



Using Sprajax to Test AJAX Security

Dan Cornell, OWASP San Antonio **Leader**
Principal, Denim Group, Ltd.
dan@denimgroup.com
(210) 572-4400

**OWASP
AppSec
Seattle**

Oct 2006

Copyright © 2006 - The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document under the terms of the Creative Commons Attribution-ShareAlike 2.5 License. To view this license, visit <http://creativecommons.org/licenses/by-sa/2.5/>

The OWASP Foundation
<http://www.owasp.org/>

SPRAJAY



Agenda

- Introduction
 - ▶ AJAX Security Basics
- Current Black Box Scanners
 - ▶ Issues with Current Scanners
- How Sprajax is Different
- Demonstration
- Sprajax Approach and Architecture
 - ▶ Example: Microsoft Atlas Support
- Next Steps
- Questions



Introduction

- Dan Cornell
- Principal of Denim Group, Ltd.
- MCSD, Java 2 Certified Programmer



AJAX Security Basics

- Shares many principles with normal web application security
- Risks are poorly understood
- AJAX increases an application's attack surface
- Same problems as before:
 - ▶ SQL injection
 - ▶ Parameter tampering
 - ▶ Authentication/Authorization issues



Why Sprajax?

- Deal with the issue of increased attack surface
- Deal with the issue of multiple AJAX frameworks



Current Black Box Scanners

- Current scanners are good at scanning traditional web applications
 - ▶ Pages, forms, parameters
 - ▶ Normal HTTP request are easy to craft
- Current scanners have *limited* AJAX abilities
 - ▶ Scan JavaScript for URLs
 - ▶ Parse/execute JavaScript to find endpoints

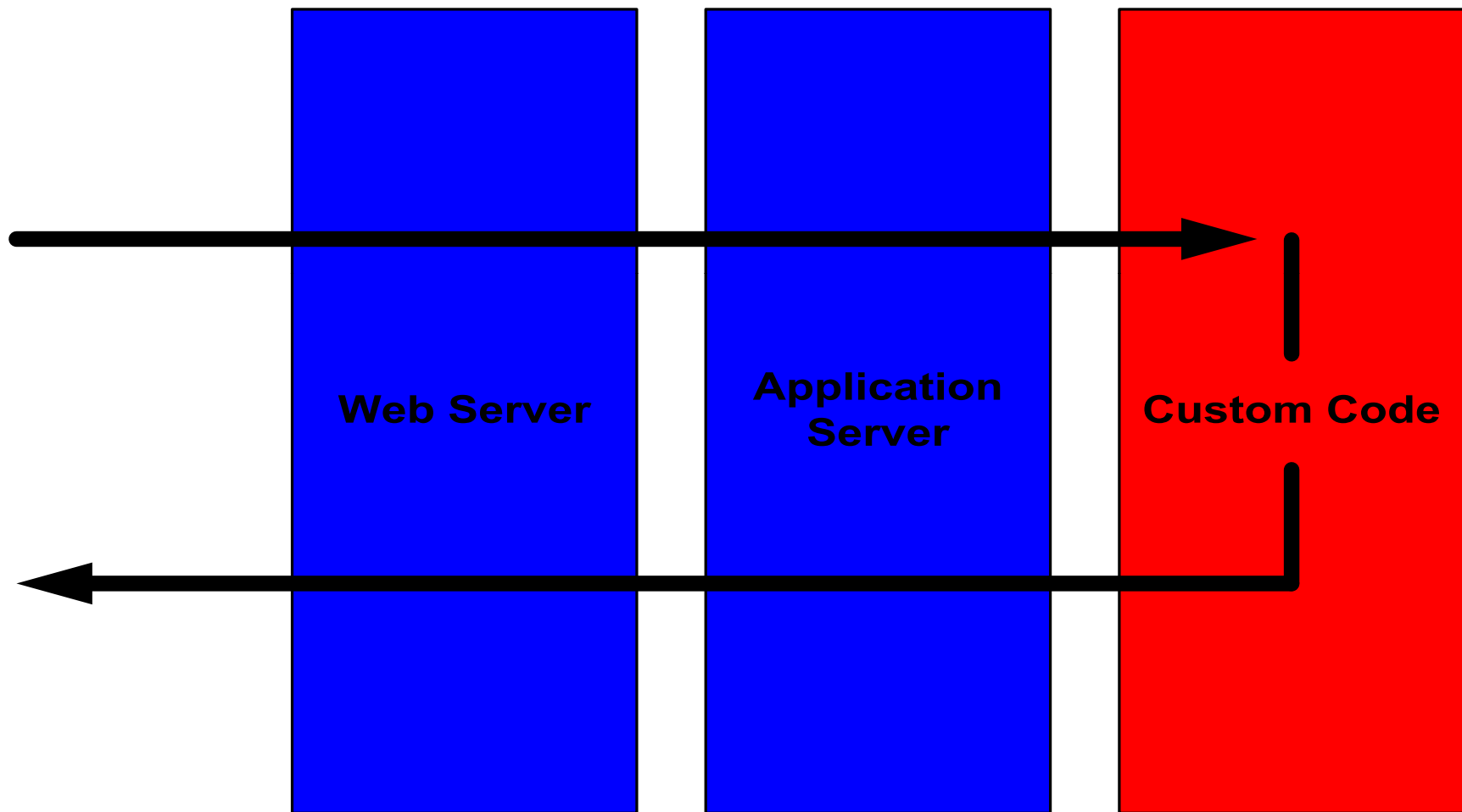


Issues with Current Scanners

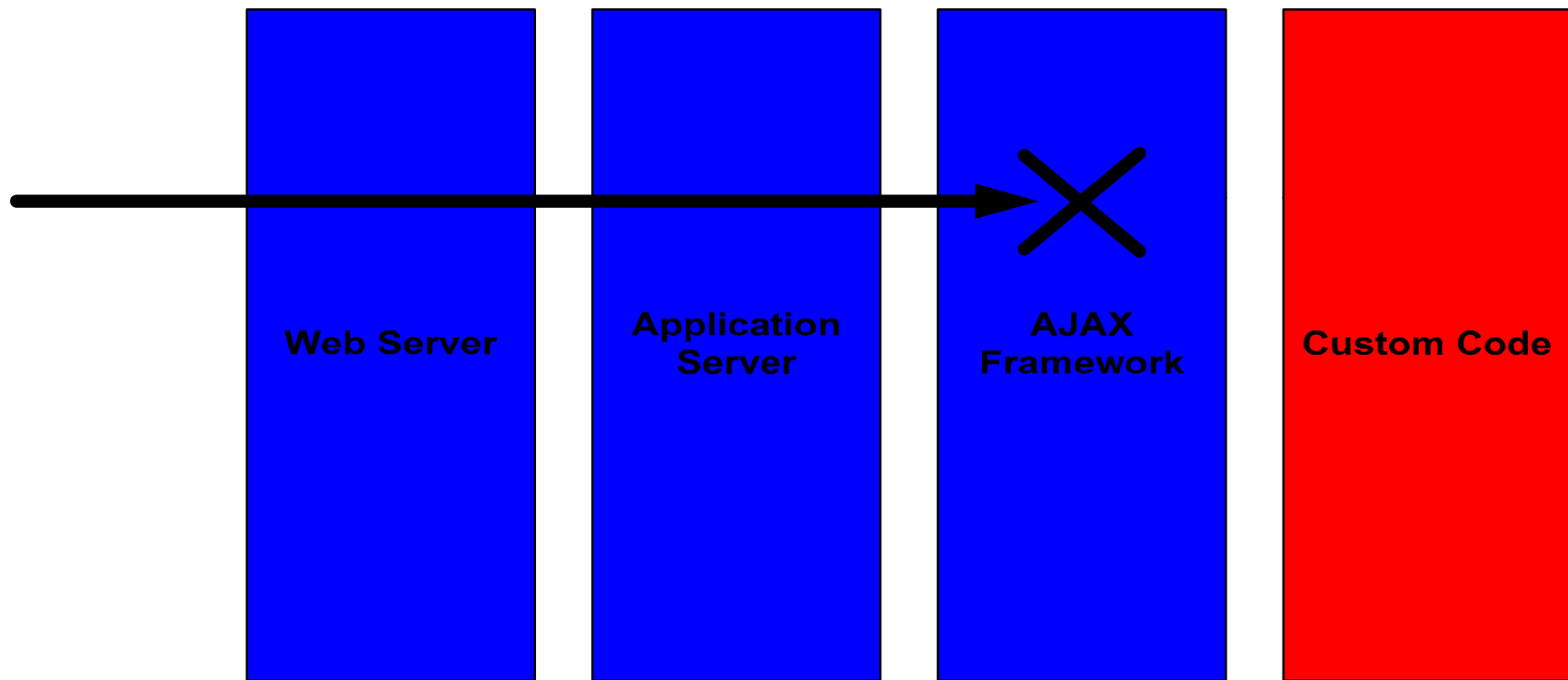
- AJAX applications often use frameworks that build on top of raw XMLHttpRequest
- Frameworks do not necessarily use plain HTTP requests
 - ▶ JSON for Atlas
 - ▶ Serialized Java for Google Web Toolkit
 - ▶ And so on...
- Normal HTTP POST data:
 - ▶ `key=Zen+and+the+Art+of+Motorcycle+Maintenance&key=Cryptonomicon`
- JSON HTTP POST data:
 - ▶ `["Zen and the Art of Motorcycle Maintenance", "Cryptonomicon"]`
- If the AJAX framework expects JSON (or something else) it will never see normally-formatted requests



Normal HTTP Request Sent to Web Application



Normal HTTP Requests Sent to AJAX Framework

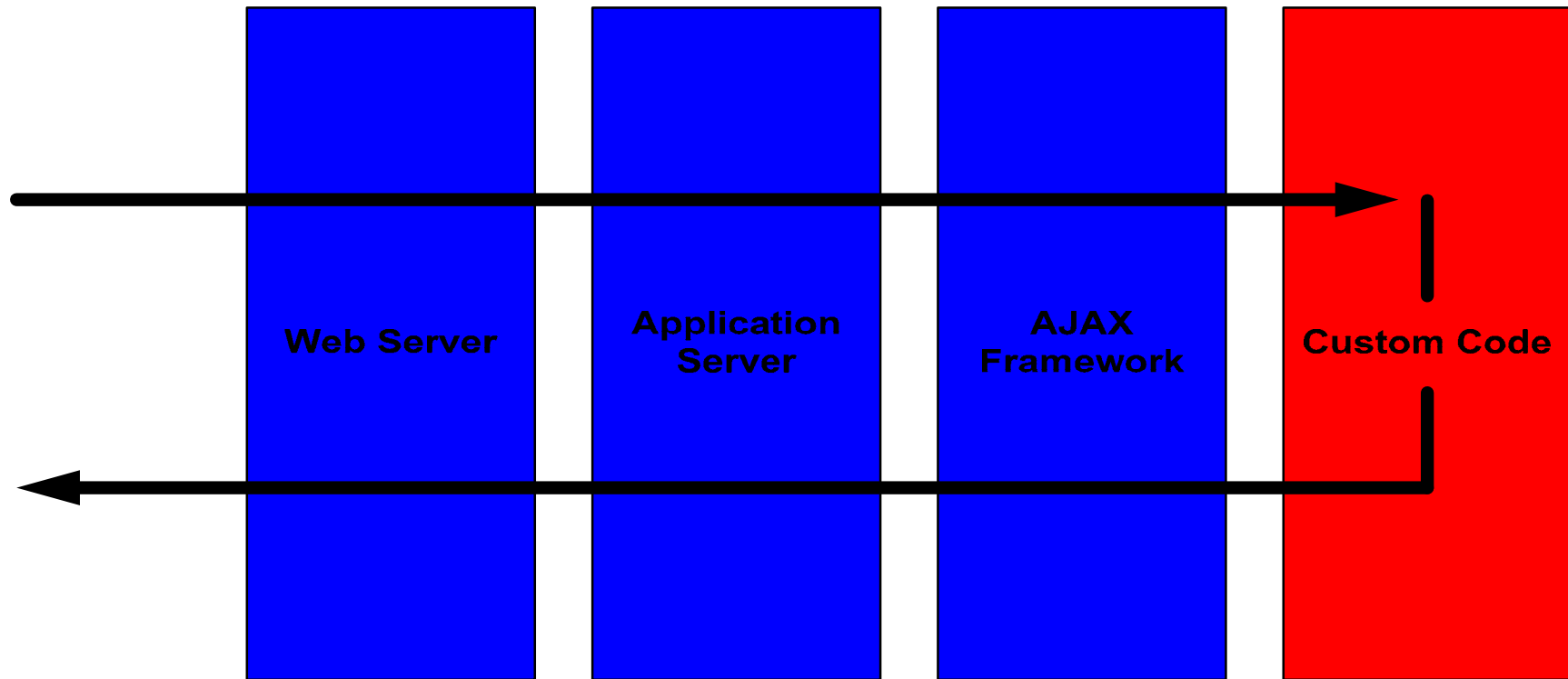


How Sprajax Is Different

- Spiders web applications as current scanners do
- Detects AJAX frameworks in use
- Detects AJAX-specific endpoints for the frameworks in use
- Fuzzes endpoints using framework-appropriate HTTP requests



AJAX Web Request Sent to AJAX Framework



Types of Vulnerabilities

■ Technical Vulnerabilities

- ▶ Surface due to insecure programming techniques
- ▶ Typically due to poor input handling, input validation and output handling and escaping
- ▶ Most “scanner” tools primarily find technical vulnerabilities
- ▶ Remediation: coding changes

■ Logical Vulnerabilities

- ▶ Surface due to insecure program logic
- ▶ Typically due to poor decisions about trust
- ▶ Most “scanner” tools are powerless to find logical vulnerabilities
- ▶ **Most “scanner” tools are powerless to find logical vulnerabilities**
- ▶ Remediation: architecture and design changes



Sprajax Limitations

- Should actually be “limitations of automated black-box testing”
- Can find *technical* application flaws
 - ▶ SQL injection
 - ▶ Cross site scripting (XSS)
 - ▶ Bad error handling
- Can't find *logical* application flaws
 - ▶ Many parameter and cookie tampering flaws
 - ▶ Authentication/authorization
- Not a limitation of the tool, but a limitation of the approach



Demonstration

- Simple Sprajax demonstration on a Microsoft Atlas site
- Simple Sprajax demonstration on a Google Web Toolkit site
 - ▶ Footprinting and fuzzing are in-progress



Sprajax Approach and Architecture

- NOTE: Included Open Source packages do not necessarily endorse Sprajax
- Spider the web site
 - ▶ Uses Jeff Heaton's C# spider (www.jeffheaton.com)
- Determine what frameworks are in use
 - ▶ Look at included JavaScript files
- Enumerate AJAX endpoints
 - ▶ Plugin architecture "watches" pages and tags through the course of the spidering
 - ▶ Implement DocumentWorkerListener interface
- Fuzz the endpoints with framework-appropriate requests
 - ▶ Microsoft Atlas uses SOAP web services with JSON
 - ▶ Uses DynWSLib for dynamic SOAP client creation (www.thinktecture.com)



Sprajax Fuzzing

■ Use a list of “interesting” values for various data types:

- ▶ String: `string.Empty`, `'JUNK'`, `"JUNK"` and so on
- ▶ Integer: `int.MinValue`, `-1025`, `-1024`, `-1023`, `-1`, `0`, `1` and so on
- ▶ Single Float: `float.MinValue`, `float.MaxValue`, `float.NaN`, `float.NegativeInfinity`, `float.PositiveInfinity`, `0.0` and so on



Sprajax Fuzzing

- Fuzzing creates an n-dimensional search space based on lists of primitives
 - ▶ Strings: currently 6
 - ▶ Integers: currently 25
 - ▶ Single Floats: currently 9
 - ▶ Double Floats: currently 9
- MyMethod(int) – 1D - 25 calls
- MyMethod2(int, string) – 2D – 150 calls
- MyMethod3(int, int, string) – 3D – 3750 calls
- Adding multi-threading will be key going forward



Example: Microsoft Atlas Support

■ Included files that are an indicator

- ▶ Atlas.js (older versions)
- ▶ WebResource.axd

■ Web Service endpoints indicated by:

```
<page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
  <references>
    <add src="Services/UserService.asmx/js"
        onscriptload="UserService.path=
          '/DenimGroup.Sprajax.Atlas.DemoSite/Services/UserService.asmx'" />
```

■ Valid requests make calls to SOAP Web Services

- ▶ Object serialization uses JSON rather than XML

```
["Zen and the Art of Motorcycle Maintenance", "Cryptonomicon"]
```



Next Steps

- More modular persistence support
- Add support for more AJAX frameworks
- Increase sophistication of testing
- Improve fuzzing
- Break out into individual tools



Next Steps: More Modular Persistence Support

- Right now SQL Server 2005 is required
 - ▶ Not really necessary – how many people need to compare results across scans at the current time
- People have requested MySQL support
 - ▶ Side note: Run using Mono?
- Replace current implementation with a Provider model
 - ▶ Support for SQL Server, MySQL (perhaps) and in-memory



Next Steps: More AJAX Frameworks

- Google Web Toolkit (GWT)

- ▶ Detection already works
- ▶ Finding endpoints is more complicated but not impossible
- ▶ Requests appear to send serialized Java objects

- Direct Web Remoting (DWR)

- And so on – at least detect all major frameworks and fuzz test the most popular

- Next release will have more modular design so that the plugins can be developed and maintained separately



Next Steps: Increase Sophistication of Testing

- Current: Only looking for error responses
 - ▶ SOAP errors
- Can tag inputs as being associated with a vulnerability type (SQL injection, Cross Site Scripting, etc)
- Can flag suspicious text in error messages
 - ▶ ODBC
 - ▶ SQL
- Test for injection attacks that might not result in errors
- Could also add tests for flawed versions of AJAX frameworks
 - ▶ More like what you would see from Nessus



Next Steps: Improve Fuzzing

- Multi-threading will be key
- Current only methods with primitive parameters are supported
- Add support for objects with properties as their own n-dimensional spaces to be traversed
- Will eventually need to get “smart” about which combinations are selected
 - ▶ Selectively choose input patterns
 - ▶ Data mine the results



Next Steps: Break Out Into Individual Tools

- This would assist in manual vulnerability testing
- SOAP Web Services Fuzzer
- GWT Request Crafter
- JSON Console
- And so on...



Questions

Dan Cornell

dan@denimgroup.com

(210) 572-4400

Sprajax Site: www.owasp.org/index.php/Sprajax

Sprajax Mailing List: owasp-sprajax@lists.owasp.org

Denim Group Website: www.denimgroup.com

Denim Group Blog: denimgroup.typepad.com

