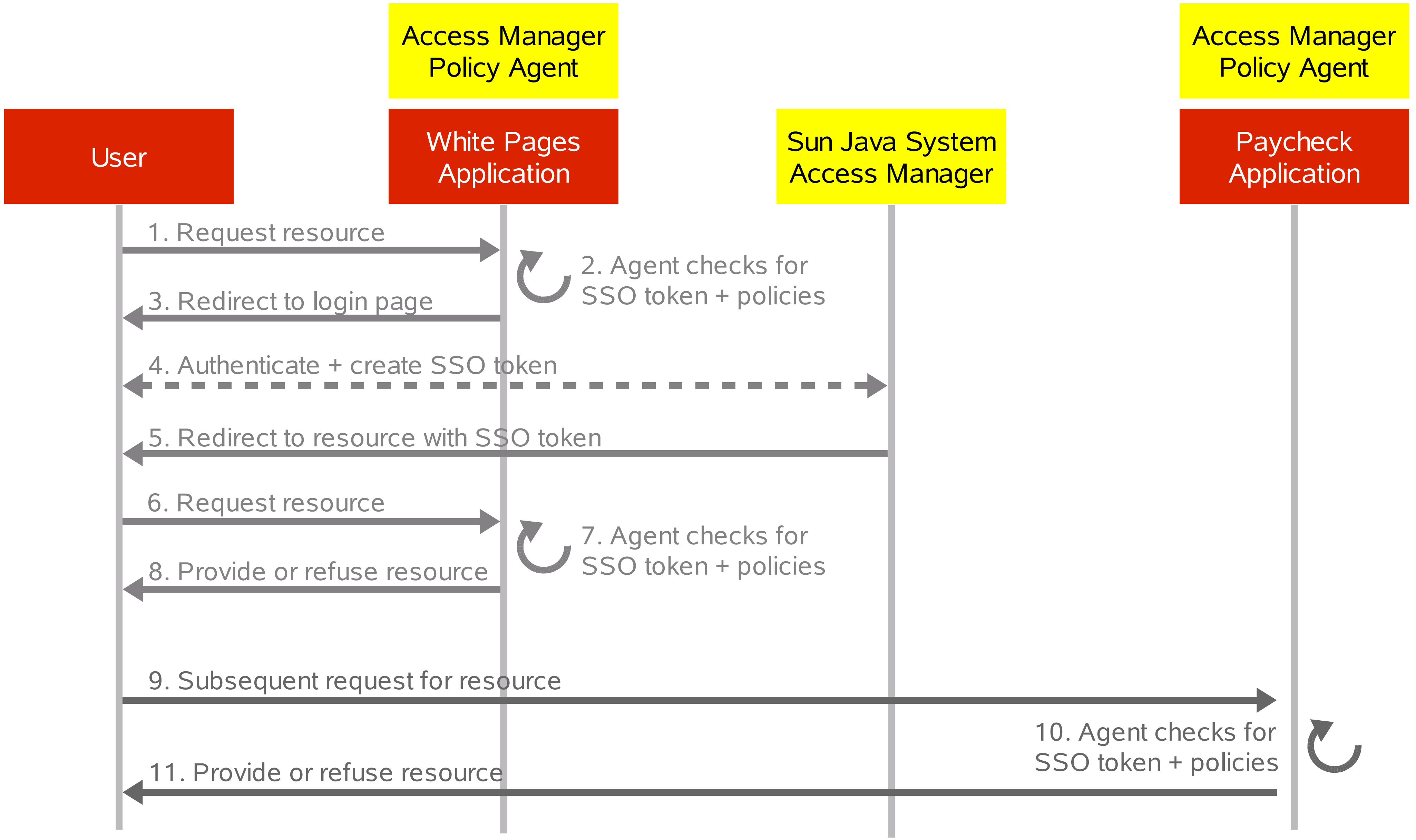
Authentication

- Standards-based, extensible authentication framework (JAAS: Java Authentication and Authorization Services)
- Supports multiple pluggable Authentication mechanisms
- LDAP, RADIUS, Certificate, SafeWord, RSA SecurID, Unix, Windows NT, Anonymous, Membership
- Custom authentication mechanisms using the SPI
- Multi-factor Authentication (Chained authentication mechanisms)
- Levels-based Authentication
- Levels assigned to authentication mechanisms
- Resource-based Authentication

Authorization

- Policy = Rules + Subjects + Conditions
  - Rules
    - Resource being protected – URL, access method, allow/deny
  - Subjects
    - Who is allowed access? User/role/group etc
  - Condition
    - Additional constraints – IP address, authN level/mechanism, day/time, session timeout
  - Referral policies, SPI allow customization

Web SSO Flow


Access Manager Policy Agent
Access Manager Policy Agent
User
White Pages Application
Sun Java System Access Manager
Paycheck Application
1. Request resource
2. Agent checks for SSO token + policies
3. Redirect to login page
4. Authenticate + create SSO token
5. Redirect to resource with SSO token
6. Request resource
7. Agent checks for SSO token + policies
8. Provide or refuse resource
9. Subsequent request for resource
10. Agent checks for SSO token + policies
11. Provide or refuse resource

Session Features

- Session upgrade
  - User provides additional credentials to access a resource with higher authentication requirements
- Client detection
  - Provide content based on client type – standard browser, WAP, etc.
- Resource-based session timeout
- Java & C Session/SSO APIs

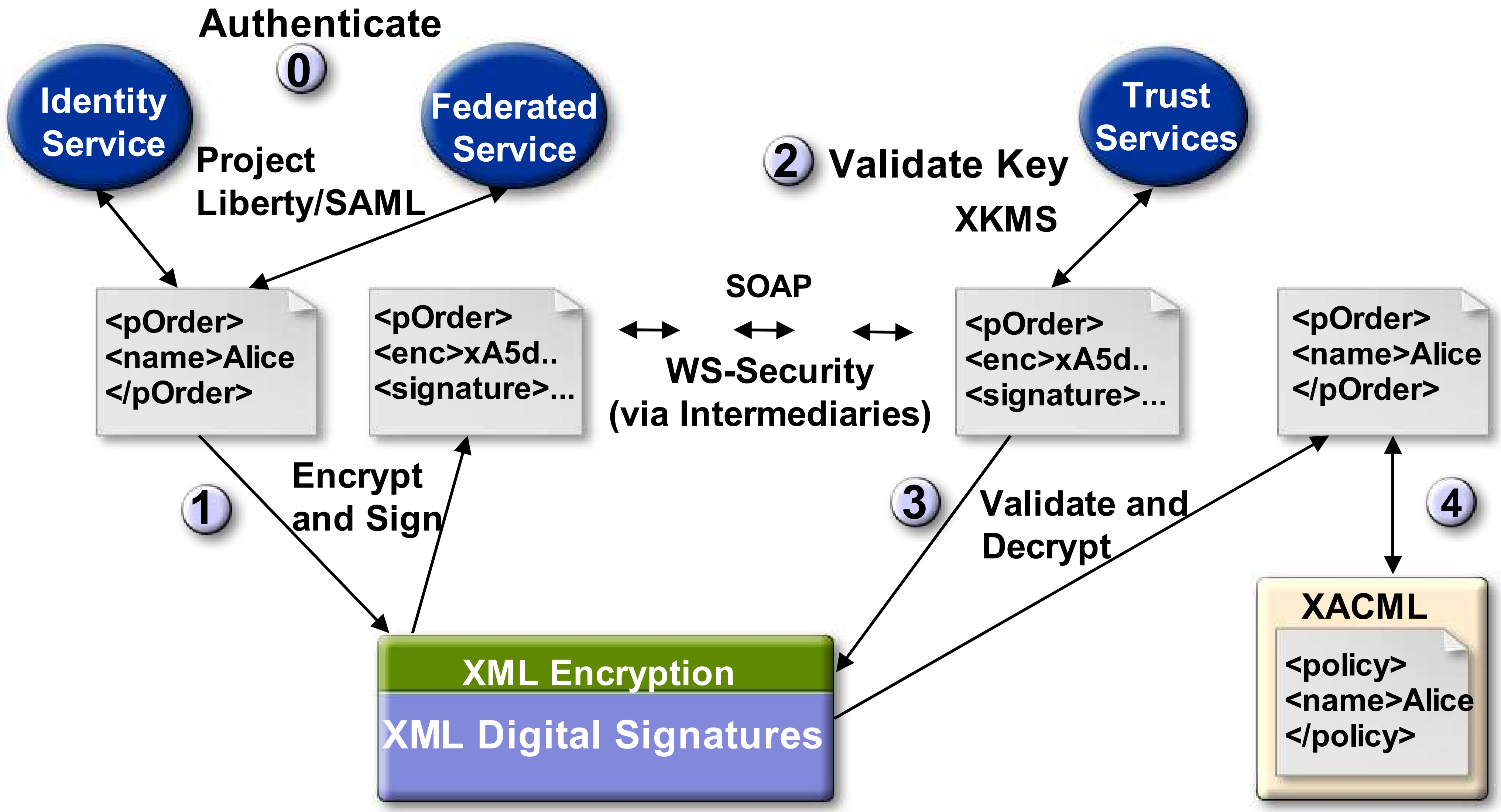
Windows Desktop SSO

- User-eye view
  - Log in to Windows
  - Surf to a protected resource
  - The resource recognizes me and gives me access based on policies, role etc
- That's it – the user logs in exactly once
  - No need for password sync process
  - Transparent integration for desktop users into web applications

IT WILL ALL BE FREE. REMEMBER TO TRY IT OUT.

FOR NOW, CHECK OUT OPENSSO [opensso.dev.java.net]

Panorama: Identity, Federation, and WSS


Authenticate
0
Identity Service
Federated Service
Project Liberty/SAML
2 Validate Key
Trust Services
XKMS
SOAP
<pOrder> <name>Alice </pOrder>
<pOrder> <enc>xA5d.. <signature>...
WS-Security (via Intermediaries)
<pOrder> <enc>xA5d.. <signature>...
<pOrder> <name>Alice </pOrder>
1
Encrypt and Sign
3
Validate and Decrypt
4
XACML
<policy> <name>Alice </policy>
XML Encryption
XML Digital Signatures

In summary...

Resources

- opensso.dev.java.net
- jwsdp.dev.java.net
- Paper on building identity enabled Web services developers.sun.com/prodtech/identserver/reference/techart/id-enabled-ws.html#3
- Standards and Technologies
  - www.w3.org
  - www.oasis-open.org
  - www.ws-i.org

SUN TECH DAYS 2005-2006
A Worldwide Developer Conference

Innovation Happens Here.

Rima Patel Sriganesh
rima.patel@sun.com