

JavaScript Malware for a Gray Goo Tomorrow!

Billy Hoffman (bhoffman@spidynamics.com)

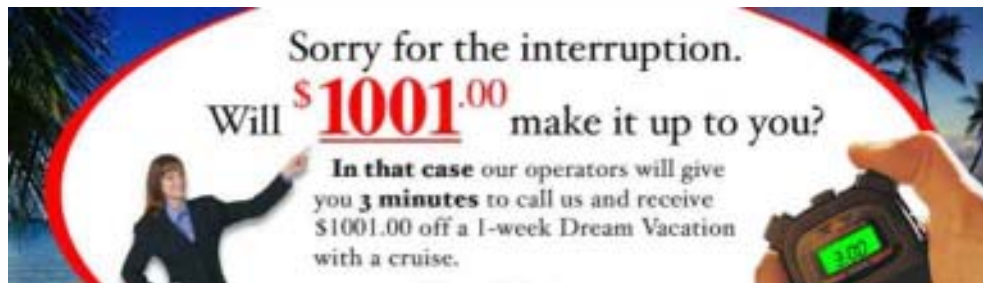
Lead Researcher, SPI Labs

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



https://a.tribalfusion.com - Netflix - Microsoft Internet Explorer

CLICK THE CORRECT SPOT AND WIN \$1000! TRY NOW!

2 Rentals

Confirm

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

OK Cancel

Click Here

CATCH THE MONKEY AND WIN \$20!

0+ titles

Internet

JavaScript Nastiness Circa 2006

Port Scanning in JavaScript - SPI Dynamics - Mozilla F...

File Edit View Go Bookmarks Tools Help

SPI DYNAMICS

Port Scanning with Java

IP To Start:

IP To End:

IP	Host Exists?	Webserver
172.16.10.100	true	Microsoft IIS
172.16.10.101	false	NA
172.16.10.102	true	Apache
172.16.10.103	true	Microsoft IIS
172.16.10.104	true	Microsoft IIS
172.16.10.105	true	none
172.16.10.106	false	NA

Done

MySpace.com

I graduated in:

State: Year:

Springfield High (100%) Martin Luther King (100%) Trinity High School (100%) Mount VERNON High School (100%)

Home Friends Groups Events Mail Blog Favorites Friends Groups Events Local Area Friends

100% Add Friend

Mail Center

Friend Request Manager

Listing 1-10 of 503664

Date	From	Confirmation
Oct 4, 2005 10:22 AM		PLEASE DON'T FEEL CHANGES Like the Loveable! You'd like to be your friend!
Oct 4, 2005 10:22 AM		MAD PHOTOGRAPH SKILLS Always!! wants to be your friend!
Oct 4, 2005 10:20 AM		Approve Deny Send Message

History from whitehatsec.com

not visited	http://login.yahoo.com/
visited	http://mail.google.com/
visited	http://mail.yahoo.com/
visited	http://my.yahoo.com/
visited	http://slashdot.org/
not visited	http://www.amazon.com/
not visited	http://www.aol.com/
not visited	http://www.bankofamerica.com/
not visited	http://www.bankone.com/
visited	http://www.blackhat.com/



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!**



This site is best viewed with Internet Explorer 5.0
or higher. The optimal screen size is 1024X768 dpi.

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 (Javascript or VBScript) into the page that is returned to the user's browser
- These scripts gets 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!

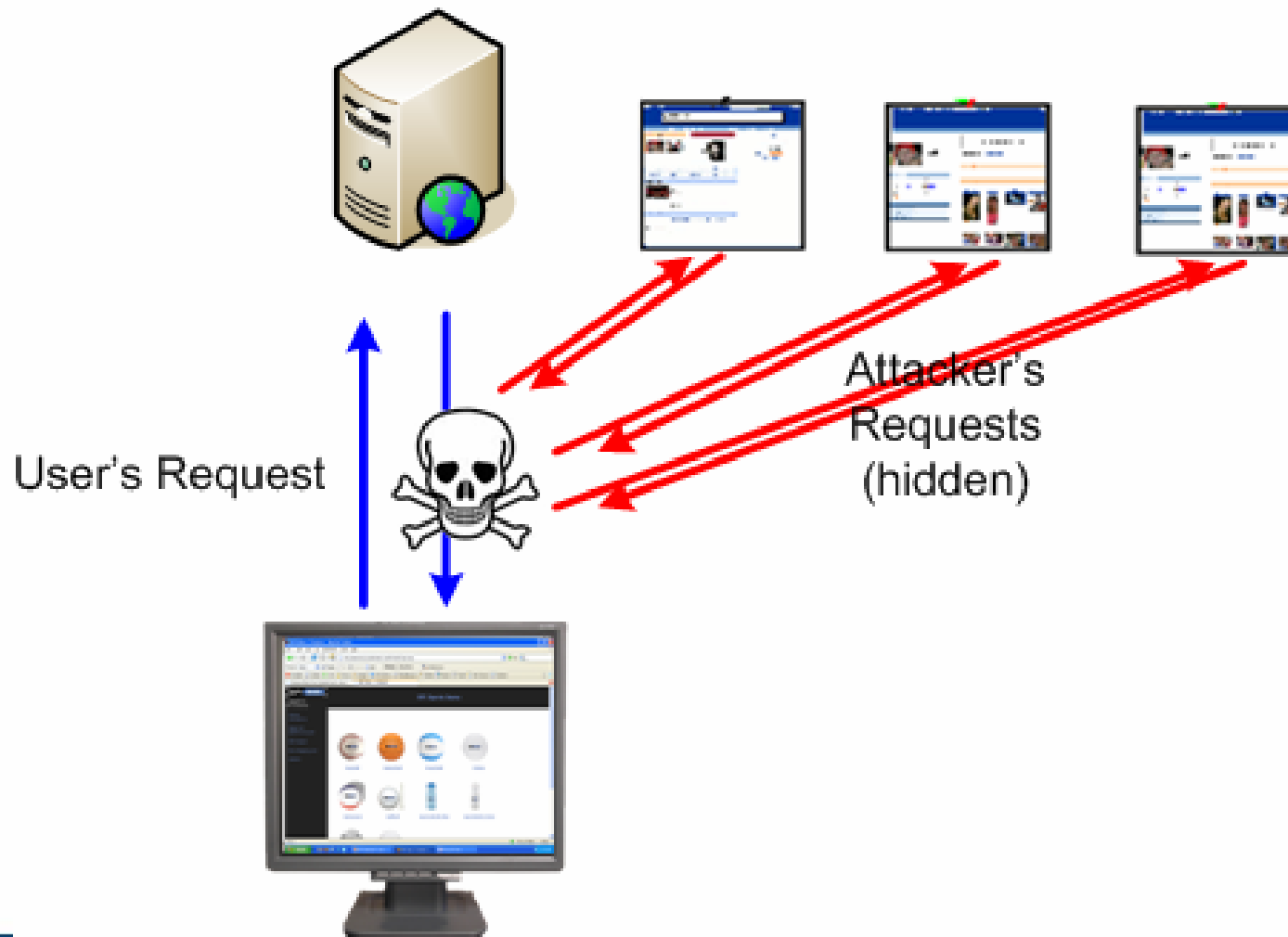
```
GET / HTTP/1.1\r\n
Host: maps.google.com\r\n
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US\r\n
Accept: text/xml,application/xml,application/xhtml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive: 300\r\n
Connection: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

GET /mt?n=404&v=w2.7&x=472&y=794&zoom=6 HTTP/1.1\r\n
Host: mt2.google.com\r\n
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:
Accept: image/png,*/*;q=0.5\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive: 300\r\n
Connection: keep-alive\r\n
Referer: http://maps.google.com/\r\n
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



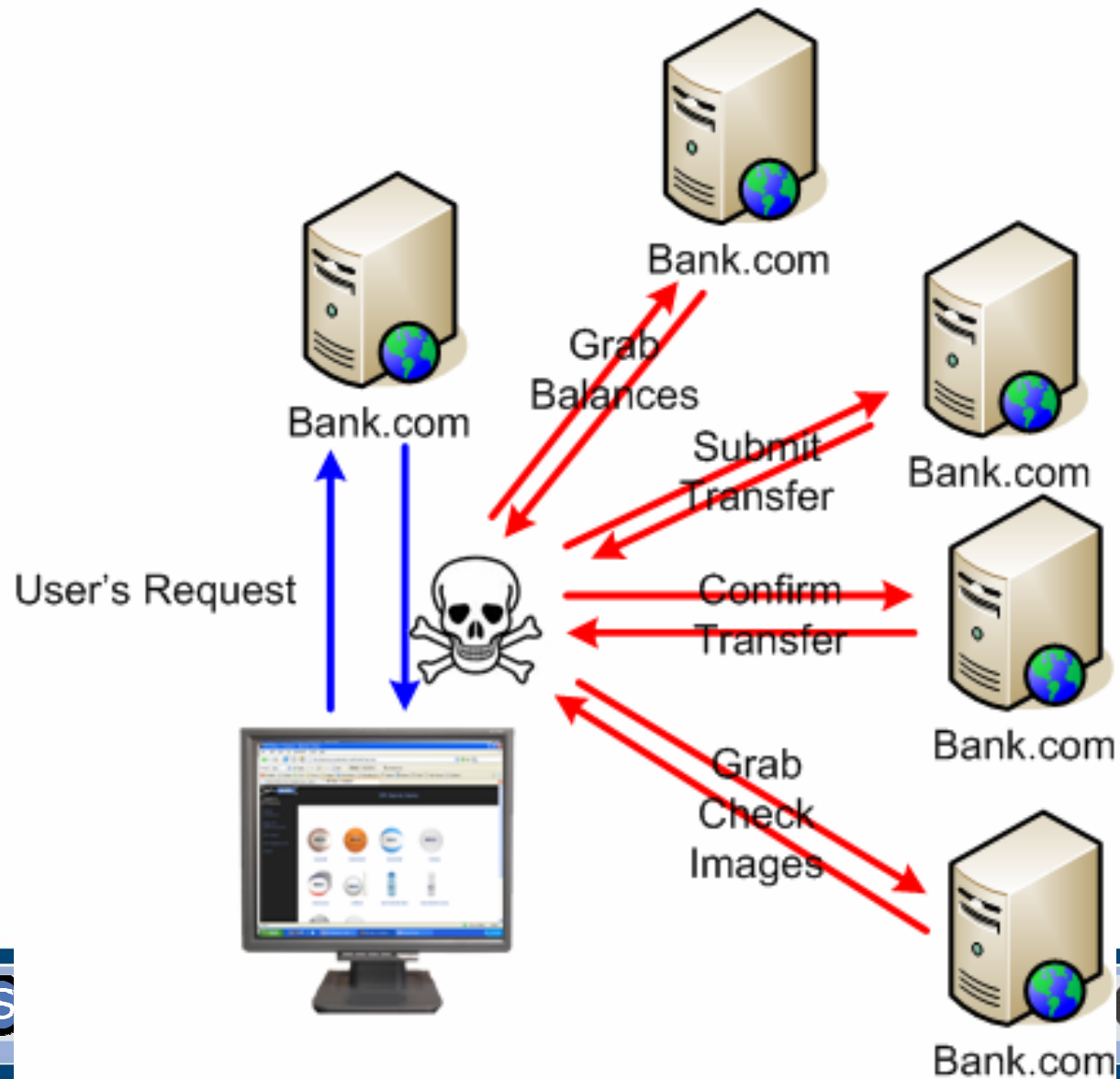
Self Propagating XSS

- 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

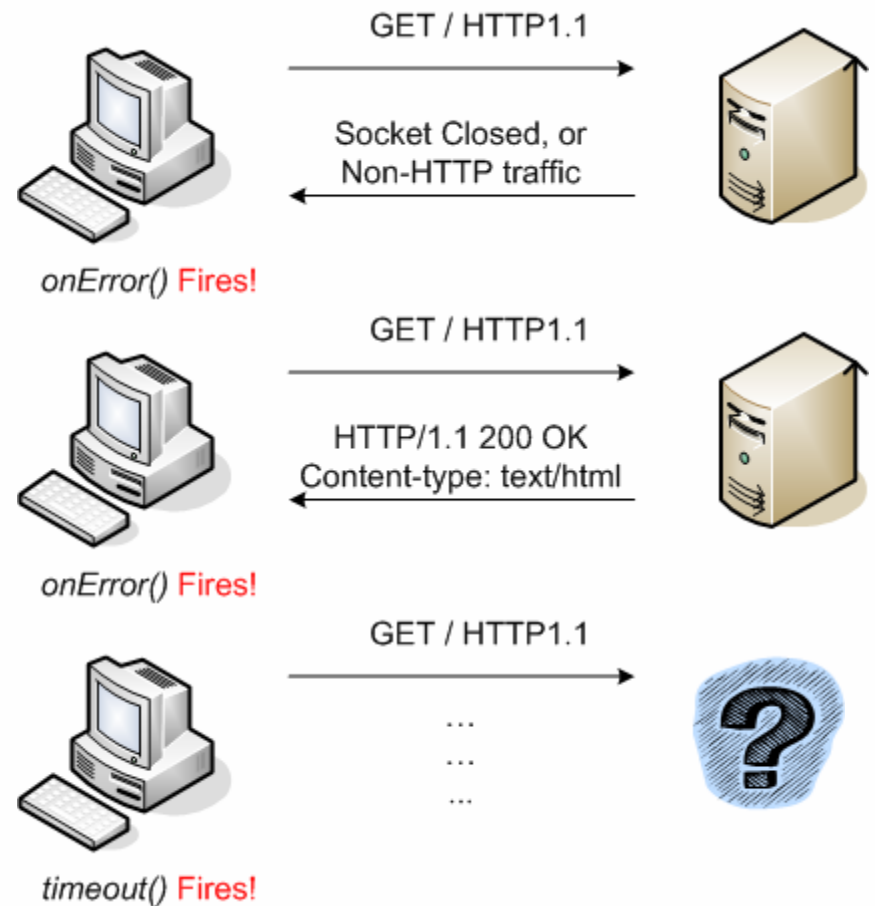


Port Scanning in JavaScript

- 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

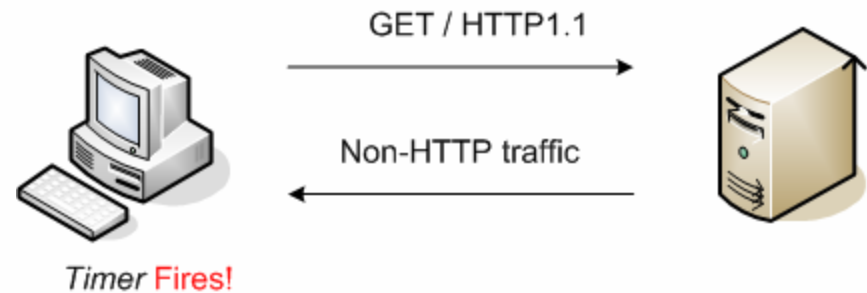
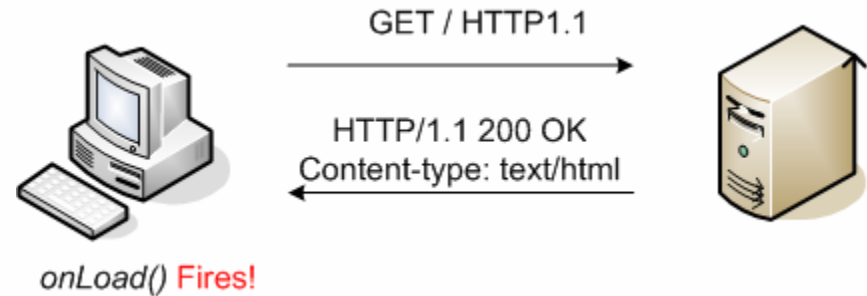
Step 1: Implementing Ping with JavaScript

- 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!

Visited Unvisited

Visited Unvisited

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

t/register.php?PHPSESSION=54183aeacfa6ddf37ab3f59173f41b32

What else can we do?

<http://www.google.com/search?hl=en&q=Acidus&btnG=Google+Search>

<http://www.google.com/search?hl=en&q=SPI+Dynamics&btnG=Google+Search>

<http://www.google.com/search?hl=en&q=free+porn&btnG=Google+Search>

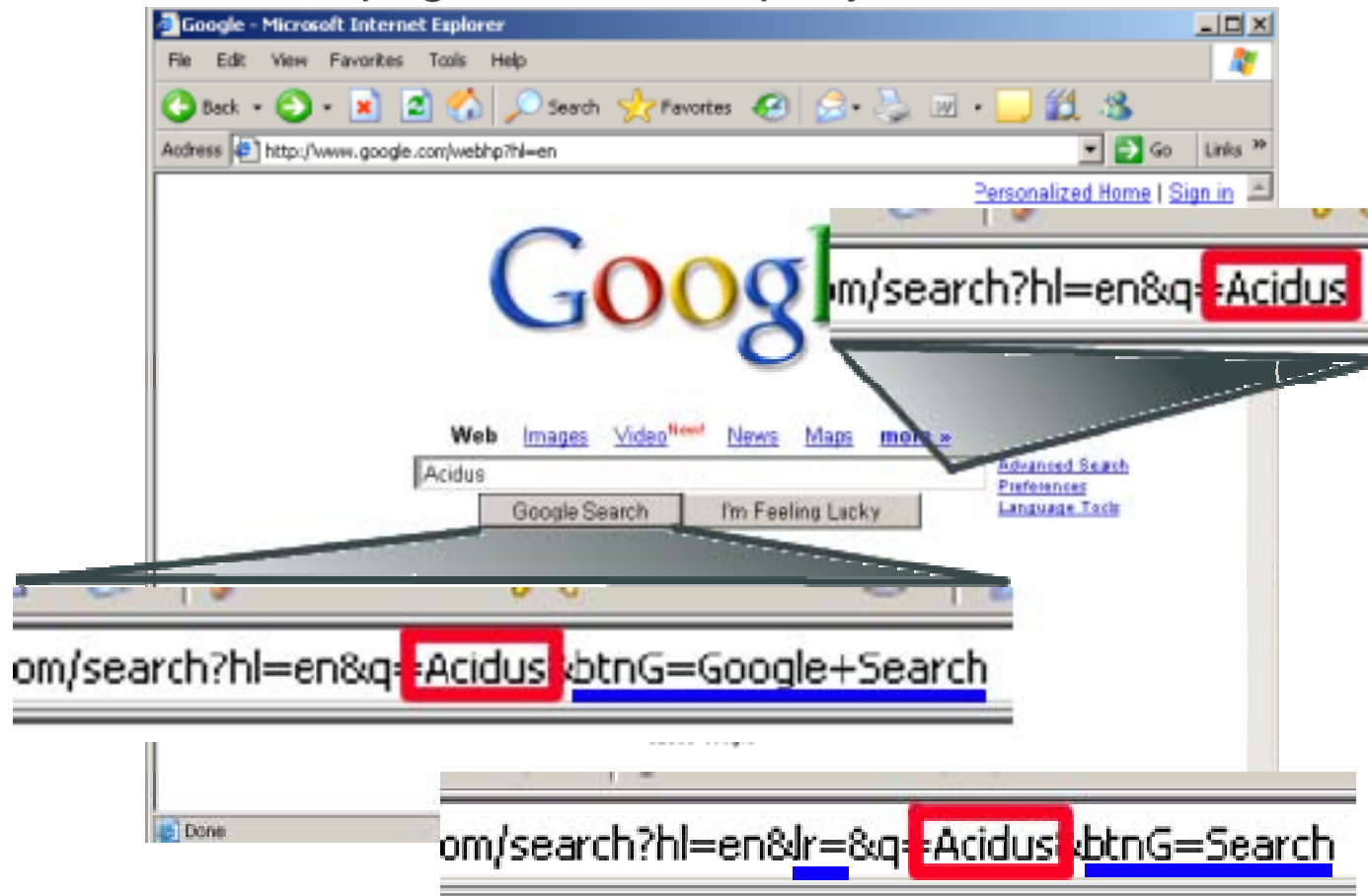
<http://www.google.com/search?hl=en&q=Snakes+on+a+plane&btnG=Google+Search>

Stealing Search Engine Queries?

- Has the user been to the results URL of a search engine?
- Hmm... Can we steal search engine queries?
- Research shows there are a few problems

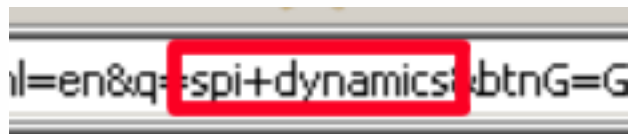
Stealing Search Engine Queries?

Problem 1: Results page for search query can have different URLs



Stealing Search Engine Queries?

- Problem 2: search query letter case produces different URLs




hl=en&q=**spi+dynamics**&btnG=G

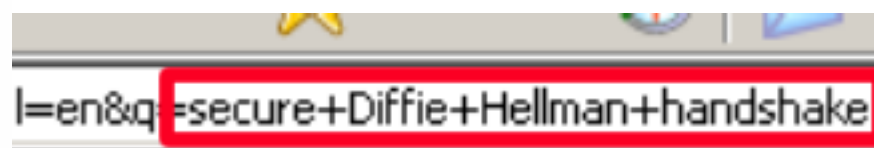


hl=en&q=**SPI+Dynamics**&btnG=G

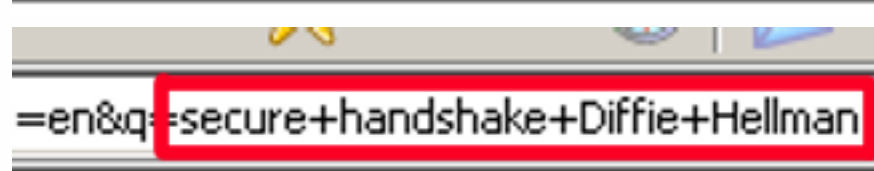
- Problem 3: word order of query produces different URLs



hl=en&q=**Diffie+Hellman+secure+handshake**



hl=en&q=**secure+Diffie+Hellman+handshake**



hl=en&q=**secure+handshake+Diffie+Hellman**

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

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

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

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

...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>

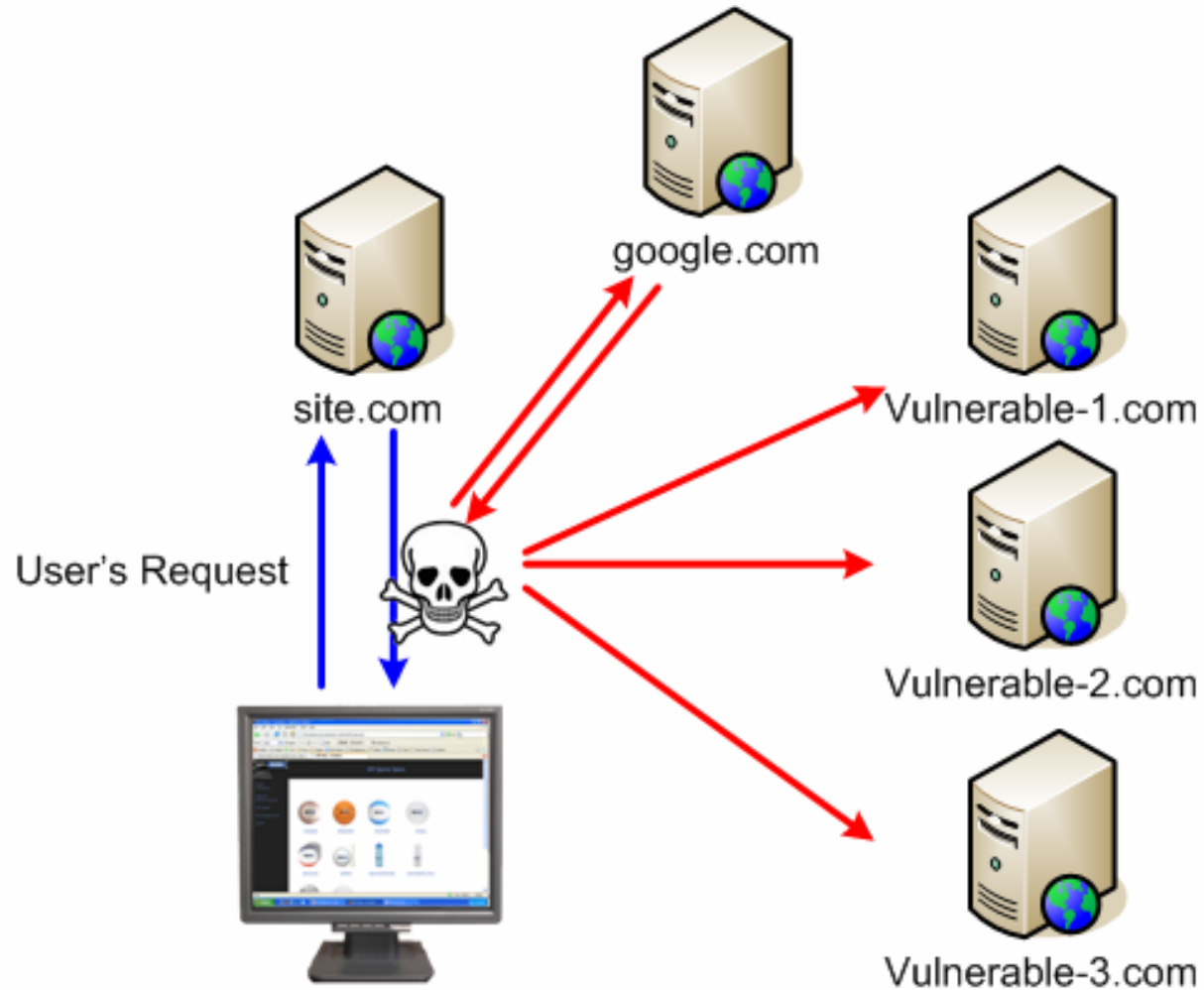
What Queries to Check for?

- 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

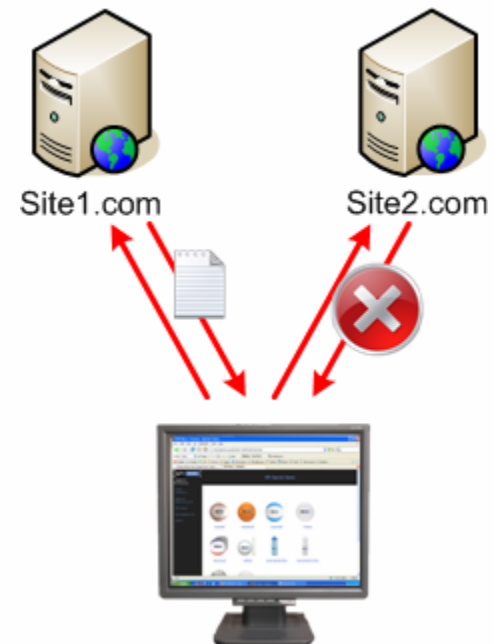
- Cross Domain XSS Web worms
 - MySpace.com and Yamanner never hopped hosts
 - Google's AJAX Search API
 - Create a SCRIPT Tag with the SRC pointing to Google
 - The query string of SCR contains your search query
 - Google returns JavaScript containing the results of query
 - XSS can now call Google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

Future JavaScript Malware



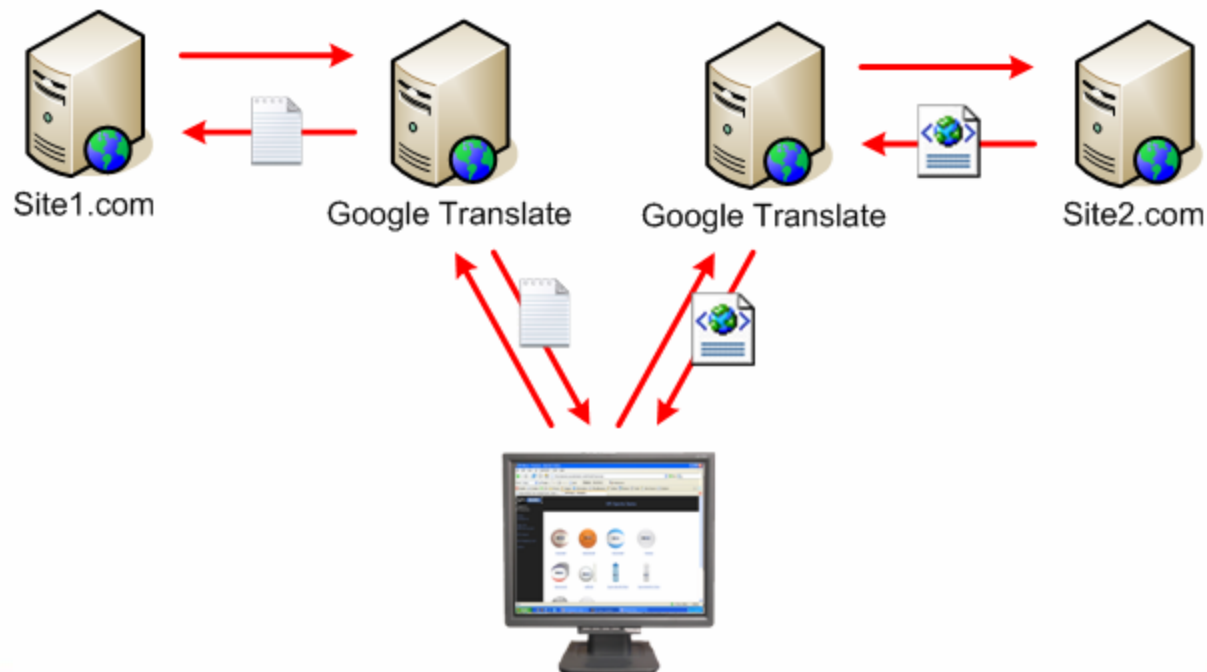
Steps towards a JavaScript web crawler

- 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 others content



Google Translate to the rescue!

- Google Translate (GT) can fetch pages from anywhere (ie, proxy)
- Content is in GT's domain
- Allows content from separates sites to be in the same domain!

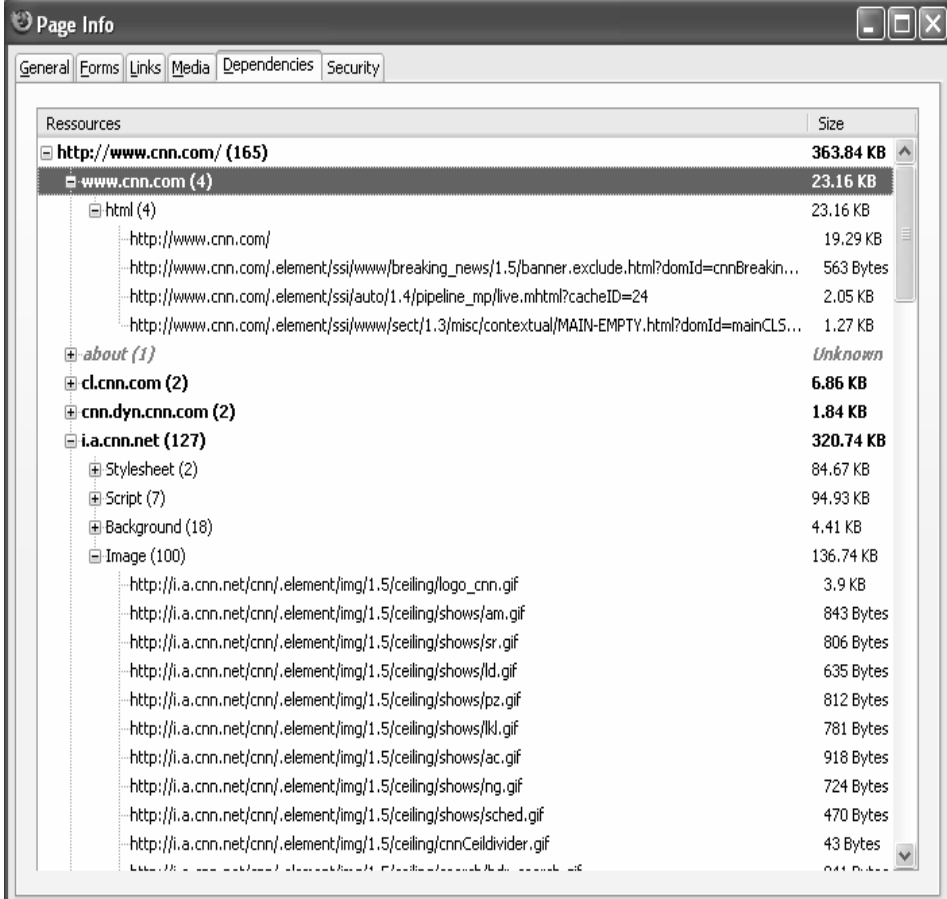


Jikto: JavaScript Web Vuln Scanner

- 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.



The screenshot shows a 'Page Info' window with tabs for General, Forms, Links, Media, Dependencies, and Security. The 'Resources' tab is active, displaying a tree view of resources loaded from various domains. The total size of resources is 363.84 KB. The largest resource is from i.a.cnn.net (127 resources, 320.74 KB total). Other significant resources include www.cnn.com (4 resources, 23.16 KB total) and cl.cnn.com (2 resources, 6.86 KB total).

Resource	Size
http://www.cnn.com/ (165)	363.84 KB
www.cnn.com (4)	23.16 KB
html (4)	23.16 KB
http://www.cnn.com/	19.29 KB
http://www.cnn.com/element/ssi/www/breaking_news/1.5/banner.exclude.html?domId=cnnBreakin...	563 Bytes
http://www.cnn.com/element/ssi/auto/1.4/pipeline_mp/live.mhtml?cacheID=24	2.05 KB
http://www.cnn.com/element/ssi/www/sect/1.3/misc/contextual/MAIN-EMPTY.html?domId=mainCLS...	1.27 KB
about (1)	Unknown
cl.cnn.com (2)	6.86 KB
cnn.dyn.cnn.com (2)	1.84 KB
i.a.cnn.net (127)	320.74 KB
Stylesheet (2)	84.67 KB
Script (7)	94.93 KB
Background (18)	4.41 KB
Image (100)	136.74 KB
http://i.a.cnn.net/cnn/element/img/1.5/ceiling/logo_cnn.gif	3.9 KB
http://i.a.cnn.net/cnn/element/img/1.5/ceiling/shows/am.gif	843 Bytes
http://i.a.cnn.net/cnn/element/img/1.5/ceiling/shows/sr.gif	806 Bytes
http://i.a.cnn.net/cnn/element/img/1.5/ceiling/shows/ld.gif	635 Bytes
http://i.a.cnn.net/cnn/element/img/1.5/ceiling/shows/pz.gif	812 Bytes
http://i.a.cnn.net/cnn/element/img/1.5/ceiling/shows/lk.gif	781 Bytes
http://i.a.cnn.net/cnn/element/img/1.5/ceiling/shows/ac.gif	918 Bytes
http://i.a.cnn.net/cnn/element/img/1.5/ceiling/shows/ng.gif	724 Bytes
http://i.a.cnn.net/cnn/element/img/1.5/ceiling/shows/sched.gif	470 Bytes
http://i.a.cnn.net/cnn/element/img/1.5/ceiling/cnnCeildivider.gif	43 Bytes
http://i.a.cnn.net/cnn/element/img/1.5/ceiling/shows/td.gif	843 Bytes

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

GET	http://zero.webappsecurity.com:80/admin/help.cgi.bak
GET	http://zero.webappsecurity.com:80/aspnet.aspx
GET	http://zero.webappsecurity.com:80/aspnet.aspx.bak
GET	http://zero.webappsecurity.com:80/cookiestest/
GET	http://zero.webappsecurity.com:80/pindex.asp
GET	http://zero.webappsecurity.com:80/pindex.asp.bak
GET	http://zero.webappsecurity.com:80/
POST	http://zero.webappsecurity.com:80/login1.asp
POST	http://zero.webappsecurity.com:80/rootlogin.asp
POST	http://zero.webappsecurity.com:80/pcomboindex.asp
POST	http://zero.webappsecurity.com:80/acctxferconfirm.asp

Vulnerability	Severity	Url
Cross Site Scripting	100	http://zero.webappsecurity.com:80/plink.asp?a=%3Cscript%3Ealert%28%27xss%27%29%3C/script%3E&c=%3Cscript%3Ealert
Cross Site Scripting	100	http://zero.webappsecurity.com:80/pformresults.asp?txtHidden=%3Cscript%3Ealert%28%27xss%27%29%3C/script%3E&dbCor
Backup File Detected!	50	http://zero.webappsecurity.com:80/default.asp.bak
Backup File Detected!	50	http://zero.webappsecurity.com:80/linking/link1/link2/index.htm.bak
Backup File Detected!	50	http://zero.webappsecurity.com:80/admin/help.cgi.bak
Backup File Detected!	50	http://zero.webappsecurity.com:80/aspnet.aspx.bak

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

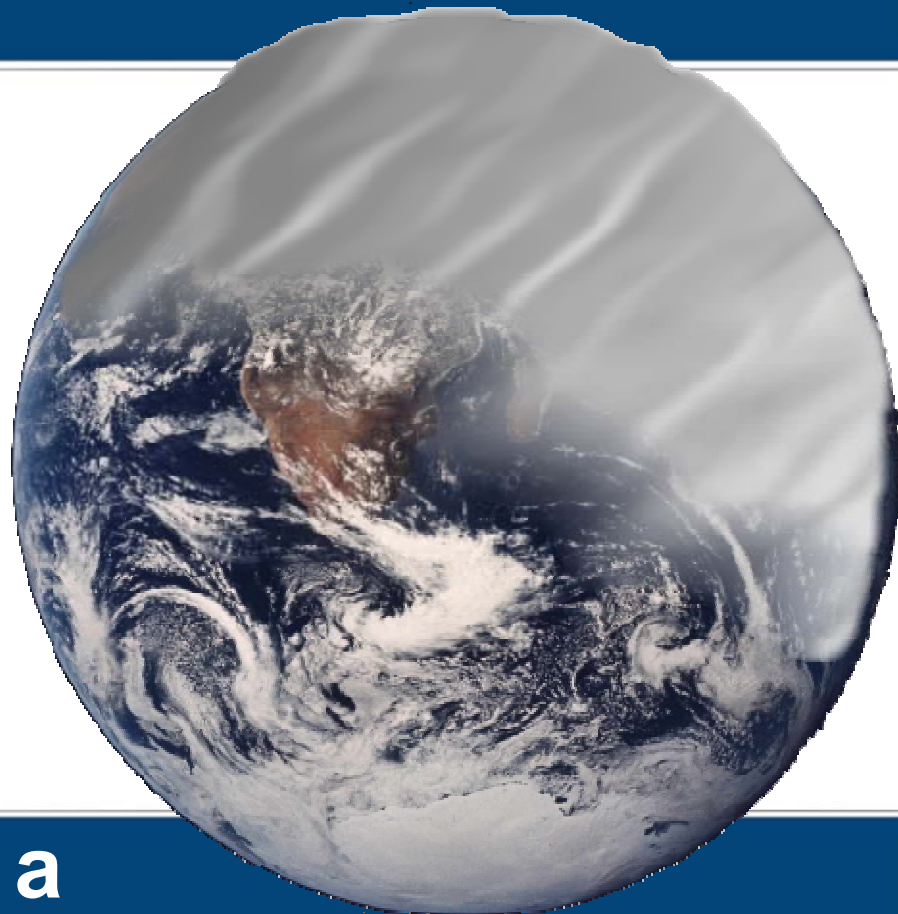
Future Advances for Jikto

- 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