Ajax Security

Douglas Crockford &

Chip Morningstar

Yahoo!

Security Matters

Security Is Hard

Weak Foundations

Inadequate Browser Security Model

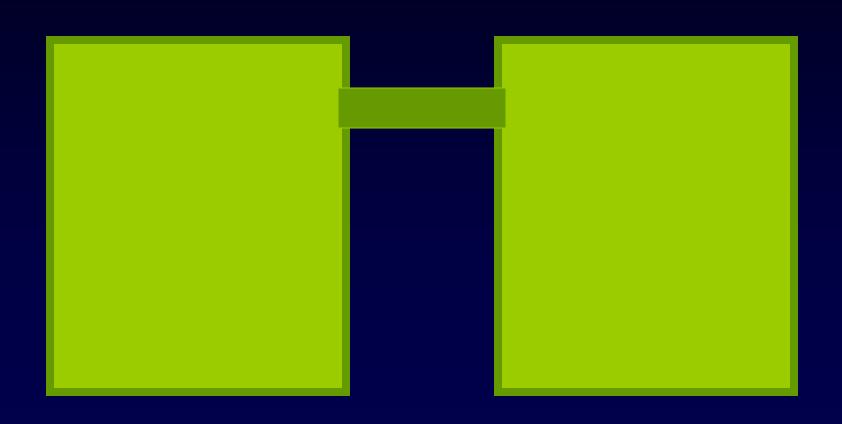
JavaScript is not a secure programming language.

There are very few secure programming languages.

DOM

Document Object Model is insecure.

Trust Boundary



Same Origin Policy

- *Restrictions on access of assets from other sites.
- *No restriction on sending, only on receiving.
- *Bad policy: Prohibits some useful actions, permits some dangerous actions.
- *Boon to idiot IT managers who rely on firewalls instead of authentication.

Circumvention

- *Poorly designed security measures prevent useful activity.
- *Developers are required to produce useful activity.
- *This leads of the circumvention of security mechanisms.
- ***Bad security design makes things** worse.

The web is accidents waiting to happen.

Serious penalties for data leakage.

Web is significantly safer than desktop applications.

But not enough safer.

XSS

- *Cross Site Scripting Attack (misnamed).
- ***Evil JavaScript gets into your page.**
- *All scripts look the same to the browser.

- ***Good hygiene. Use correct encoding.**
- *Server must do white box filtering on all user submitted content.

Be Rigorous

Sloppiness aids the Enemy.

Neatness counts.

CSRF

- ***Cross Site Request Forgery**
- *Cookies are not sufficient to authenticate requests.
- ***Use shared secrets in the request.**

Cookies

- *Cookies were not intended to be an authentication mechanism.
- *Cookies are widely used as an authentication mechanism.

SQL

- ***SQL** injection. Be extremely cautious when building query text from external content.
- ***Remote SQL: Madness.**
- *Never expose SQL to the network.

JSON is Safe and Effective when used correctly.

Like everything else, dangerous when used recklessly.

Script Tag Hack

- ***Scripts (strangely) are exempt** from Same Origin Policy.
- *A dynamic script tag can make a GET request to a server.

```
receiver(jsontext);
```

*Extremely dangerous. It is impossible to assure that the server did not send an evil script.

eval

- ***JSON text is JavaScript, so eval** can turn it into data structures.
- ***Fast, convenient.**

```
\overline{\text{myData}} = \text{eval}('(' + \text{jsontext} + ')');
```

*Dangerous. If the text is not actually JSON, an evil script can execute.

parseJSON

***Use the string.parseJSON method.**

```
myData = jsontext.parseJSON();
```

- ***Evil script will cause a syntax error exception.**
- ***Standard equipment in the next** version of JavaScript.
- *Available now: http://www.json.org/json.js

Server accepts GET requests with cookies

- *Data leakage. A rogue page can send a request to your server that will include your cookies.
- *There are holes in browsers that deliver data regardless of Same Origin Policy.
- *Require POST. Require explicit tokens of authority.

Don't wrap JSON text in comments

***Intended to close a browser hole.**

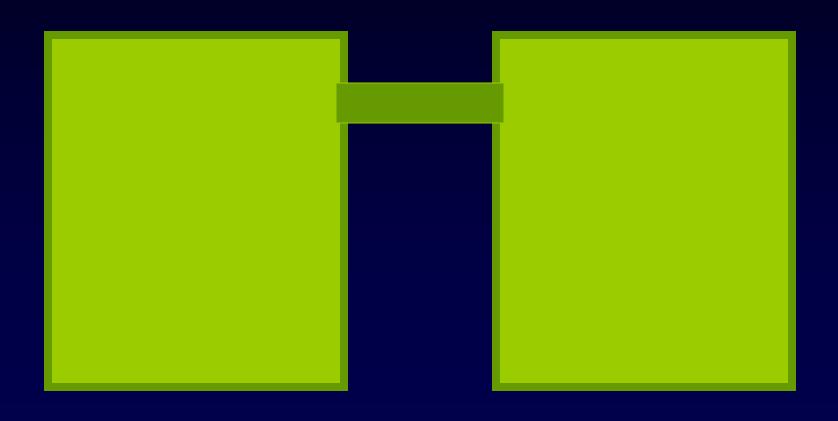
```
/* jsontext */
```

*May open a new hole.

```
"*/ evil(); /*"
```

- ***Security is not obtained by tricks.**
- *Never put data on the wire unless you intend that it be delivered. Do not rely on Same Origin Policy.

The Future



The Caplet Group

- *Good research is being done at IBM, Microsoft, HP, Google, Yahoo, and other places.
- ***A** discovery and messaging system that can safely deliver data across trust boundaries.
- *Connections between pages, iframes, worker pools, desktop widgets, web services.

An example of a secure application framework using today's technology.

Yahoo Ajax Server

- ***Context & session architecture**
- ***Secure session protocol using JSON and HTTP**
- *Why?

Why a new kind of server?

*Some applications go against the grain of the conventional web paradigm

Real-time interactivity (Ajax!)

anything with short-lived session state on the server

Multi-user interactivity

chat, presentations, games, etc.

Server-initiated events

alerts, auctions, process monitoring, games, etc.

*These are all awkward in a standard web server

- ***HTTP-transported message passing scheme**
- * Messages are:
 - Bidirectional Asynchronous Object-to-object
- ***Uses 2 HTTP connections** asymmetrically
 - One to transmit client server messages
 One to poll for server client messages
- ***HTTP** requests DO NOT correspond 1to-1 to object messages!

***Open a session**

```
GET root/connect/randomstuff
```

*Where root identifies the application

```
e.g., http://wingnut.yahoo.com/chat/connect
```

*Reply is JSON containing unguessable session identifier

```
{"sessionid": sessionID }
```

* Send messages to the server

```
POST root/xmit/sessionID/xseqnum
```

- * sessionID from the connect request
- * xseqnum from previous xmit request, or 1 to begin

```
http://moonbat.yahoo.com/chat/xmit/hb5t1fhyku42/3
```

- *** POST body contains one or more messages being sent**
- * Reply contains sequence number for next xmit

```
{"seqnum": newxseqnum }
```

* Post whenever you have something to say to the server

* Poll for messages from the server

GET root/select/sessionID/sseqnum

- * sessionID from the connect request
- * sseqnum from previous select request, or 1 to begin http://wingnut.yahoo.com/chat/select/in5uuf67xjlnogr/47
- Reply contains messages and sequence number for next select

```
{"msgs": [ msg, msg, ... ], "seqnum": newsseqnum }
```

- * Request after reply to connect or previous select
- Client always has a select pending while session is live
- Reply might contain 0 messages (connection heartbeat)

JSON Messaging

Simple convention for encoding object-addressed messages

```
{"to": targetref, "op": verb, params ... }
```

* targetref identifies message target in scope of receiver

Can be simple ("foo") or complex ("user. 47.3699102")

Can be static or dynamic

Can be known & predictable or random & unguessable All up to the application protocol designer

- * verb identifies the operation, params depend on verb

 Standard O-O stuff
- * All messages are unidirectional and asynchronous Never block, never deadlock

Contexts define Applications

- ***YAS** serves contexts containing objects
- ***Clients can enter these contexts**
- *Clients in a context can send messages to the objects in it (and viceversa)
- *The web page whose script initiates a connection contains JavaScript for the client side of the various objects

Multi-user Interactivity

- *Multiple clients can enter a YAS context concurrently
- *Server can fan messages to some or all of the clients in a context
- ***Server can relay messages between clients**

Server-initiated Events

*Autonomous processes running in the server can send messages to clients

- ***So the server just sends a message**
- ***Yes, it's that simple**

What's this got to do with Security?

- ***Our most powerful security tools are modularity and encapsulation**
- *Web paradigm says "abandon encapsulation"

REST dogma actually elevates this to a virtue

- ***YAS** is a scheme to get encapsulation back
- *In the world of Web 2.0, Ajax, mashups, etc. we really need it

Where to keep session state?

* In the browser: cookies, form vars, URLs

Clumsy, Insecure, Limited capacity

Your data is in the hands of the enemy

* In a database

Clumsy, Slow, Inefficient

Reintroduces the bottleneck that motivated a stateless architecture in the first place

In the server's memory

Fast & Easy

Conventional web scaling paradigm says do not do this!

Scale Differently

- ***** Keep session state in RAM on the server
- * Scale by session, not by page
- * Browser just keeps talking to same server
- * Web infrastructure is not optimized for this...
- * ...but it's not very difficult to do
 - Route by session rather than by HTTP GET request
 - Have application page server act like a session-level VIP or HTTP director
 - Browser is already handshaking with server anyway