JavaScript Malware for a Gray Goo Tomorrow!

Billy Hoffman (bhoffman@spidynamics.com)

Lead Researcher, SPI Labs
**A Moment of Clarity**

**JavaScript** – (noun) A client side computer programming language, largely misunderstood by the general public, that can be used to create malicious, cross platform, and self-replicating software.

**Gray Goo** – (noun) A hypothetical end-of-the-world scenario involving nanotechnology in which out-of-control, self-replicating robots consume all matter on Earth, destroying life as we know it.
JavaScript Nastiness Circa 1999

Sorry for the interruption. Will $1001.00 make it up to you? In that case our operators will give you 3 minutes to call us and receive $1001.00 off a 1-week Dream Vacation with a cruise.

Click the Correct Spot and Win $1000! Try Now!

Would you like to make "www.bonzibuddy.com" your homepage?

Catch the Monkey and Win $20!
JavaScript Nastiness Circa 2006

Port Scanning in JavaScript - SPI Dynamics - Mozilla F...
Why JavaScript, why now?

- Why didn’t Web 2.0 happen in 2000?
  - Lack of standards compliant browsers
    - JavaScript implementations all different
    - DOM manipulation/Eventing all different
    - CSS support lacking
  - Lower connection speeds/processing power

Ajax succeeds because it’s cross browser!
Now is the time for JavaScript malware

- Homogenous platform
  - Same browsers
  - Different devices (PC, Sidekick, iPhone, embedded)
- JavaScript is much more powerful
  - OO, extendable: `String.prototype.foo = function() {...}
  - Dynamic code execution
  - RegExs
  - Very rich interface to/from browser/plugins
    - If JavaScript can’t do it, Flash/Java can…
  - Large number of “networking” functions
Current State-of-the-Art
JavaScript Malware
Cross Site Scripting (XSS) And Ajax

- Cross Site Scripting (XSS) is injection of a script (Javascrip or VBScript) into the page that is returned to the user’s browser.
- These scripts get executed by the user’s browser, exposing them to a variety of threats:
  - Session hijacking
  - Information leakage
  - Content manipulation
  - Keylogging/Screen scraping
- With Ajax, XSS can make requests hidden HTTP requests!

Why does this matter?
HTTP Requests

- HTTP requests made by Ajax look identical to requests made by user
  - Headers
  - Statekeeping/Authentication tokens
- Server cannot discern Ajax requests from browser requests!

```plaintext
GET / HTTP/1.1\r\nHost: maps.google.com\r
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv: Accept: text/xml,application/xml,application/xhtml\r
Accept-Language: en-us,en;q=0.5\r
Accept-Encoding: gzip,deflate\r
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r
Keep-Alive: 300\r
Connection: keep-alive\r
Cookie: PREF=ID=86e9ce4c2b9dd60f;FF=4;LD=en;NR=10;
```

```plaintext
GET /mt?n=404&v=w2.7&x=472&y=794&zoom=6 HTTP/1.1\r\nHost: mt2.google.com\r
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv: Accept: image/png,*/*;q=0.5\r
Accept-Language: en-us,en;q=0.5\r
Accept-Encoding: gzip,deflate\r
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r
Keep-Alive: 300\r
Connection: keep-alive\r
Referer: http://maps.google.com/\r
Cookie: PREF=ID=86e9ce4c2b9dd60f;FF=4;LD=en;NR=10;TM=11
```
Ajax Amplifies XSS Attacks

- In other words
  - XSS can make requests for resources
  - Request is hidden from user
  - Happens in background while you are using the computer
  - Browser automatically adds authentication information
  - XSS can read response, send derived requests
  - Server thinks you initiated the request
Ajax Amplifies XSS
Self Propagating XSS

- XSS payload can now autonomously inject itself into pages
- Easily re-inject same host with more XSS
- Can do all this seamlessly (no hard refresh)
- Can send multiple requests using complex HTTP methods to accomplish propagation
Analysis of MySpace.com Virus

- Web virus
- October 2005: Infected 5th largest domain on the Internet
- JavaScript with Ajax
- Attack vector: XSS exploit allowed <SCRIPT> into user’s profile
- Propagation:
  - Used Ajax to inject virus into the user profile of anyone who viewed an infected page
- Payload:
  - Used Ajax to force viewing user to add user “Samy” to their friends list
  - Used Ajax to append “Samy is my hero” to victim’s profile
XSS+Ajax on a Bank
Port Scanning in JavaScript

- JavaScript can make HTTP connections to arbitrary hosts
  - Cannot see the response (Ajax restriction)
    - Not always true... Images, iFrames
  - Can detect if successful
  - Can detect if there was an error
  - Can set timers and see if any event fired

- JavaScript can use load events, error events and timeouts to detect the presence of HTTP servers on arbitrary hosts and ports!
  … even on intranets
Step 1: Implementing Ping with JavaScript

- Use Image object with `onLoad()` and `onError()` events and a timer
- Setting `src` on Image causes an HTTP GET
- Start timer
- If host exists, `onError()` or `onLoad()` will fire
- If host doesn’t exist, timer fires
Step 2: Detecting HTTP content in JavaScript

- Image’s `onError()` fires if its HTTP traffic or not
- To confirm HTTP traffic, use iFrame with `onLoad()` event and a timer
- Set `src` on iFrame
- Start timer
- If host is HTTP server, `onLoad()` fires
- If host is not HTTP server, timer fires
Step 3: Fingerprinting Web Servers in JS

- Fingerprint by requesting images unique to a web server or application
- Use Image object with `onLoad()`
- Send requests for known images
- If image exists, check the dimensions
- If dimensions match, found fingerprint successful
- If not, move to next image
- Can also check for existence of style sheets or JavaScript files
Stealing Browser History

• In the beginning, visited links looked different than unvisited
• This styling was performed by the user agent
• With Cascading Style Sheets (CSS), users could style links
• With JavaScript it is possible to determine the style of any DOM element on the page, including links
• JavaScript + CSS = theft of URL history!
Stealing Browser History

From the W3C Cascading Style Sheet Standard:

Note. It is possible for style sheet authors to abuse the :link and :visited pseudo-classes to determine which sites a user has visited without the user's consent.

(http://www.w3.org/TR/CSS21/selector.html#link-pseudo-classes)
Stealing Browser History

• How it’s done
  – Use JavaScript to dynamically create a new link to any URL
  – Apply a style attribute to the link, defining different styles for :link and :visited
  – Browser automatically renders link with appropriate style
  – Use JavaScript to check style on the link
Stealing Browser History

- Browser history = giant hash table
  - Cannot enumerate through it
  - Can ask it yes/no questions
    - Can perform thousands of look ups a second!
    - Just have to know what questions to ask it… more on this in a minute.
- JavaScript can now detect very specific URLs
- Sometimes URLs are different for everyone
  - In URL session state/authentication tokens
What else can we do?
Stealing Search Engine Queries?

- Has the user been to the results URL of a search engine?
- Hmmm… Can we steal search engine queries?
- Research shows there are a few problems
Stealing Search Engine Queries?

Problem 1: Results page for search query can have different URLs
Stealing Search Engine Queries?

- Problem 2: search query letter case produces different URLs

- Problem 3: word order of query produces different URLs
Stealing Search Engine Queries?

What if we solve all the problems by brute force?

Given query Q with x number of words:
There are $2^x$ combinations where first letter is upper or lowercase
There are $x!$ ways to order search words
If there are y number of unique result URLs

$$\text{Num URLs} = (2^x \times x!) \times y$$

To see if user searched Google for some variation of “secure handshake Diffie Hellman”

$$(2^4 \times 4!) \times 3 = 1152 \text{ URLs!}$$

…and what if they don’t use Google?
Stealing Search Engine Queries!!!

• Don’t Panic!
• We can do thousands of look ups a second!
• SearchTheft.js
  – Detects what search engines are used
  – Tries all combinations of letter case and word order
  – Reports if user has searched for a term

Demo of SearchTheft.js

http://www.spidynamics.com
What Queries to Check for?

- How do you know what queries to check for?
- User supplied
  - `billysbooks.com` can see if you also searched for something on a competitor’s site
- Precomputed List
  - FBI can check for common kiddie porn queries, JavaScript automatically reports you!
  - DoubleClick could use a list of terms relevant to the topics for each site that uses their ads
Future JavaScript Malware

- Cross Domain XSS Web worms
  - MySpace.com and Yamanner never hopped hosts
  - Google’s AJAX Search API
    - Create a SCRIPT Tag with the SRC pointing to Google
    - The query string of SCR contains your search query
    - Google returns JavaScript containing the results of query
      - XSS can now call Google to find other vulnerable hosts
      - XSS can then use blind GETs and POSTs to infect these new hosts
Future JavaScript Malware
Steps towards a JavaScript web crawler

- HTML can open content from Site2.com
- JavaScript from Site1.com cannot access the content!
- This is the Same Origin Policy!
  - Basis of entire JavaScript security model
- Prohibited from accessing each other's content
Google Translate to the rescue!

- Google Translate (GT) can fetch pages from anywhere (i.e., proxy)
- Content is in GT’s domain
- Allows content from separate sites to be in the same domain!
Jikto: JavaScript Web Vuln Scanner

- Written entirely in JavaScript (~875 lines)
- Can crawl and audit third party site
- Results can be displayed or sent to a different user
- Based heavily on the work of pdp’s crawler (http://gnucitizen.org)
  - He used iframes, cross iframe communication
  - Nifty proof of concept but not viewed as realistic
    - Slow! (timers + iframe onloads = bottleneck)
- Ajax >>>>>= iframes
  - Can we stop this silly “Ajax doesn’t change security bit”
iFrames vs. XmlHttpRequest

Both iFrames and XmlHttpRequest can be used to fetch content.

- iFrames are a dirty hack!
- Hooks `onload` event
- iFrame’s `onload` doesn’t fire until entire page has loaded.
- Normally an order of magnitude slower.
How Jikto works

• Our JavaScript needs to be in same domain as website we are scanning
• We load an iframe to Google Translate (GT), and point GT to site with Jikto code
• Jikto code is now in GT’s domain, so it can use Ajax to tell GT to get any public page from any site. Ajax *much* faster than iFrames here!
• Jikto can analyze response, send derived requests, make attacks, etc.
Jikto Pros and Cons

Pros
- Very, very fast
- No application install required
- Cross browser
- Cross platform
- For attackers:
  - Now can find exploits!
  - Weaponizable/XSS-able
  - XSS + Jikto + Social Networking = Botnets

Cons
- Proxy can limit you
  - Does it forward HTTP headers?
  - Cookies?
  - Thru POSTs or lame?
  - Rate limiting?
- `XmlHttpRequest` auto follow 3xx with no input
More About Jikto

- Requests a page from Request Queue
- Processes response
  - Scraps out hyperlinks
  - Creates Requests from FORM tags
  - If Requests was an attack…
    - Score attack according to RegEx
  - Pass Response to attack library
    - Generates new attack requests for the Request
    - Currently some Backup file checks and XSS/SQL checks
More About Jikto

• ~875 lines of JavaScript (heavily commented)
  – ~500 lines of parsing code
    • Url parsing, resolving relative links, extracting, etc
    • Form parsing for inputs, HTTP methods, etc
  – ~220 HTTP glue code
    • XmlHttpRequest, proxy management
    • Request and Response objects
  – ~20 lines GUI interface
  – ~40 lines attack library
  – Misc stuff: debugging, rot13, global variables
<table>
<thead>
<tr>
<th>Method</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td><a href="http://zero.webappsecurity.com:80/admin/help.cgi.bak">http://zero.webappsecurity.com:80/admin/help.cgi.bak</a></td>
</tr>
<tr>
<td>GET</td>
<td><a href="http://zero.webappsecurity.com:80/cookieTest/">http://zero.webappsecurity.com:80/cookieTest/</a></td>
</tr>
<tr>
<td>GET</td>
<td><a href="http://zero.webappsecurity.com:80/pindex.asp">http://zero.webappsecurity.com:80/pindex.asp</a></td>
</tr>
<tr>
<td>GET</td>
<td><a href="http://zero.webappsecurity.com:80/pindex.asp.bak">http://zero.webappsecurity.com:80/pindex.asp.bak</a></td>
</tr>
<tr>
<td>GET</td>
<td><a href="http://zero.webappsecurity.com:80/">http://zero.webappsecurity.com:80/</a></td>
</tr>
<tr>
<td>POST</td>
<td><a href="http://zero.webappsecurity.com:80/login1.asp">http://zero.webappsecurity.com:80/login1.asp</a></td>
</tr>
<tr>
<td>POST</td>
<td><a href="http://zero.webappsecurity.com:80/rootlogin.asp">http://zero.webappsecurity.com:80/rootlogin.asp</a></td>
</tr>
<tr>
<td>POST</td>
<td><a href="http://zero.webappsecurity.com:80/pcomboindex.asp">http://zero.webappsecurity.com:80/pcomboindex.asp</a></td>
</tr>
<tr>
<td>POST</td>
<td><a href="http://zero.webappsecurity.com:80/acctxferconfirm.asp">http://zero.webappsecurity.com:80/acctxferconfirm.asp</a></td>
</tr>
</tbody>
</table>

### Vulnerabilities

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Severity</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross Site Scripting</td>
<td>100</td>
<td><a href="http://zero.webappsecurity.com:80/plink.asp?a=%3Cscript%3Ealert%28%27xss%27%29%3C/script%3E&amp;c=%3Cscript%3Ealert%28">http://zero.webappsecurity.com:80/plink.asp?a=%3Cscript%3Ealert%28%27xss%27%29%3C/script%3E&amp;c=%3Cscript%3Ealert%28</a></td>
</tr>
<tr>
<td>Cross Site Scripting</td>
<td>100</td>
<td><a href="http://zero.webappsecurity.com:80/pformresults.asp?btHidden=%3Cscript%3Ealert%28%27xss%27%29%3C/script%3E&amp;cbCode=">http://zero.webappsecurity.com:80/pformresults.asp?btHidden=%3Cscript%3Ealert%28%27xss%27%29%3C/script%3E&amp;cbCode=</a></td>
</tr>
<tr>
<td>Backup File Detected!</td>
<td>50</td>
<td><a href="http://zero.webappsecurity.com:80/default.asp.bak">http://zero.webappsecurity.com:80/default.asp.bak</a></td>
</tr>
<tr>
<td>Backup File Detected!</td>
<td>50</td>
<td><a href="http://zero.webappsecurity.com:80/admin/help.cgi.bak">http://zero.webappsecurity.com:80/admin/help.cgi.bak</a></td>
</tr>
</tbody>
</table>
Jikto Architecture

- Abstracted into 4 parts
  - Add new proxies
  - Add new attacks
- Not all 4 parts on same machine!
  - Controller and Reporting can be on remote host
  - Allows distributed Requestors and Analyzers controlled by central system

==Reporting/UI==
receives events, UI, stores for later analysis,

==Controller==
decides what to do new attacks, fuzzing, etc

==Analyzer==
parses links/forms scores attacks

==HTTP Requestor==
Handles Proxy Ajax/Requests
Future Advances for Jikto

- Exploitation
  - Check XSS attack execution with browser’s JavaScript interpreter!
  - Wormable?
    - Yep
    - I can now find and confirm XSS vulns in other sites

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'tbl_products' to a column of data type int.

/product_detail.asp, line 170
JavaScript Malware for a Gray Goo Tomorrow!

Billy Hoffman (bhoffman@spidynamics.com)

Lead Researcher, SPI Labs