The future is here…

(Sometimes I'd rather live in the past)

The Past: Javascript same domain origin policy.

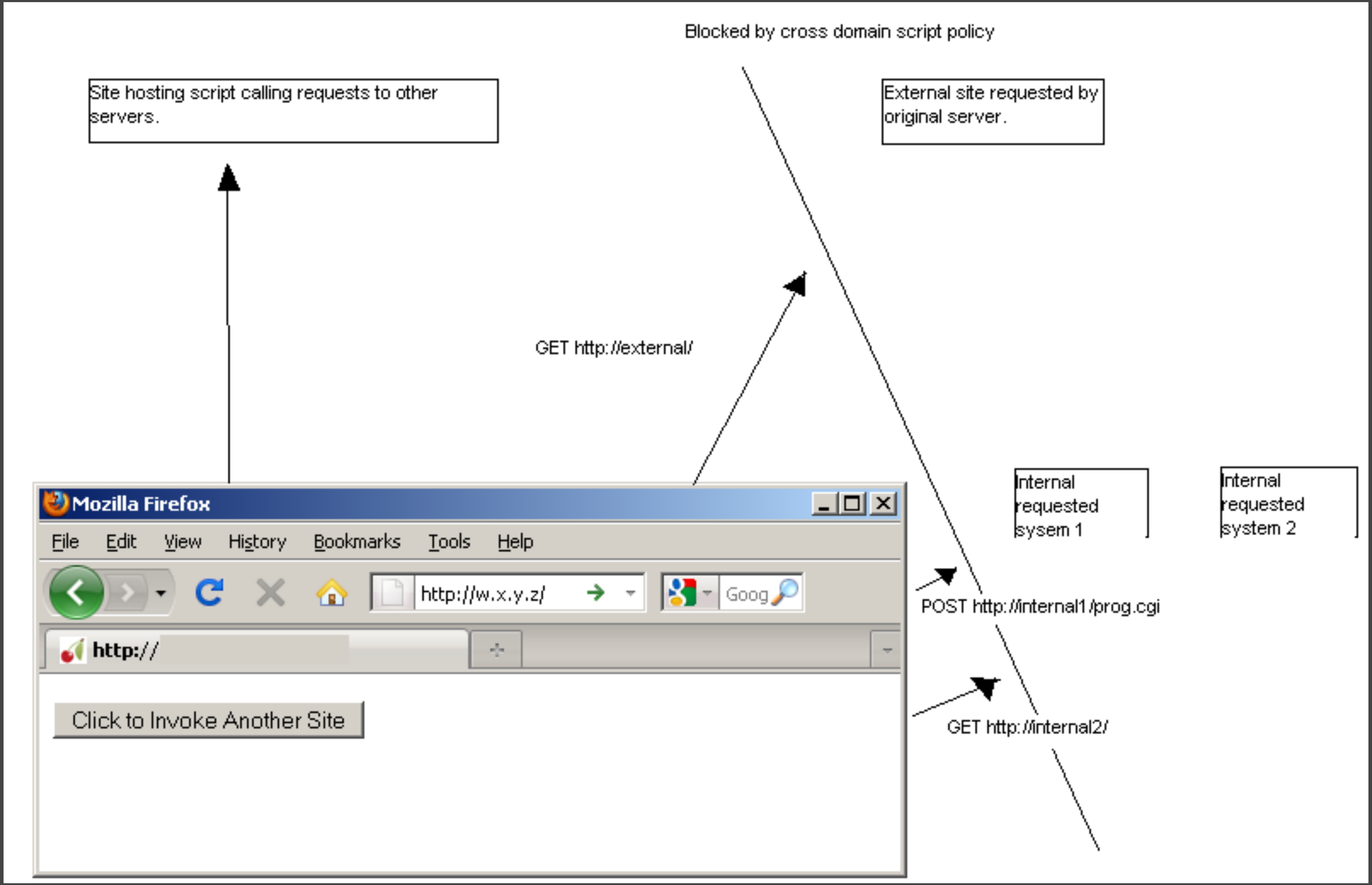
Until recently there was one very small, but important protection built into web browsers for javascript.

The same domain origin policy meant that javascript run from one site could not speak to machines within another domain.

Prevented a scripts from one domain from calling other sites.

In the past few years however w3c decided to "improve" upon this specification by adding new and exciting extensions to the mix...

The same domain restriction


Blocked by cross domain script policy
Site hosting script calling requests to other servers.
External site requested by original server.
GET http://external/
Internal requested sysem 1
Internal requested system 2
POST http://internal1/prog.cgi
GET http://internal2/
Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://w.x.y.z/
Goog
http://
Click to Invoke Another Site

The future: "Access-Control-Allow-Origin"

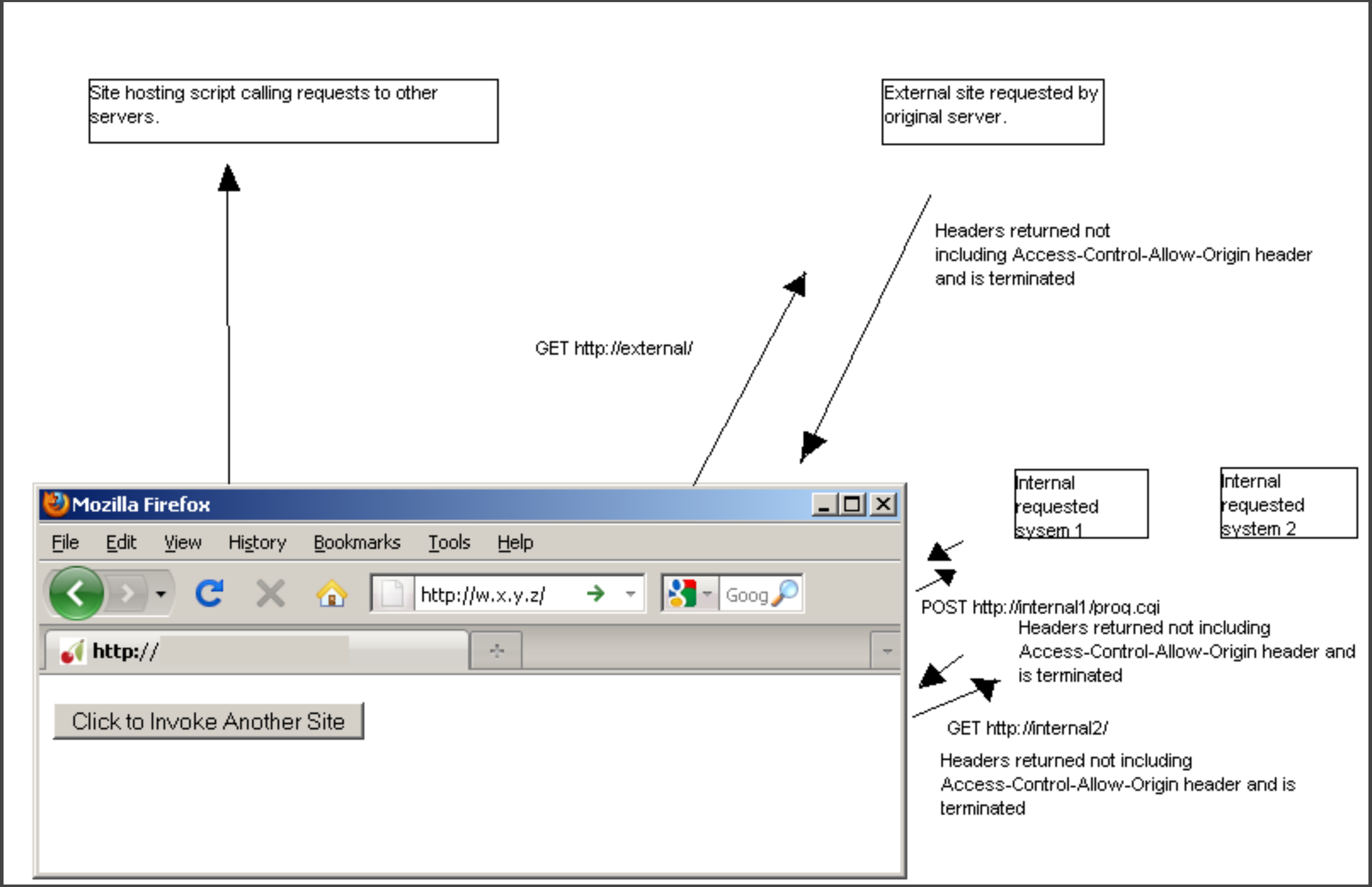
* In 2008 the world wide web consortium (w3c) added a new set of extensions to provide access control for cross domain XMLHTTPRequests (typically AJAX).

* These controls provide for access checks by the browser for all requests of this type made to any system outside of the domain scope of the originating javascript.
* When the second server has a tag including the original domain of the javascript the browser allows the requests to go forward. The headers for this look like:

```
GET /
Origin: http://domain.tld
Referrer: http://domain.tld/index.html
User-Agent: Mozilla 5
```

```
OK 200
Access-Control-Allow-Origin: *
User-Agent: Mozilla 5
```

It works in most browsers but very few sites support it


Site hosting script calling requests to other servers.
External site requested by original server.
Headers returned not including Access-Control-Allow-Origin header and is terminated
GET http://external/
Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://w.x.y.z/
Goog
http://
Click to Invoke Another Site
Internal requested sysem 1
Internal requested system 2
POST http://internal1/prog.cgi
Headers returned not including Access-Control-Allow-Origin header and is terminated
GET http://internal2/
Headers returned not including Access-Control-Allow-Origin header and is terminated

The fun starts when the request is aborted

* For each request successful or otherwise an event is fired on the XMLHTTPRequest object.

* This OnReadyStateChange occurs at least twice on successful requests and at least once on aborted XMLHTTPRequests.

* When a request is aborted due to lacking the header a supporting browser triggers this immediately.

* When a rejection or an unreachable event occurs the event is only fired off when the request times out.

* This means you can use a setTimeout handler to trigger the abort before the timeout actually occurs.

This results in...

* When an abort occurs prior to the setTimeout() occurring we know that xmlhttprequest sent to the second server is highly likely to be listening for http style connections.

* When the setTimeout() occurs prior to the abort the target is either not reachable by the browser or not listening on that port for http style connections.

* Finally because this is being initiated by another domain's javascript the results of these aborts can be issued back to the original requesting server along with the ip address of the machine that appears to be listening on the port in question.

* In effect you can use any browser supporting this method to scan arbitrary networks for listening systems upon executing the javascript.

In some cases it is even worse

* While current releases of safari, firefox, chrome as well as other webkit based browsers are vulnerable to this one of them has an even larger hole.

* When Firefox aborts a request due to lack of of the Access-Control-Allow-Origin header containing the origin header from the original javascript it leaks additional information.

* Firefox in versions 3.5+ on all platforms also issues allows retrieval of the "statusText" value from the xmlhttprequest by the javascript upon abort.

* This statusText includes "OK," "Forbidden" and "Unauthorized meaning that it can be used to scan individual sites for certain forms of content.

Proof of concept tool

* I wrote a simple tool in python called xmlhttpscanner.py that can be used to scan class C address ranges for listening servers, but is easily modified for other purposes.

* It takes roughly a minute to a minute and a half to scan a class C, but could be tuned to work faster by increasing the number of simultaneous requests.

Mitigation

* Use a non-vulnerable browser: Internet explorer 8 and earlier do not employ this feature yet, but 9 may upon its release.

* A proxy can be used to screen out requests that include the Origin headers that are used when connecting to secondary sites using javascript.

* No-script for Firefox will prevent this from working provided none of the sites that the scanner is attempting to access are in its allowed list already.

Detection

* On systems access by the a cross domain xmlhttprequest headers including the Origin will appear along with a referrer that are easily identifiable if logging is enabled to save these.

* A proxy server in addition to fitlering the headers can also log and alert when they are employeed.

References and Contact

w3c page specifying the cross origin policy:
http://www.w3.org/TR/cors/

Mozilla development blog:
https://developer.mozilla.org/En/HTTP_access_control

Proof of concept scanning tool:
https://www.bordergatewayprotocol.
net/aricon/software/python/xmlhttpscanner/xmlhttpscanner.py

Email:
nberthaume -[at]- gmail.com

Irc (efnet, freenode):
aricon

Twitter:
@nberthaume