TRUSTSTIX INC

Patterns, Practices & Results

Less than 10 things you should know about

Web 2.0 Security

Version 1.0
January 2008

Shivaram H. Mysore
www.TrustStix.com

Table of Contents

Executive Summary

Web 2.0 is an umbrella term coined to include technologies used for providing user-centric web based services. Here, the services are architected and programmed so that they can be personalized and used dynamically. The architectural philosophy is called Service Oriented Architecture (SOA).

This document provides security aspects for Web 2.0 based Services. It provides a comprehensive list of threats that need to be considered for mitigation when deploying Web 2.0 services. It also provides ideas on mitigating the described threats.

Audience

This document is intended for:

1. CIOs and Enterprise IT Professionals (ex. Administrators, Directors) who are planning or implementing or deploying Web 2.0 Services
2. Network & System Architects

Web Services: REST & SOAP

Web Services are programmatic interfaces for application to application communication. This interface is normally exposed via a Web Service Description Language (WSDL). There are 2 architectural models associated with Web Services (1) REST and (2) SOAP based.

REST is an acronym for Representational State Transfer. It is a stateless client-server protocol. Here each message contains all the necessary information to understand the request. As a result, neither the client nor the server needs to remember any communication-state between messages. Messages are normally carried over HTTP (Hyper Text Transfer Protocol) with XML or JSON (Java Script Object Notation) payloads.

Here is an example of a REST based URI (Uniform Resource Indicator):

```
http://www.example.com/validate?uname=dilbert&pwdhash=z786
fwnu&output=xml
```

Another REST example is from Amazon Web Services

```
http://xml.amazon.com/onca/xml2?t=webservices-20&dev-
t=[developer's ID goes here]&AuthorSearch=[author name
goes here]&mode=[product line goes here: must be
books]&type=[lite or heavy]&page=[page # goes
here]&sort=[sort type goes here (optional)]&f=xml
```

SOAP is a XML based end-to-end application layer protocol. SOAP messages are used when message-level security is desired. The message contains all the necessary information to protect itself and can contain optional Quality Of Service (QOS) attributes (for reliable messaging, transactions, security, etc). This meta-data information is recorded as a part of SOAP Message header. With security information in the message header, end to end security can be enforced as compared to Transport Layer Security (TLS/SSL) which is hop to hop.

REST and SOAP based architectural models can be used together in deployments.

REST model is commonly used when resources can be directly accessed from a web browser. This enables content providers to better use their existing enterprise architectures providing caching and application acceleration services and hence scale gracefully. REST methodology is also used for streaming large amounts of data. Resource manipulation is normally enabled via web forms (GET, POST, PUT, and DELETE) similar to usage in WebDAV (Web-based Distributed Authoring and Versioning) protocol (ex. used in Calendar servers). Really Simple Syndication (RSS) and Atom feeds are common examples of REST based methodology.

When SOAP messages are sent only one-way (say from client to server or from server to client), they are identified as “document-style” SOAP messaging. If the message exchange is request-response, then they are identified as “RPC-style” (RPC: Remote Procedure Call) SOAP messaging.

SOAP RPC implies both an encoding and a request/response message exchange, and is about sending 'objects' over the wire. “document-style” is about sending XML documents. RPC implies 'data binding' mapping; document-style doesn't.

SOAP model is predominantly used for (1) Protocol Translation (adapter) – convert a JMS (Java Message Service) message coming into the enterprise to SOAP message and vice versa (2) Service workflow (business process) integration (3) Processing QOS attributes on the message for additional operations (4) Message-level security (ex. encrypting or digitally signing individual SOAP messages).

As mentioned before, a SOAP based Web Service is exposed by a WSDL. The WSDL includes all the necessary information to interact with the service like, arguments it is willing to accept and how the results are provided including data formats. It also includes necessary information to locate the service and the protocol to be used for communicating with the same.

Below is an example of how a WSDL looks like. This example provides a service to get stock quote information.

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
xmlns:tns="http://example.com/stockquote.wsdl"

xmlns:xsd1="http://example.com/stockquote.xsd"

xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <schema
targetNamespace="http://example.com/stockquote.xsd"
            xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="TradePriceRequest">
        <complexType>
```



```
          <all>
            <element name="tickerSymbol" type="string"/>
          </all>
        </complexType>
      </element>
      <element name="TradePrice">
         <complexType>
           <all>
             <element name="price" type="float"/>
           </all>
         </complexType>
      </element>
    </schema>
  </types>

  <message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
  </message>

  <message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
  </message>

  <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput"/>
      <output message="tns:GetLastTradePriceOutput"/>
    </operation>
  </portType>

  <binding name="StockQuoteSoapBinding"
type="tns:StockQuotePortType">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
      <soap:operation
soapAction="http://example.com/GetLastTradePrice"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <service name="StockQuoteService">
    <documentation>My first service</documentation>
    <port name="StockQuotePort"
binding="tns:StockQuoteSoapBinding">
      <soap:address
location="http://example.com/stockquote"/>
    </port>
  </service>
```



```
</definitions>
```

There are three important items of interest here (as highlighted):

1. **portType**: describes a single operation `GetLastTradePrice` which uses the message types for input and output
2. **binding**: indicates that the communication is via SOAP
3. **port**: included in the service description, which includes endpoints or ports that define where the messages need to be sent. In other words, it associates the binding with the URI `http://example.com/stockquote` where the running service can be accessed.

Threats

Uses of Web Services (REST or SOAP) are always in addition to traditional web usage. Traditional web usages include use of HTML based web pages (include JSP/ASP and JavaScript enabled pages). As with traditional web, web services also use HTTP as the protocol. Even SOAP messages are conveyed over HTTP protocol. There is also relatively small usage of other protocols like SMTP, FTP and likewise. All of these imply that traditional protocol level attacks on HTTP, SMTP, FTP, etc apply even when conveying SOAP/XML messages. Some of these attacks include Denial Of Service (DOS), Abuse of Functionality, authentication and authorization.

In the Web 2.0 world, RSS/ATOM based feeds are popular forms of distributing content. The core protocols are even used for updating content. The following sections describe some of the threats associated with the same.

Feed Injection

In the case of RSS/Atom based feeds, there are 2 main things to consider:

1. The feed content which is in XML
2. The ultimate resource which is normally a HTML based page (could be XHTML/JSP/ASP page)

XML based feed content is susceptible to traditional attacks (similar to HTML based pages) like code/SQL/LDAP injection, OS commanding (remote execution of Operating System executables), Buffer Overflow and content spoofing.

Once the feed is inspected for threats, the ultimate resource as referenced in the "item" element of the feed when fetched will be subject to similar processing of the content.

Authentication

If a feed requires authentication, then authentication based attacks such as replay, session/credential hijacking, and insufficient authentication must be addressed.

When RSS/Atom is used as WebDAV like protocol for uploading data based on identity, authentication is necessary. Atom Protocol [8] has a standardized authentication methodology. One of the authentication protocols that can be used is Web Services Security Username Token Profile [10]. Using Web Services way of

authentication helps with better integration with SOAP based Web Services. A good description of this works is provided in an article on Atom Authentication [9].

Validation

RSS feed mime-type is still being standardized to use application/rss+xml. In many cases, mime-type text/xml is used extensively. If RSS 1.0 is used, then there may be cases when mime-type application/rdf+xml may also be used.

Atom feeds normally use a mime-type of application/atom+xml.

When processing mime-types, header content inspection must also be performed to validate and get feed type (RSS or Atom) version information.

With the obtained feed type and version information, XML well formed-ness (making sure that all the "angle" brackets are correctly wrapped) and schema validation on the content needs to be performed.

Client Side Attacks: Cross-Site Scripting & Forgery

The nature of a feed is such that it can contain content from various sources. When such content is presented to the user in a single web page, applying a single security model for a browser is almost impossible. This will eventually lead to cross-site scripting and forgery attacks.

Client Side Attacks: Command Execution and Zones

Feeds when consumed directly by client side applications can be dangerous. Much of the security is being focused towards browser based interaction. Email clients are another source of feed readers. For example, Outlook 2007 has a functionality to subscribe to RSS feeds which will be in the inbox of the user. This means that the RSS feed is stored as a file on the local disk (similar threats are possible when caching in browsers are turned on for storing RSS content). With the feed stored as HTML on the local disk, when the html engine opens the file (IE or Outlook), the security zone and privileges differ and hence introduce threat to the system. If the ActiveX is signed, then there may not be an information alert shown to the user. As with AJAX, JSON objects may be present in the code. This means that XMLHttp and XMLHttpRequest APIs can be used to perform port scanning, make network calls and other IO operations as a privileged user.

As feeds are consumed programmatically, with the above kind of threats, zero-day attacks can be launched.

Client Side Attacks: Generic

AJAX and Rest based services go hand in hand. With AJAX, the client has lot more information now than ever before so that it can respond with ease for different requested views. This naturally enables information disclosure and increased attack surface. So, care must be taken as to when and how much of information needs to be provided to the client.

By default, XMLHttpRequest API executed by a browser, can only make requests to the originating server. For Mozilla based browsers, digitally signing the script

enables this API to be used more easily, but, the signature is not compatible with IE. With some amount of work and in some cases, a combination of methodologies can be used to dynamically create JavaScript code to make a request to an application proxy thus circumventing the call to the originating server.

An interesting blog entry on JSON and browser security is available here [7]

Impact of Network Architecture

This section illustrates the impact of deployment architecture on security. The scenario section illustrates a customer need. The sections on "Architecture Solution x" provide ways that the scenario can be addressed.

Scenario

An end-user customer is interested in getting feeds from various sources in a single web page.

Architecture Solution 1

The user creates a JavaScript enabled single html page stored on a local disk to dynamically get RSS/Atom feed content from various sources and populate the same in the custom web page created.

The impact of this solution is that there may not have been enough check performed by the client for malicious content from the feeds. More over as the page is loaded from a local disk, zone attacks are possible as the loaded page is running with different privileges. All the scripts running on the page including the ones loaded as a part of the feeds have the same privilege.

If a similar page is provided by a portal, then remote zone attacks are also possible in addition to the above.

Architecture Solution 2

Here a portal provides a way to customize feeds for the user. The portal creator fetches the feeds for the user and then populates the same into the web page.

As the portal creator is fetching feed data, the content must be inspected for malicious code which could harm the end-user otherwise, the portal may be liable for security issues caused to the end-user. The portal's internal programs may be affected if the content is not inspected and the malicious code was actually meant to cause problems to the portal programs.

Architecture Solution 3

This is similar to Architecture Solution 2, except that the fetched data is stored in a database by the portal before being recomposed for a web page presented to the user.

Similar threats as per Architecture Solution 2 are seen here. As the architecture now involves a database, SQL injection attack vector must also be considered.

Architecture Solution Summary

Depending on what kind of architecture is deployed, the requirements for application firewall may differ in addition to the placement of same in the network infrastructure.

Threat Mitigation Management

Threat mitigation is a continuous process for the lifetime of the application or service. An application or service is built on a set of assumptions and infrastructure and if these change, threat assessment is redone. Threat modeling [4] is normally developed based on usage scenarios so that mitigation techniques are applied to address these scenarios. If the usage scenarios change, then threat assessment must be redone.

For most of the threats addressed earlier in this paper, there are various commercial products that are available to mitigate some or all of it. Product selection is based on the usage scenario and the requirements of the solution being developed. Note that if the solution is not architected appropriately for the usage scenario, then, even the best products may not be of enough value. Usage scenarios are even more critical in the Web Services area as usage patterns can dramatically change based on how the services are configured to be used. While best practices [5] can be used to develop the software, usage model may break down the security of the system as a whole.

References

1. SQL Injection Attacks by Example: http://unixwiz.net/techtips/sql-injection.html
2. SQL Injection – Wikipedia - http://en.wikipedia.org/wiki/SQL_injection
3. XPATH and Related Specifications: http://www.w3.org/Style/XSL/
4. Threat Modeling, Microsoft Press & Online content:
   http://www.microsoft.com/mspress/books/6892.aspx
   http://msdn2.microsoft.com/en-us/security/aa570411.aspx
5. Security Development Lifecycle, Microsoft Press:
   http://www.microsoft.com/mspress/books/8753.aspx
6. Web Application Security Consortium – Threat Classification:
   http://www.webappsec.org/projects/threat/classes_of_attack.shtml
7. JSON & Browser Security: http://yuiblog.com/blog/2007/04/10/json-and-browser-security/
8. Atom Protocol: http://en.wikipedia.org/wiki/Atom_(standard)
9. Atom Authentication: http://www.xml.com/pub/a/2003/12/17/dive.html
10. Web Services Security Username Token Profile: http://www.oasis-open.org/committees/wss/documents/WSS-Username-02-0223-merged.pdf

About the Author

Shivaram Mysore is a Software Architect and Product Strategy consultant mainly in the areas of Strong Authentication, SOA and Secure Web Services. Currently, he is an invited expert in the XML Security Specifications Maintenance Working Group at W3C.

Previously, Shivaram was a Senior Technical Program Manager working on Windows Security @ Microsoft. His focus areas included Core Cryptography, authoring/reviewing Client-Server Protocol Specifications, FIPS Compliance, Smart Card Infrastructure for Windows and related Standards.

Shivaram represented Microsoft in ANSI/INCITS B10 Working Group and W3C, a Standards Organization. He has been a contributor for XML Encryption and Key Management (co-chair and co-editor) Specifications. Shivaram has also been the Chairman of the PC/SC Standards Organization

Before Microsoft, Shivaram was a Software Architect at Sun Microsystems's Java Software Division. He has worked on many projects in the areas of Cryptography and Security for the Java Card system, Messaging Servers, Web Servers and emerging technologies for Securing Web Services, Payment Systems and Digital Rights Management.

The author would like to thank Rich Salz, Byron Kataoka, Minoo Gupta, and Sridhar Guthula for providing input, careful review, and feedback on this paper.

To provide suggestions, comments and feedback on this paper, please send an email to whitepaper-comments@TrustStix.com. Alternatively, you can submit comments on my blog: http://www.truststix.com/blog

