Implementing Web Tracking

Gregory Fleischer
gfleischer@gmail.com

Abstract

Web tracking necessitates several complementary approaches. By combining passive and active fingerprinting, persistence of tracking identifiers, and unmasking techniques, an effective web tracking regimen can be developed. Although powerful new web browser features may offer additional opportunity for tracking, the variety of implementations and lack of consistently supported feature sets may limit effectiveness. Browser vendors have begun integrating privacy protections, but many of the implementations are flawed in ways that may be challenging to address without breaking users' expectations of how the web works.

Introduction

Web tracking has never been very popular with users. But it is probably as popular as ever with those people who are implementing the tracking to capitalize on vast datasets of user information. Many of the most effective approaches have been used by media companies to implement behavioral advertising and by social networking companies looking to monetize user generated content. In several cases, web users actively volunteer this information.

Within this context though, there is an overall lack of tools and research into how one goes about implementing web tracking for one's own sites. A great deal of attention is paid to what tracking may be implemented "in the wild", but there is limited emphasis placed on what practical mechanisms are actually usable for those people just looking for solutions.

The research presented in this paper builds upon the long history of web browser weaknesses that have allowed for user web tracking. Some of these techniques are well known, some are possibly misunderstood, and others have not been explored thoroughly. Where practical, an attempt has been made to provide more exhaustive documentation of web browser and plugin features and discuss how practical these are for implementing in a tracking server. This research should be seen a starting point, not a definitive treatment of a vast topic.

Implementing and deploying web tracking is based on a series of trade-offs. Browsers are constantly integrating new features and, with the rapid release approaches being used by major vendors, developing workable solutions is dependent on a number of interrelated factors. The most effective techniques are rooted in features that are difficult for browser and plugin vendors to address with compromising user experience. For instance, the "`:visited`" CSS history enumeration took years to address after being reported (and people are still arguing about it).

Ultimately, an effective tracking solution needs to encompass all facets of information exposed by the browser, but care should be taken to avoid relying on user cooperation, deception, or outright theft of user information. Those activities are better left as external to the primary motivations of user tracking that rely on strictly technical means.

Fingerprinting

Feature Detections in Browsers

User-agent sniffing is an outdated technique to detect specific features sets. Instead of parsing the “navigator.userAgent” value, properties in the browser that are specific to the actual version are used. This is the same technique used by modern JavaScript libraries such as “modernizr”.

For example, a basic (and naive) approach could appear as:

```
if ("netscape" in window) {
 alert("Firefox");
} else if ("chrome" in window) {
 alert("Chrome");
} // etc...
```

In addition to identifying specific feature sets, it is also helpful to determine if some technological means has been put in place to block execution of plugin content in the browser. Although in some cases this can be accomplished by simply attempting to load the content, other times it can be more challenging.

For example, a popular Firefox extension is the NoScript add-on. The NoScript add-on has fine-grained control over JavaScript and plugin execution. In situations where it is not possible to execute JavaScript, it could still be beneficial to detect plugin blocking.

The following example HTML can be used to detect if NoScript blocking is present for a specific plugin content-type by making a background-image request based on a specific NoScript style:

```
<html>
<head>
<style>
a.__noscriptPlaceholder__ {
background-image:url('/noscript-embed-flash-blocked.png?fefe1234');
}
</style>
</head>
<body>
<div style="width: 0px; height: 0px; overflow: hidden; display: block;">
<embed type="application/x-shockwave-flash"></embed>
</div>
</body>
</html>
```

Detecting Browser Extensions

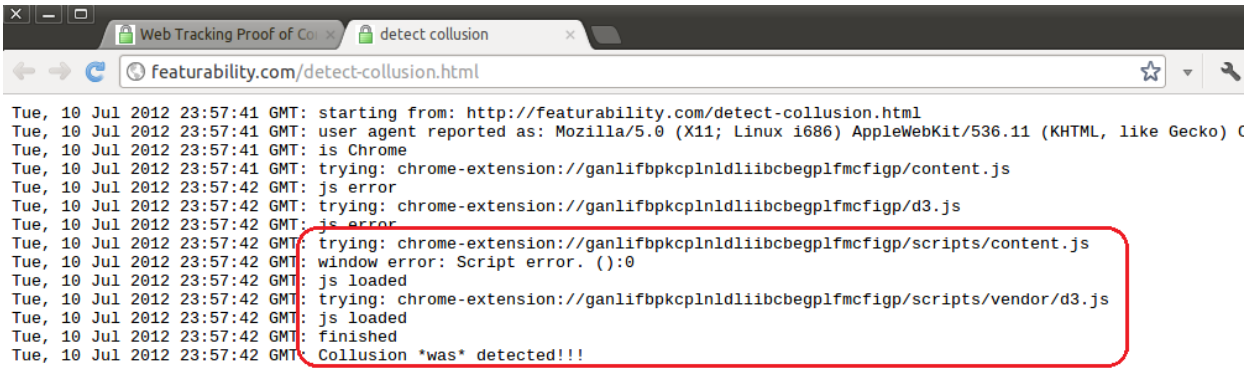
Both FireFox and Chrome promote the development of custom add-ons and extensions by developers. These custom browser extensions are created using a variety of mechanisms but are primarily based on a certain set of JavaScript components that run with higher privileges and use browser-specific conventions to interact with the user. In some cases, it is possible to detect the presence of a particular add-on or extension based on browser behaviors.

Chrome Extensions

Chrome exposes a pseudo URI for extensions “chrome-extension://”. These URIs are based on the ID of extension and can be used to reference content. Each chrome extension has a “manifest.json” file that can possibly be detected, but any suitable content can be used for detection. If a specific file and type is loadable, this can be a more powerful detection approach.

The “Collusion” extension for Chrome add-on graphs the requests between third-party sites. It can be detected using a similar approach by loading specific content that is exposed:

chrome-extension://ganlifbpkcplnldliibcbegplfmcfigp/d3.js
chrome-extension://ganlifbpkcplnldliibcbegplfmcfigp/scripts/vendor/d3.js



Firefox Resource References

Depending on how the add-on chooses to implement expose certain features, it may be possible to detect the add-on. Many add-ons implement a custom resource handler to resolve internal resources. The “resource://” pseudo URI can be used to access these values from remote content. The “chrome://” pseudo URI may be accessible if the add-on marks the reference as explicitly accessible (implicit blocking may itself be detectable). Depending on the type of value exported, dynamic script, image or CSS loading could be used to access and detect the content.

For example, the “Collusion” add-on graphs the relationships between sites that set third-party cookies. The add-on uses the Jetpack SDK and the bootstrap routine in the SDK will register a reference to embedded resources using “resource://” URI based on the ID of the add-on.

From bootstrap.js:

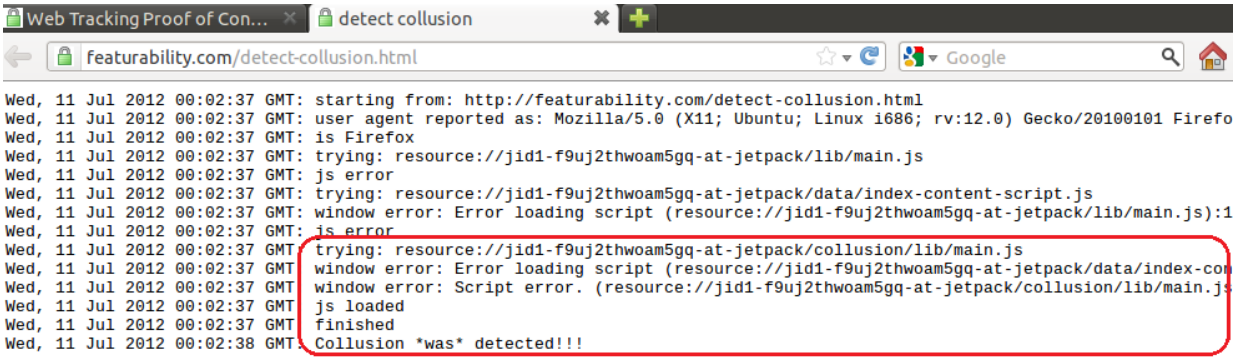
```
// Register a new resource "domain" for this addon which is mapping to
```

```
// XPI's `resources` folder.
// Generate the domain name by using jetpack ID, which is the extension ID
// by stripping common characters that doesn't work as a domain name:
let uuidRe =
  /^\{([0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12})\}$/;
let domain = options.jetpackID.toLowerCase()
                          .replace(/@/g, "-at-")
                          .replace(/\./g, "-dot-")
                          .replace(uuidRe, "$1");

let resourcesUri = ioService.newURI(URI + '/resources/', null, null);
resourceHandler.setSubstitution(domain, resourcesUri);
options.uriPrefix = "resource://" + domain + "/";
```

This allows a remote page to reference the resource content based on what is exposed:

> resource://jid1-f9uj2thwoam5gq-at-jetpack/lib/main.js
> resource://jid1-f9uj2thwoam5gq-at-jetpack/collusion/lib/main.js



Any add-on that implements a similar bootstrap routine could potentially be detected based on resource enumeration.

Firefox Custom XPCOM

If a Firefox add-on exposes a custom XPCOM interface, the XPT type information can be extracted from the add-on and used to discover the custom type by examining “Components.interfaces” [Component.interfaces]. This is a well known problem and is currently unaddressed (Bugzilla #429070). Additionally, the “Components.interfacesByID” list can be used based on a specific IID value.

For example, the Greasemonkey add-on implements a custom XPCOM interface type. This can be detected by searching for the appropriate IID.

```
'gmIGreasemonkeyService' == Components.interfacesByID['{c5826e20-1cc7-11da-8cd6-0800200c9a66}']
```

A list of all custom types can be built by extracting type information from each available add-on and then performing an exhaustive search based on the IID.

Fingerprinting via Plugins

Plugins offer a deeper level of access to operating specific data than the browser. The following charts indicate some of the information that is available to remotely hosted content. Once the information is examined in the plugin, it can either be transferred as is using an available network transmission mechanism in the plugin or returned via the JavaScript (depending on the host container to submit the data).

A more efficient mechanism may be to generate a fingerprint on the client instead of returning the complete list of data. With proper identification and versioning of the fingerprinting method, these can be used to capture fingerprints without using excessive bandwidth or resources on the server.

Flash

Action Script 2 Language Reference http://help.adobe.com/en_US/AS2LCR/Flash_10.0/help.html?content=splash.html

Version Information	getVersion()
Capabilities	System.capabilities
Camera names	Camera.names
Microphone names	Microphone.names
Fonts	TextField.getFontList()

Internet Explorer will also send a “x-flash-version” request header in some circumstances:

- `x-flash-version: 11,2,202,235`

Action Script 3 Language Reference
http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/index.html
http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/index.html?filter_flashplayer=11.2

Memory information	System.totalMemory, System.privateMemory, System.totalMemoryNumber
Capabilities	System.ime, Capabilities, Accelerometer.isSupported, PrintJob.isSupported, Geolocation.isSupported, DRMManager.isSupported
Locales	LocaleID(LocaleID.DEFAULT), StringTools.getAvailableLocaleIDNames, DateTimeFormatter.getAvailableLocaleIDNames, Collator.getAvailableLocaleIDNames, NumberFormatter.getAvailableLocaleIDNames, CurrencyFormatter.getAvailableLocaleIDNames

Camera	Camera.names
Microphone	Microphone.names
Fonts	Font.enumerateFonts
Input devices	Mouse, Multitouch, Keyboard

Silverlight
http://msdn.microsoft.com/en-us/library/cc838158%28v=vs.95%29.aspx

Environment information	System.Environment
Timezone info	System.TimeZoneInfo.Local
Audio Capture	System.Windows.Media.CaptureDeviceConfiguration.GetAvailableAudioCaptureDevices()
Video Output	System.Windows.Media.LicenseManagement.VideoOutputConnectors
Video Capture	System.Windows.Media.CaptureDeviceConfiguration.GetAvailableVideoCaptureDevices()
Fonts	System.Windows.Media.Fonts.SystemTypefaces()

Java
http://docs.oracle.com/javase/6/docs/api/

Graphics Environment	GraphicsEnvironment.getLocalGraphicsEnvironment()
Display devices	per graphics - getScreenDevices()
Display modes	per device - getDisplayModes()
Fonts	per graphics - getAllFonts()
Network interfaces	NetworkInterface.getNetworkInterfaces()

Examples of Java specific user-agents:

- `Mozilla/4.0 (Windows XP 5.1) Java/1.6.0_31`
- `Mozilla/4.0 (Mac OS X 10.7.2) Java/1.6.0_31`
- `Mozilla/4.0 (Linux 2.6.38-8-generic) Java/1.6.0_31`

Quicktime
https://developer.apple.com/library/mac/#documentation/QuickTime/Conceptual/QTScripting_JavaScript/bQTScripting_JavaScri_Document/QuickTimeandJavaScri.html#//apple_ref/doc/uid/TP40001526-CH001-SW5

Version info	GetQuickTimeVersion()
Language	GetQuickTimeLanguage()
Connection speed	GetQuickTimeConnectionSpeed()
Is professional version registered	GetIsQuickTimeRegistered()
Plugin version	GetPluginVersion()
Compute component version	GetComponentVersion()

Adobe Acrobat PDF

https://www.adobe.com/devnet/acrobat/javascript.html
http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/reader_overview.pdf

Monitors	app.monitors
Plugins list	app.plugIns
Color profiles	app.printColorProfiles
Printer names	app.printerNames
Media players	app.media.getPlayers()
Printer parameters	getPrintParams()

Adobe Acrobat PDF ActiveX

http://stackoverflow.com/questions/3666839/get-adobe-reader-version-using-javascript

Version information	GetVersions()

Internet Explorer

Client Capabilities information: http://msdn.microsoft.com/en-us/library/ms531416%28v=vs.85%29.aspx

Properties	clientCaps
Component information	getComponentVersion / isComponentInstalled

Tracking

Web Browser Methods

The use of various tracking mechanisms supported natively by the web browser is well documented. For miscellaneous references and discussions on the current state of usage, see the recent paper "Flash Cookies and Privacy II: Now with HTML5 and ETag Respawning".

Cookies

HTTP cookies are well supported in some form by every major web browser [http://en.wikipedia.org/wiki/HTTP_cookie]. Although the storage space is limited, the standard web cookie has the benefit that it will be automatically sent on every web request to servers that match the path and domain. This means even in completely passive monitoring scenarios, the cookie can still be available to be read by the receiving server.

Although web security best practices recommend setting the cookie with the HttpOnly and Secure flags, these features are undesirable when implementing web tracking. The goal should be to make sure the browser will be sent by the browser in every possible scenario as well as being readable by client-side code running within a specific domain.

Browser Cache

Web browsers support caching of content set with far-future expiration values. Unless the browser's cache is disabled, the cache history is cleared, or the resource is retrieved using a private browsing mode, any content may be stored indefinitely.

The simplest approach to using the browser cache for tracking purposes is to embed the tracking identifier directly in the content and then force it to be cached by setting the appropriate headers. By embedding the tracking identifier directly in the content, this allows for any active content to gain access to value where it is exposed.

How this is accomplished and how practical it actually is depends on the type of content.

Roughly, in order of preference of functionality and usefulness:

- JavaScript
- HTML
- CSS

In script content, storing an identifier could be as straightforward as embedding a variable and then exposing the value as a variable, function or callback. Part of what should be considered is how much global variable pollution is acceptable and whether unrelated script content should have easy access to the values.

```
// global variable
var tracking_id = "fefe1234";

// functional approach
function get_tracking_id(){return "fefe1234";}

// callback approach
(function() {if('undefined'!=typeof(tracking_id_callback))
{tracking_id_callback("fefe1234")}})();
```

Or, in HTML, the could be embedded using any practical mechanism. For example, embedded as DIV text value:

```
<div id="tracking_id">fefe1234</div>
```

In instances where JavaScript is not currently enabled or executing, this value could be read at a future date by referencing the cached HTML file.

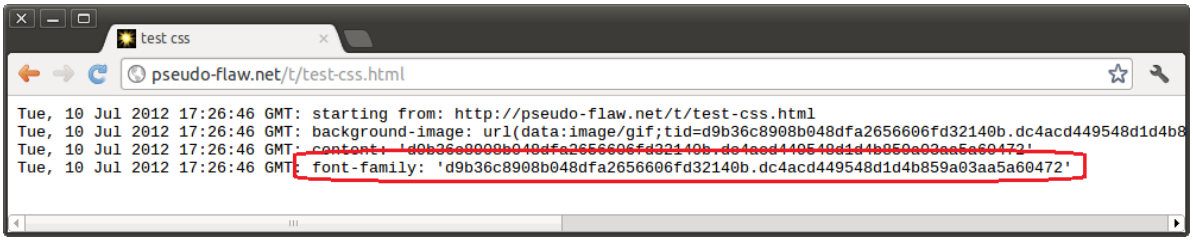
Embedding desired tracking identifiers into cached CSS content can be done in several ways, but browser support varies. Specific CSS directives that support cross-site extraction are the most beneficial.

For example, the following CSS constructs could be used to embed a tracking identifier without relying on external references:

```
#tracking_id {
   font-family: "fefe1234" !important;
background-image: url('...w==');
}

#tracking_id:after {
   content: "fefe1234";
}
```

Then from within JavaScript, the content is read using [getComputedStyle](). What is accessible will be dependent both on browser vendor and versions. For instance, the following shows that the “font-family” value is populated with the verbatim value in Chrome, but it is not readily available in the Opera browser.



These values can be read by additional script or active plugin content and propagated to additional client-side storage locations or embedded in network requests.

The caching of HTML content with embedded tracking identifiers can be improved by adding an "ETag" that duplicates the value. This can allow the server to correlate the tracking requests and potentially populate cookie values if the user's cookies have been cleared or expired. The use of ETag values for tracking has been used in real world situations to implement "Zombie cookies" by Hulu/Kissmetrics [http://www.wired.com/business/2011/07/undeletable-cookie/] and Microsoft's synching with atdmt.com [https://cyberlaw.stanford.edu/node/6715/].

In many cases, it is also possible to read the ETag header in XHMLHttpRequst responses from CORS requests as long as the appropriate header value for Access-Control-Expose-Headers is returned.

HTML5 Storage

HTML5 storage has many features that support the ability to store persistent values [http://www.html5rocks.com/en/features/storage]. The localStorage object is the most flexible and well support in modern browsers. Although other options may offer finer-grained control, these do not come without significant drawbacks. Currently, many browsers have missing or prefixed implementations based on evolving standards. And when support is available within a specific browser, user opt-in may be required.

For example, attempting to use mozIndexedDB in Firefox generates a user prompt:

Obviously, depending on user interaction is undesirable in a remote tracking scenario especially if discretion is needed.

Other storage features may not ever be implemented. Chrome implements the FileSystem API [http://www.w3.org/TR/file-system-api/], but Firefox does not intend to implement the feature [https://hacks.mozilla.org/2012/07/why-no-filesystem-api-in-firefox]. Still, this Chrome specific feature does support an additional storage mechanism through the use of temporary file storage.

Flash Shared Objects

Local Shared Object (LSO)

The use of Flash LSO objects for storage of identifiers for tracking purposes is well established. These persistent object can be shared among flash objects loaded from coordinating domains.

Traditionally, there was not an effective mechanism for browsers to interface with these values so clearing cookies and emptying the cache within the browser had no impact on these persistent values. This made Flash cookies an attractive choice to store persistent identifiers. Adobe has made changes that allow these values to be cleared in the same way as traditional cookies and some integration with private browsing modes of the various browsers has been implemented.

An item of note is that the Pepper Flash implementation in Chrome stores the shared objects in a separate directory location. For example, on Linux:

```
~/.config/google-chrome/Default/Pepper Data/Shockwave Flash/
WritableRoot/#SharedObjects/
```

These object files are not shared with other browser instances and may contain separate values that are not easily correlated between different browsers.

Remote Shared Object

In addition to LSO, Flash also supports the use of remote shared objects using a media server [http://livedocs.adobe.com/flashmediaserver/3.0/hpdocs/help.html?content=00000100.html]. The remote object supports local persistence which is functionally equivalent to the more well known LSO when a third parameter is specified:

```
var myRemote_so:SharedObject = (name, uri, _root._url);
```

By specifying `_root._url`, cooperating Flash objects from the same domain can access the stored contents of a locally persisted object. The resulting client-side persistent file ends in ".sor" [http://livedocs.adobe.com/flashmediaserver/3.0/hpdocs/help.html?content=00000099.html].

Internet Explorer UserData

Internet Explorer support limited storage in the “user data” feature available as a behavior. http://msdn.microsoft.com/en-us/library/ms531424%28v=vs.85%29.aspx

Silverlight

Silverlight supports the use of Isolated Storage to store a limited amount of data in a persistent format. Similar to Flash shared objects, modern browsers prevent access to these values in private browsing mode.

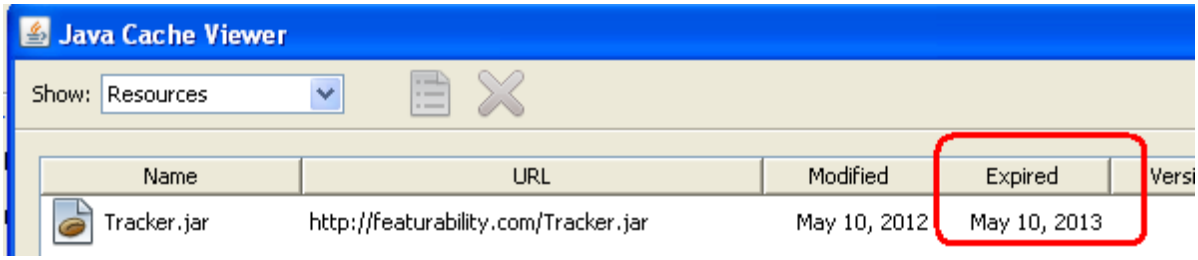
Adobe Acrobat PDF

A less well-known feature of Adobe Acrobat is support for persistence of global values in JavaScript run from PDF documents. The global.setPersistent call will save values that are automatically re-populated whenever the document is opened. These values are stored in a “glob.js” or “GlobData” file (stored beneath the user’s application data directory), although different versions of Acrobat handle the security and scoping of the values differently.

Because these values can be shared, coordinating documents from the same domain can access the values. This allows for the persistent storage of values outside of the browser.

Java

The Java runtime implements its own temporary file storage where application, applets and other resources are cached after being downloaded. This cache exists outside of any web browser and supports forced caching of content with far-future values.

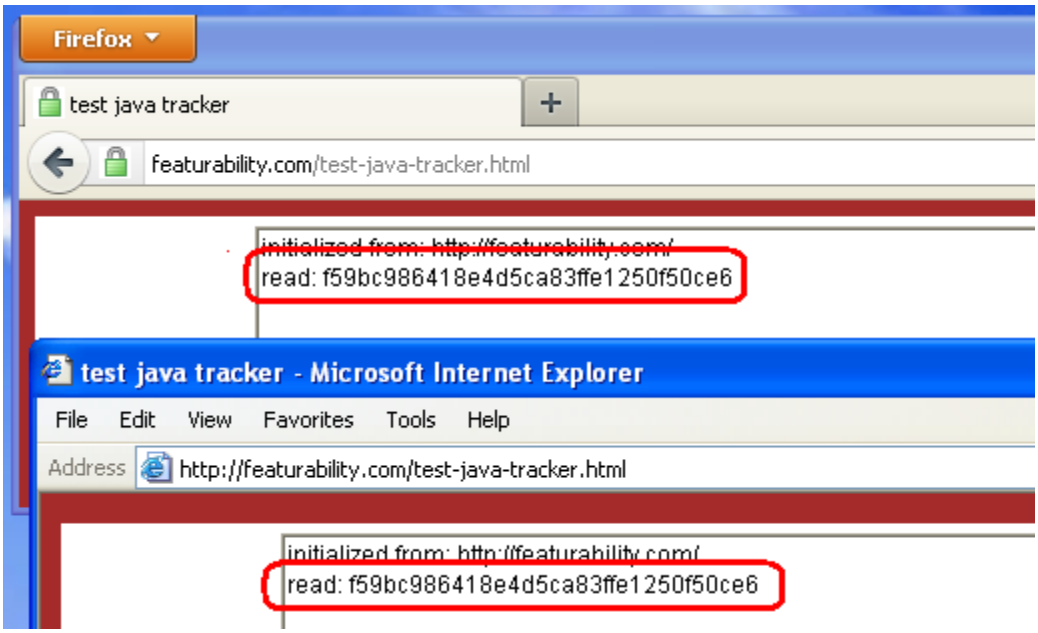


Java applets can either store embedded identifiers or load embedded resource content directly from the stored archive. Reading from an archive is more efficient than emitting custom Java class files directly.

For example, to read embedded resource content:

```
InputStream is = getClass().getResourceAsStream("trackingid.txt");
BufferedReader br = new BufferedReader(new InputStreamReader(is));
String line;
while ( (line = br.readLine()) != null) {
    response += line;
}
```

The following screenshot shows how the same applet loaded in different browsers. The same instance is loaded from the cache, even across the different browser instances.


Firefox
test java tracker
featurability.com/test-java-tracker.html
initialized from: http://featurability.com/
read: f59bc986418e4d5ca83ffe1250f50ce6
test java tracker - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://featurability.com/test-java-tracker.html
initialized from: http://featurability.com/
read: f59bc986418e4d5ca83ffe1250f50ce6

Unmasking

Background

This issue has been a constant topic of interest over the years. In many cases, the same techniques discussed years ago continue to work with little modifications. Between plugins that can make direct socket connections, requests that leak DNS information or external media players that do not respect system settings, it is frequently possible to detect the remote IP (or username) of web browser users.

For some relevant background information, see:

- Metasploit Decloaking Engine (outdated and slightly broken)
- Unmasking You (Black Hat and DEF CON presentation)

Occasionally, new web browser features may expose additional attack surface.

Recent Web Features

Tor WebSocket Weakness

An vulnerability related to DNS leakage was identified in the WebSocket support used in the Tor Web Browser Bundle. https://blog.torproject.org/blog/firefox-security-bug-proxy-bypass-current-tbbs

Content Site Policy and CGI Proxies

The Content Site Policy (CSP) is an attempt to protect websites from a variety of content injection issues. The specification is currently in a state of flux.

In order to facilitate implementation and deployments, a reporting mechanism for violations is defined. When a violation is observed, a report is sent to a predefined location. The policy and reporting location are transmitted via HTTP response headers.

The Chrome browser implements support for CSP using the “X-WebKit-CSP” header. Reporting of violations is supported, but the URI used does not have to be “same-origin” as the violation (this differs from the Firefox implementation where violations must be same-origin).

Where this is interesting is in the case of CGI proxies that transparently rewrite HTML body content, but return HTTP headers unmodified. If the CGI proxy does not modify the headers by rewriting external references, then by including an appropriate CSP policy, it is possible to force the browser to make an unproxied connection to a server.

For example, the following screenshots demonstrate this issue in Chrome by requesting a resource using the CGI proxy at http://www.1proxy.de. The response contains a “X-WebKit-CSP” header that will cause a report violation for an embedded image. This URL is not sanitized in the response headers; the Chrome browser will process the violation and make the request directly and bypass the CGI proxy. The actual user IP address will be exposed.

GET request to http://www.1proxy.de/index.php?q=aHR0cDovL3BzZXVkby1mbGF3Lm5ldC9jL2...

previous | next | action

request | response

raw | headers | hex | html | render

```
HTTP/1.1 200 OK
Date: Wed, 09 May 2012 02:32:42 GMT
Server: Apache/2.2.16 (Debian)
X-Powered-By: PHP/5.3.3-7+squeeze8
Cache-control: no-cache, no-store
X-Content-Security-Policy-Report-Only: img-src 'none'; report-uri
http://pseudo-flaw.net/c/csp-test/report.cgi/6d6d5815f6e3923a2dc95cf352d29357

X-WebKit-CSP: img-src 'none'; report-uri
http://pseudo-flaw.net/c/csp-test/report.cgi/6d6d5815f6e3923a2dc95cf352d29357

Set-Cookie: flags=2ed; expires=Wed, 06-Jun-2012 02:32:42 GMT; path=/;
domain=.www.1proxy.de
Content-Disposition: inline; filename="index2.cgi"
Content-Length: 2339
Vary: Accept-Encoding
Content-Type: text/html
```

www.1proxy.de/index.php?q=aHR0cDovL3BzZXVkby1mbGF3Lm5ldC9jL2NzcC10ZXN0L2luZGV4Mi5jZ2k%3D&hl=

Address: http://pseudo-flaw.net/c/csp-test/index2.cgi Go [go: up one

☑ Include Form ☐ Remove Scripts ☑ Accept Cookies ☑ Show Images ☑ Show Referer ☐ Rotate13 ☑ Base64 ☑ Strip Meta ☐ Strip Title

Raw Headers (217.79.179.55)

```
HTTP_REFERER: http://pseudo-flaw.net/c/csp-test/index2.cgi
HTTP_CONNECTION: close
HTTP_ACCEPT: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
HTTP_USER_AGENT: Mozilla/5.0 (X11; Linux i686) AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.168 S
HTTP_HOST: pseudo-flaw.net

Generated: 6d6d5815f6e3923a2dc95cf352d29357
```

CSP Response (217.79.179.55)

```
Request ID: 6d6d5815f6e3923a2dc95cf352d29357
Remote IP: 50.17.181.9
```


www.1proxy.de/index.php
what is my ip - Google Sea
www.google.com/search?sourceid=chrome&ie=UTF-8&q=what
+You
Search
Images
Maps
Play
YouTube
News
Gmail
Documen
Google
what is my ip
Search
About 270,000,000 results (0.12 seconds)
Everything
Your public IP address is 50.17.181.9 -
earn more

Private Browsing Modes

Implementation Weaknesses

Private browsing modes are often subject to implementation weaknesses. These can manifest themselves within the browser functionality or in interactions with external plugins.

Safari

In Safari 5, when “Private Browsing” is launched, a new browser session is not started which introduces a number of weaknesses. For example:

- page level JavaScript variables are not cleared
- background JavaScript continues
- window.name is not cleared
- window expando properties are not cleared

So, if Safari private browsing is launched on a given page, that page can simply use “setInterval()” to invoke a function to periodically check if cookies need to be repopulated. Using a Web Worker would be another possible approach to allow for the execution of background script. This is trivially observed in Safari 5 by launching the prime number calculation demo and then starting the “Private Browsing” mode -- the background processing continues uninterrupted: http://www.whatwg.org/demos/workers/primes/page.html.

Local Connections

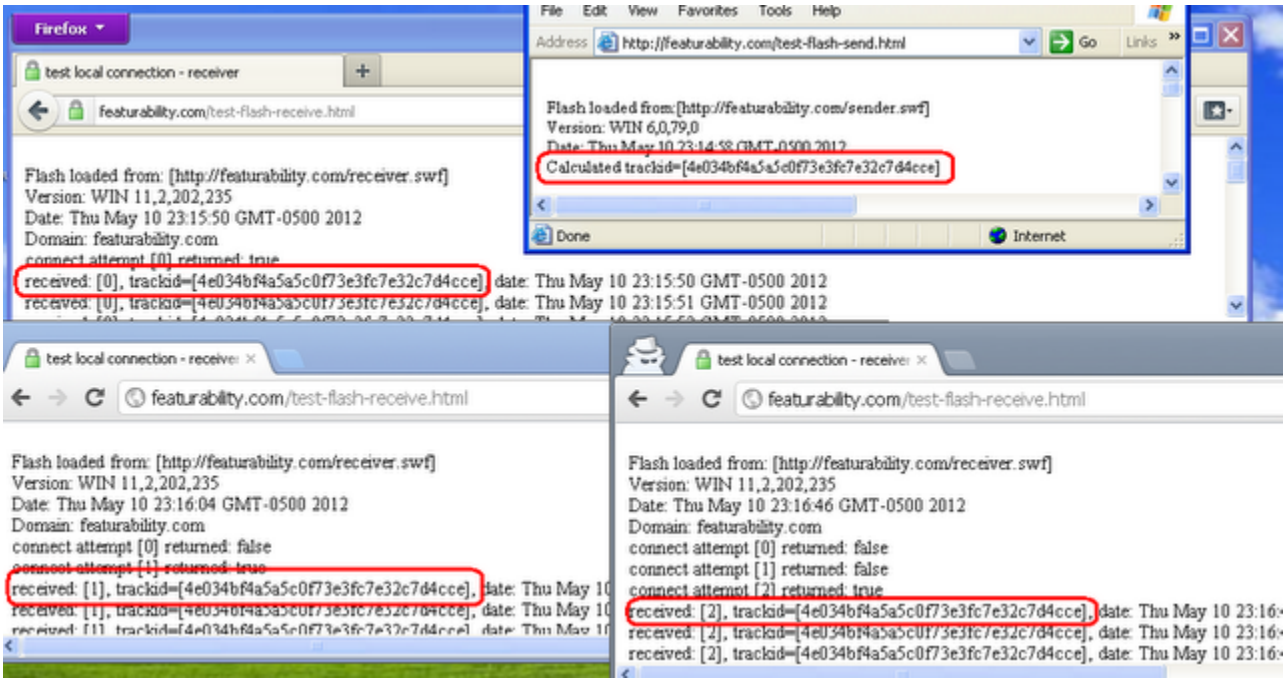
Some extensions offer the ability to initiate local connections between separate plugin instances on the same machine. When these communications extend between content running in a private browsing session and normal browsing session, it can be used to correlate user activity.

Inter-browser and Intra-browser Communication with Flash

Flash supports LocalConnection objects to facilitate communications between different SWF files residing on the same computer. As the documentation states, “LocalConnection objects can communicate only ***among files that are running on the same client computer***” [http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/LocalConnection.html].

For private browsing that launches a new browser session, Flash can be used to communicate between instances running in the initial browser session and the new private browsing session. This can also be used to communicate between separate browser versions if the same Flash implementation is used.

The following screenshot shows a browser that is sending content via several LocalConnection objects and multiple browser instances receiving the same value (including some running private browsing mode).

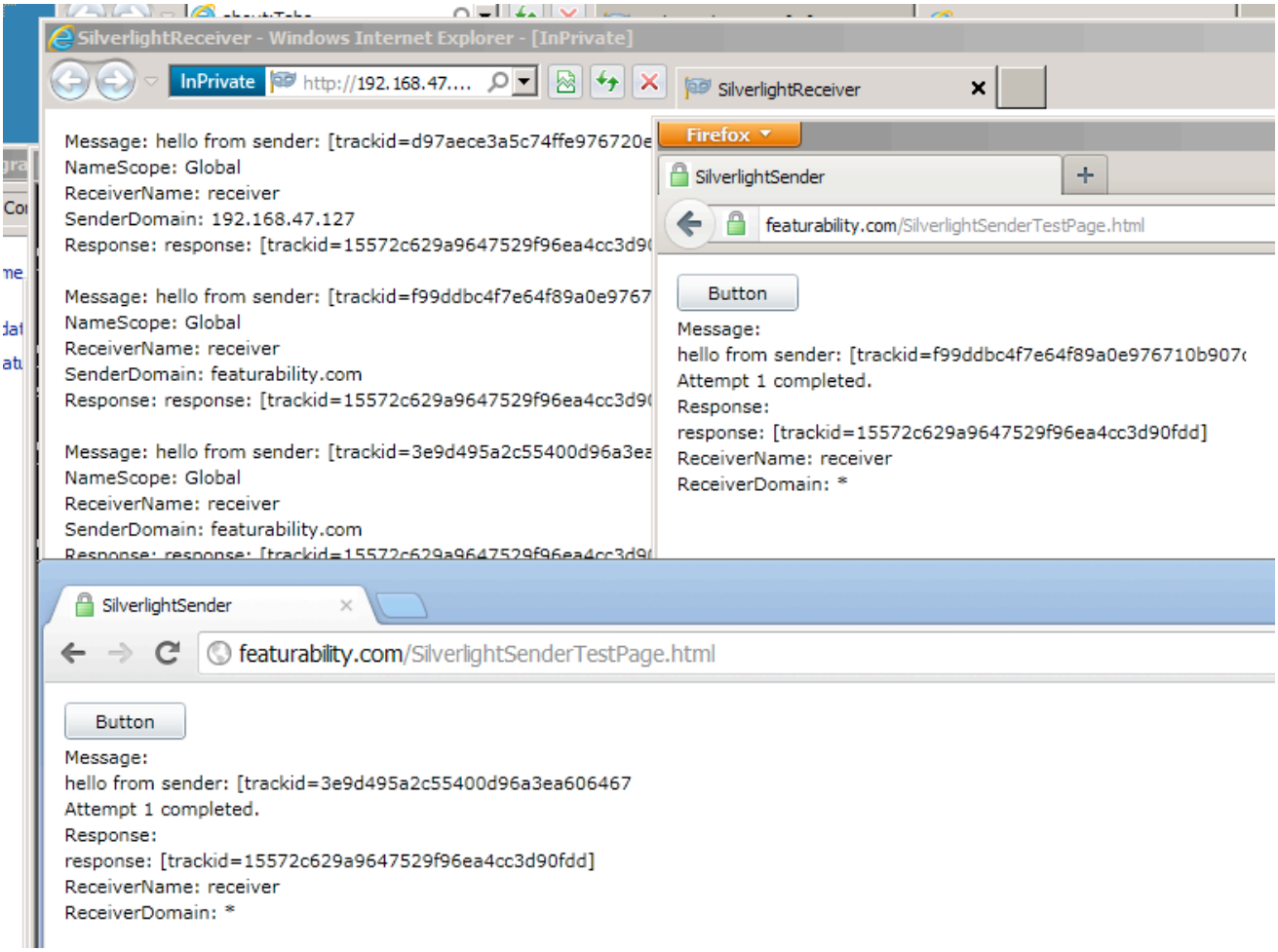


If a user visits a website (or coordinating websites) with multiple browsers, this can be detected through the sending and receiving of Flash methods via LocalConnection.

However, when disparate versions of Flash are used, the LocalConnection may not function across browser versions. For example, Chrome has added an integration with the Adobe Flash Pepper API. Future versions of Flash for Linux may be limited to Pepper. In these instances, the LocalConnection communication is limited to the same browser version, but can still communicate between the private browsing session and normal session.

Inter-browser and Intra-browser Communication with Silverlight

The Silverlight runtime supports communication between local instances of Silverlight applications [http://msdn.microsoft.com/en-us/library/dd833063(v=vs.95)]. By using senders (LocalMessageSender) and receivers (LocalMessageReceiver), messages can be transferred between cooperating instances.



The Silverlight implementation has the added benefit that receiver responses are automatically supported.

Tracking Protection Lists

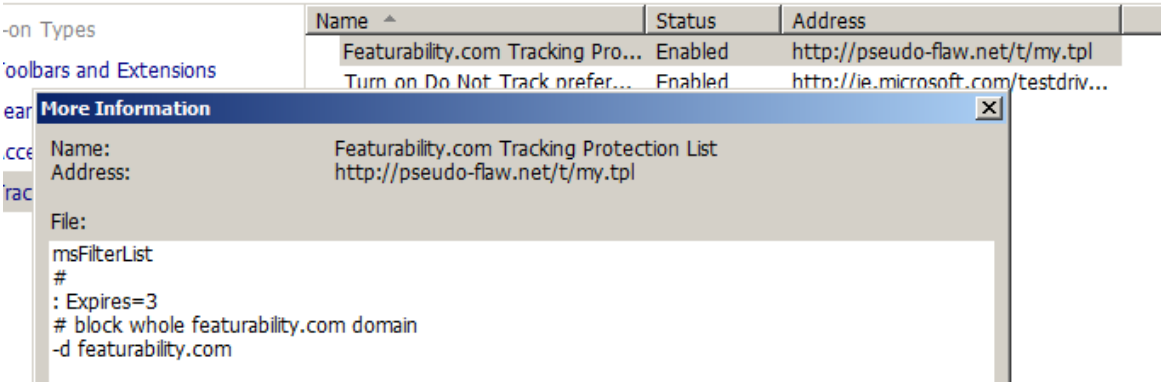
Microsoft's Internet Explorer supports the concept of Tracking Protection Lists (TPL). These lists allow for third-party sites to be blocked based on matches against a set of curated lists provided by several organizations [https://ie.microsoft.com/testdrive/Browser/TrackingProtectionLists/].

When a TPL is installed and enabled, any third party URL content appearing on the list is intended to be blocked from being loaded by the browser unless the content is loaded when the user navigates to it. In other words, if the URL is served as a first-party resource, it will be loaded.

A tracking protection configuration to block all of the "featurability.com" domain could appear as:

```
msFilterList
#
: Expires=3
# block whole featurability.com domain
-d featurability.com
```

This TPL would be loaded based on user acceptance and is stored within the browser. In IE9, the TPL functionality is also used to enable Do Not Track headers.



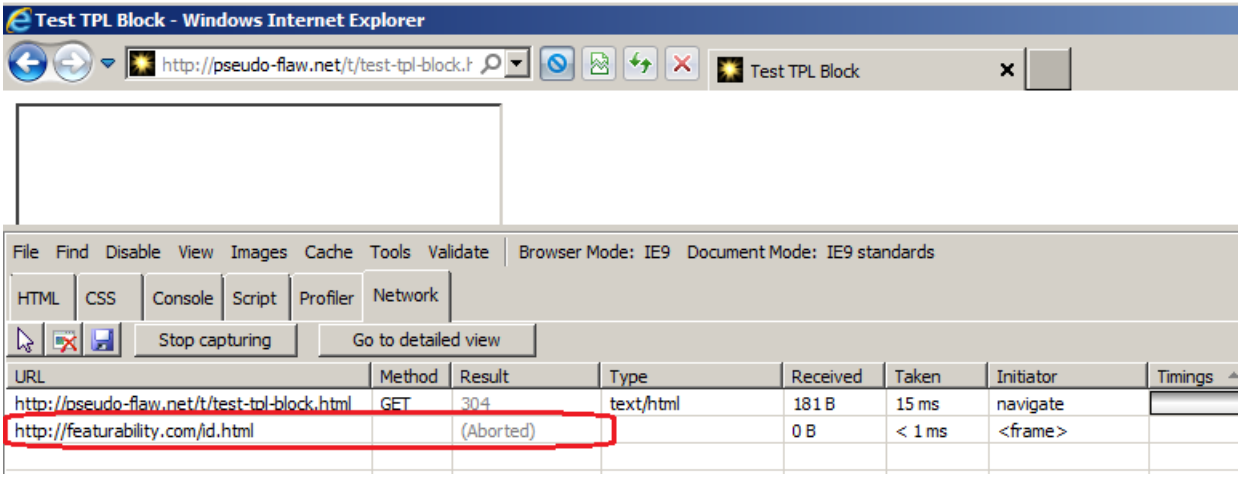
However, there are several possible mechanisms to work around these TPL restrictions.

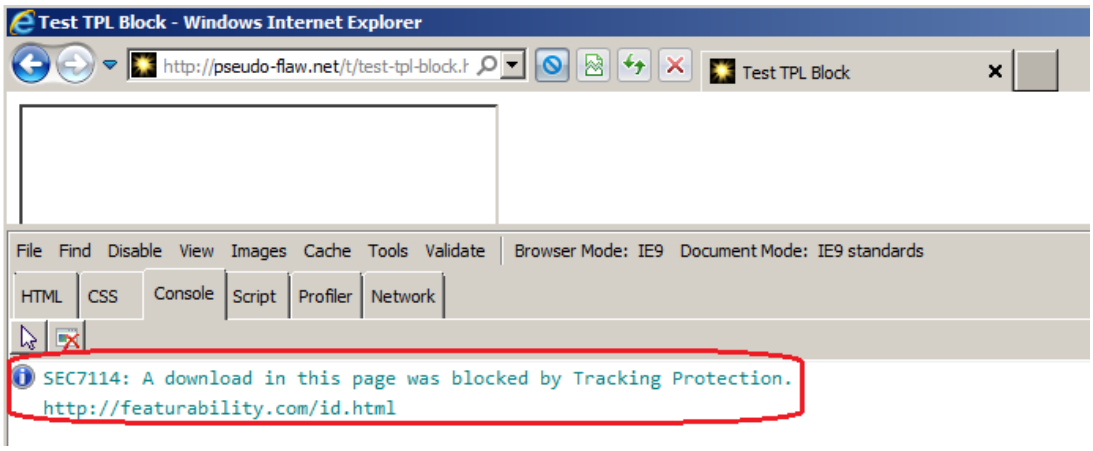
One method is to simply navigate the user's browser to the third-party location. When the page is loaded as first-party content, the TPL validation is bypassed. For example, the following HTML attempts to load an iframe reference to the blocked content. This should be blocked. Then, after a slight delay, the browser is navigated to the same window.

```
<!DOCTYPE html>
<html>
<head>
<title>Test TPL Block</title>
<script type="text/javascript">
function go() {
 setTimeout(function() {
  window.location = "http://featurability.com/id.html";
 }, 5000);
}
```

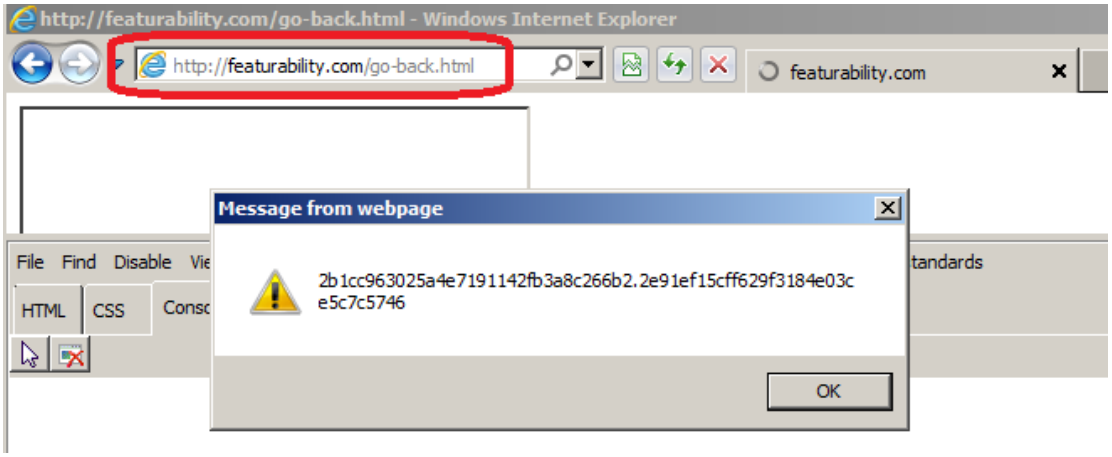
```
</script>
</head>
<body onload="go()">
<iframe src="http://featurability.com/id.html"></iframe>
</body>
</html>
```

When the page content is displayed initially, the request to the blocked content is rejected.





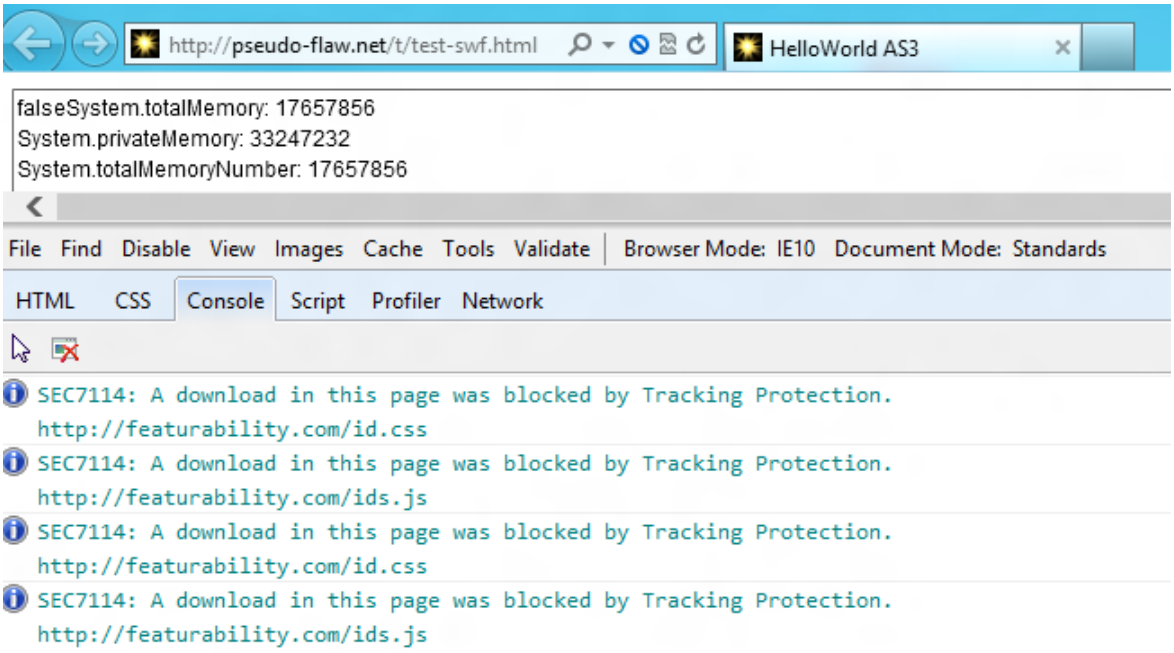
But, after the delay, the page will be navigated to the location and will bypass the tracking protection.

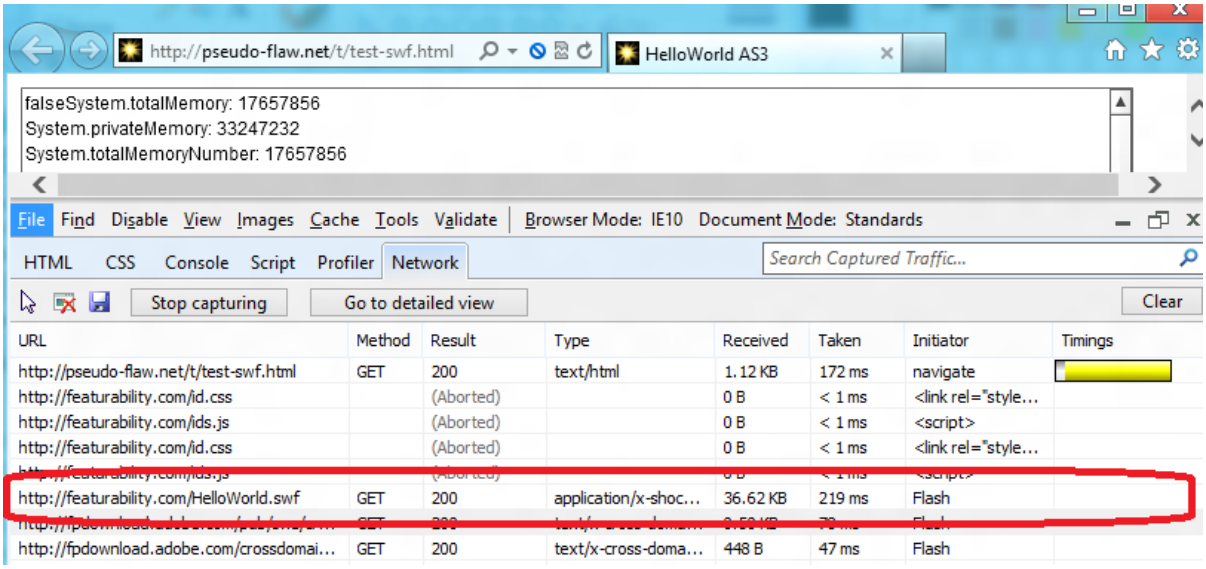


Although this page simply displays a test message to the user, normally the user would be silently redirected back to the original page.

Another method is to use active content such as Flash, Silverlight or Java plugins to load the third-party content. These are not restricted in the same way as normal browser content.

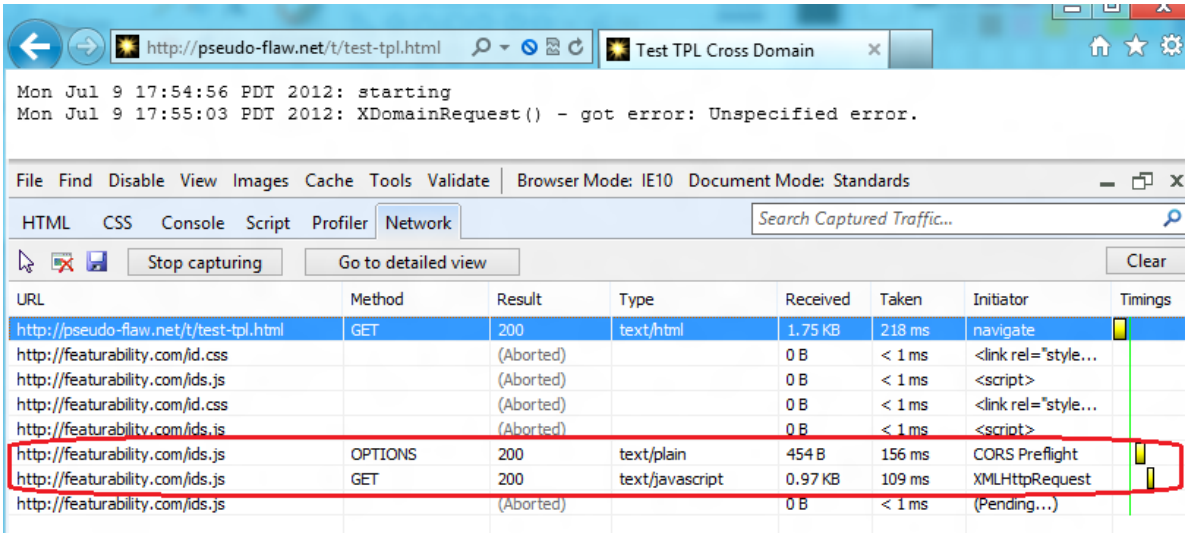
For example, the following screenshot shows a sample Flash object being loaded IE10 Preview Release and bypassing the TPL restrictions applied to the other resources loaded from the same domain.





Similarly Java and Silverlight content can be loaded by including appropriate crossdomain references.

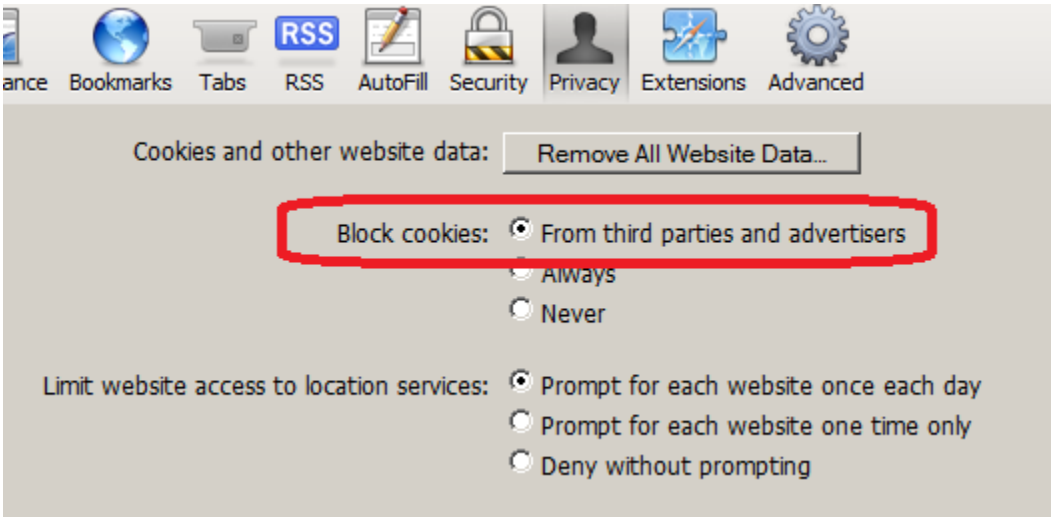
In IE10 Preview Release, added enhanced support for cross-domain XML requests using XMLHttpRequest. Previously, the IE specific XDomainRequest was supported, but it had numerous limitations. As shown in the following screenshot, the TPL restrictions are applied to XMLDomainRequest, but not requests made using XMLHttpRequest for either the CORS preflight or the actual content retrieval.



Third Party Cookie Restrictions

Some web browsers implement cookie restrictions that prevent third party sites from setting cookies in the browser. Any content included from the the third party site is prevented from setting new cookies or accessing existing cookies unless the user has visited the site (in the past) or manually interacted with the site.

By default, Safari will block cookies from third party sites unless the user visits the site. However, once the user visits the site, then any cookies are available and can be accessed or read in the future.



There are several possible approaches to bypassing these restrictions.

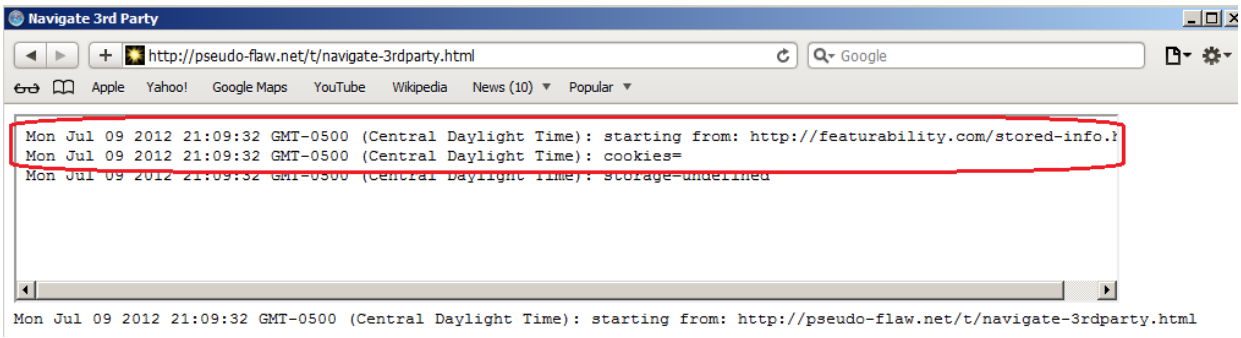
If there is a technical vulnerability in browser implementation, it may be possible to set cookies directly. For example, the Safari restrictions on third party cookies were able to bypassed using a automatic form post to the third party site.

But as seen with bypassing IE TPL restrictions, navigating the current browser to the third-party site is sufficient to set the required cookies.

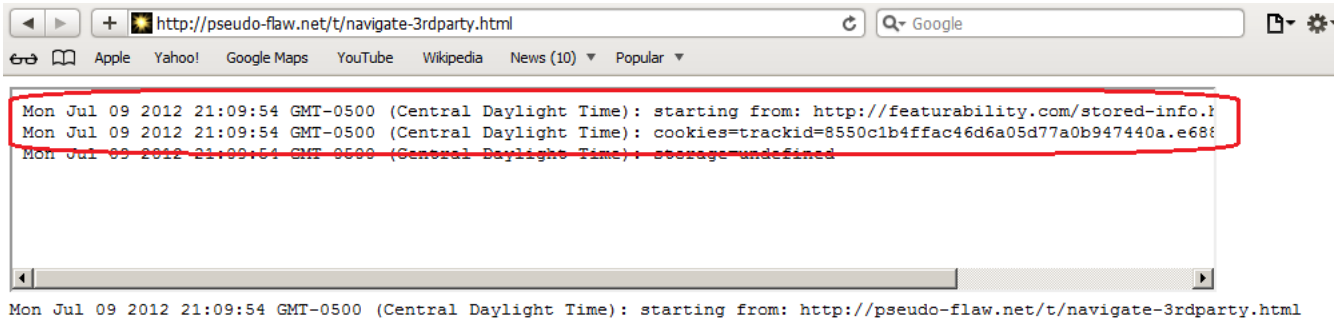
```
<!DOCTYPE html>
<html>
<head>
<title>Navigate 3rd Party</title>
<script>
var display;
function log(message) {
 display.innerHTML += (new Date()) + ": " + message + "\n";
}
function init() {
 display = document.getElementById("display");
 log("starting from: " + document.location);
}
function go() {
 setTimeout(function() {
```

```
  window.location = "http://featurability.com/go-back.html";
 }, 5000);
}
</script>
</head>
<body onload="init(); go();">
<iframe src="http://featurability.com/stored-info.html" width="90%"></
iframe>
<div id="display" style="white-space: pre; font-family: courier
new;font-size: smaller;"></div>
</body>
</html>
```

When this page is loaded, no third-party cookie is set.



The HTML will navigate the browser window and set the needed cookie.

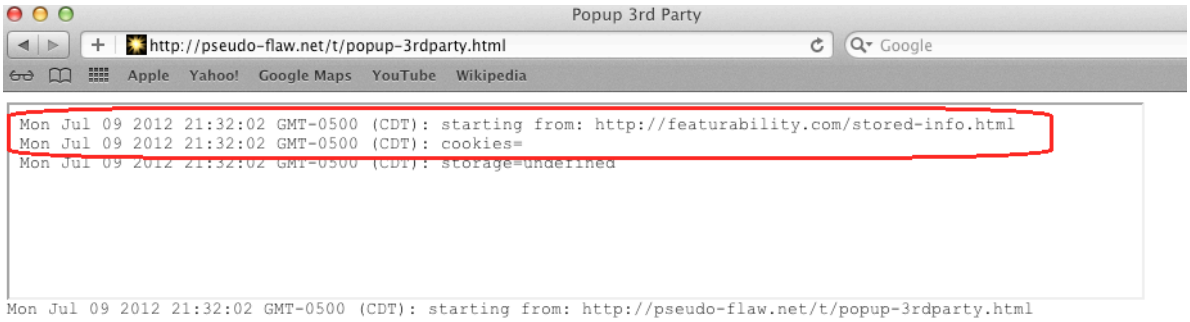


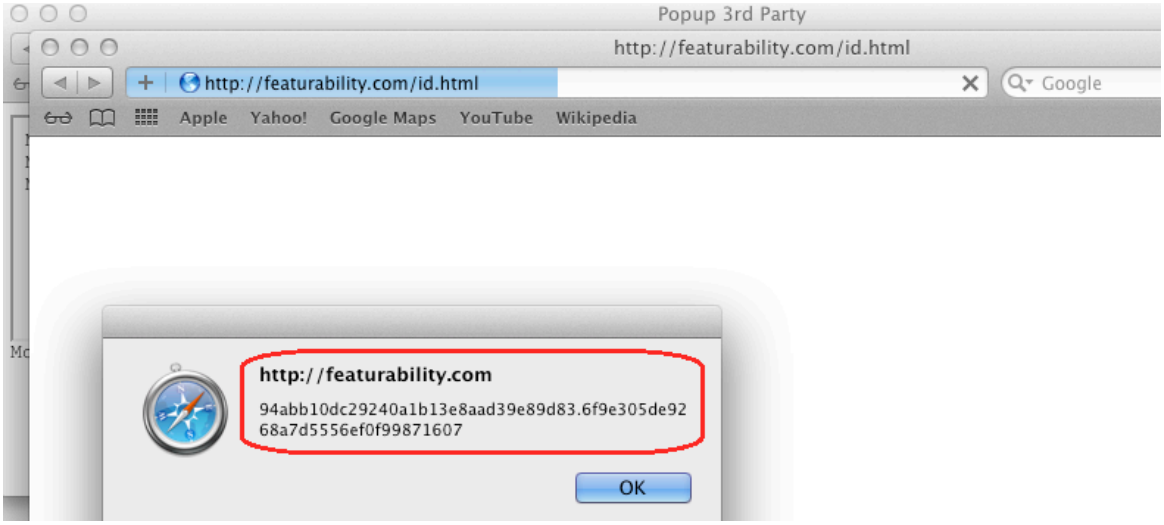
There is also probably a number of variations on this approach. For example, by modifying the previous script to open a popup window with the third party site, the cookies will also be set.

```
var topw, iframe0;
function go() {
 iframe0 = document.getElementById("iframe0");
 setTimeout(function() {popup();}, 5000);
}
function popup() {
 try {
  var a = document.createElement("a");
  a.href = "http://featurability.com/id.html";
  a.target = "_xxx";
  document.body.appendChild(a);
```

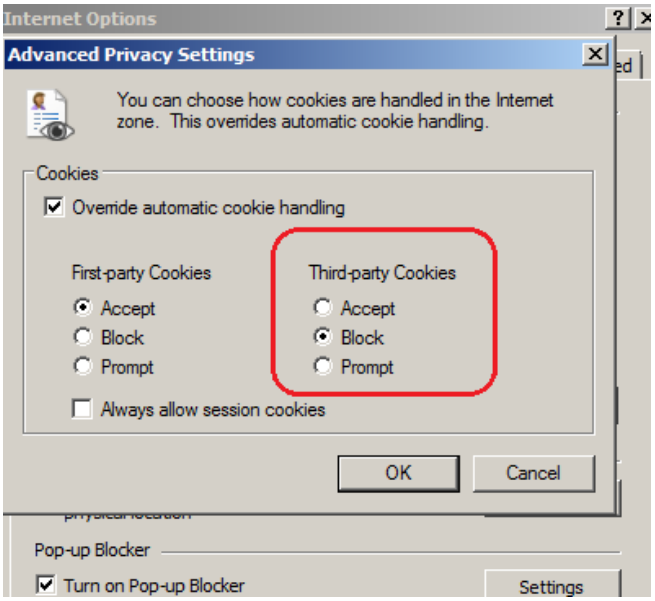
```
    var evt = document.createEvent("MouseEvent");
    evt.initMouseEvent("click", true, true, window, 0, 0, 0, 0, 0, false,
false, false, false, 0, null);
    a.dispatchEvent(evt);
    setTimeout(function() {
     topw = window.open('javascript:self.close()', '_xxx');
     topw.close();
     iframe0.contentWindow.location = iframe0.src;
    }, 1000);
   } catch (e) {log('error: ' + e.message);}
  }
```

The popup is opened and then closed. In the process, the cookie is set.





Internet Explorer offers a custom setting for blocking of third-party cookies.



As with the previously discussed TPL bypass, navigating the IE browser to the third-party location will set the cookie.

Third Party Cookie Alternatives

Assuming that third-party cookies can be completely blocked, modern browsers offer a powerful alternative based on the `localStorage` and `postMessage`. This is the approach taken by site to maintain authentication across multiple sites (e.g., XAuth, Stackoverflow's authentication, http://kevinmontrose.com/2011/09/26/disabling-third-party-cookies-doesnt-meaningfully-improve-privacy/).

For example, the following HTML content could be used to store an identifier using `localStorage` and then communicate the value to any parent page that loaded it by using `postMessage`.

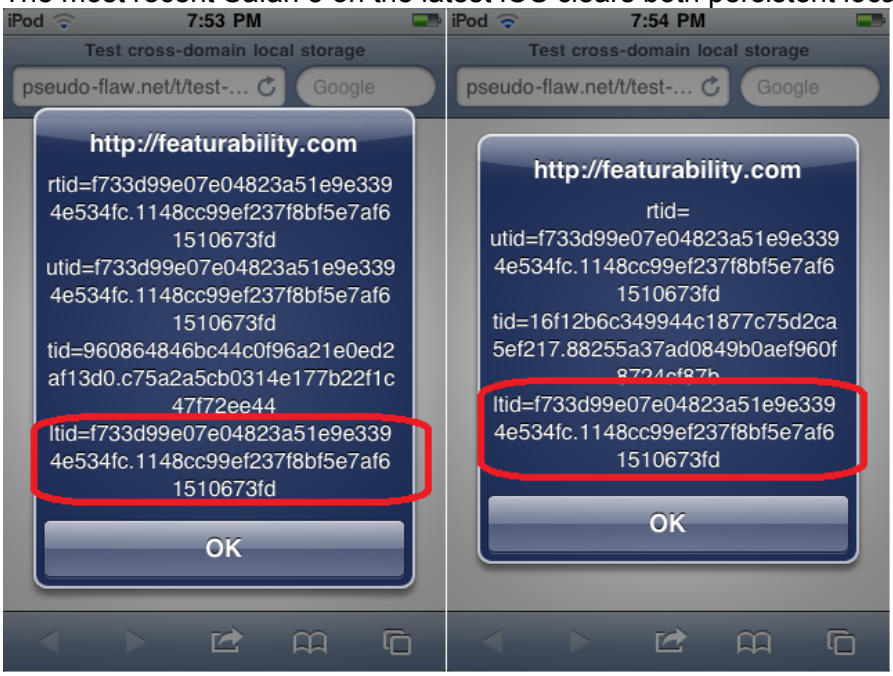
```
<html>
<body>
<script type="text/javascript">
(function(){
 if (!parent || !parent.postMessage) { return ; }
 var w = window;
 var tid = "fefe1234";
 var utid = tid, ltid = null;
 try {
   ltid = localStorage["tid"];
   if (!ltid || ltid == "null" || ltid == "undefined") {
    localStorage["tid"] = tid;
    ltid = null;
   }
 } catch(e) {}
 if (ltid) {
  utid = ltid;
 }
 var cb = function(event) {
```

```
  var rtid = (event.data || "").toString();
  alert('rtid='+rtid+'\nutid='+utid+'\ntid='+tid+'\nltid='+ltid);
 };
 if (w.addEventListener) {
  w.addEventListener("message", cb, false);
 } else {
  w.attachEvent("onmessage", cb);
 }
 parent.postMessage(utid, "*");
})();
</script>
</body>
</html>
```

Any page that wished to communicate with this content could import a JavaScript routine that register for the expected messages. The following implementation also stores any received values in a first-party cookie that is decorated with the name of the remote origin domain.

```
(function () {
    try {
      var w = window, d = document;
      var cb = function(event) {
          try {
            var tid = event.data.toString(), ctid = null;
            var cleaned = event.origin.toString().replace(/\W+/g, '_');
            if (document.cookie) {
                var lookup = cleaned + '_tid=';
                var cookies = document.cookie.toString();
                var content = null;
                var n = cookies.indexOf(lookup);
                if (n >= 0) {
                  var content = cookies.slice(n+lookup.length);
                  n = content.indexOf(';');
                  if (n >= 0) {
                      content = content.substring(0, n);
                  }
                  ctid = unescape(content);
                }
            }
            document.cookie = cleaned + '_tid='+ escape(tid)
+';path=/;expires='+(new Date(1666666666666)).toGMTString();
            event.source.postMessage(ctid, event.origin);
          } catch(e){}
      };
      if (w.addEventListener) {
          w.addEventListener("message", cb, false);
      } else {
          w.attachEvent("onmessage", cb);
      }
      var i = d.createElement("iframe");
      i.heigth = i.width = 0; i.frameBorder = 0;
      i.top = i.left = "-1234px";
      i.src = "//featurability.com/lstorage.html";
      d.body.appendChild(i);
    } catch(e) {}
})();
```

The following screenshots show the use of `localStorage` and `postMessage` to persist an identifier across domains in the older Mobile Safari on iOS 4 after cookies have been cleared. The most recent Safari 5 on the latest iOS clears both persistent local storage and cookies.



Do Not Track

Do Not Track

The Do Not Track ("DNT") header is supposed to be an opt-in indicator that a user does not want to be "tracked". First implemented in Firefox, other browsers are slowly adding support and online advertisers are pledging limited support (unless it conflicts with their respective business models). There is considerable discussion about if it serves any purpose, although a W3C standard is being developed [http://www.w3.org/TR/tracking-dnt/].

The DNT header does enforce any tracking protection and is not a technical control. It is similar to how the "Accept-Language" can be used to indicate a preference, but there is no actual restriction placed on the response. The remote server must be configured to respect the header.

DNT Header Weakness in Firefox XMLHttpRequest

The Firefox implementation is subtly flawed, because client-side script can remove the header value in submissions (Bug #751452). For example:

```
var xhr = new XMLHttpRequest();
xhr.open("GET", "http://tracker.example/", false);
if (navigator.doNotTrack) { // not a boolean
  xhr.setRequestHeader("DNT", null);
}
xhr.send()
```

If the case of cross-domain requests, a preflight request will be made, and an appropriate CORS response can be returned. This response can be future cached using the "Access-Control-Max-Age" header to limit round trips.

DNT Header in Plugins and Non-HTTP Protocols

There are some rather obvious shortcomings related to the DNT header in plugins and protocols that do not support HTTP headers. The W3C tracking draft specification alludes to the potential issues in a couple of sections:

4.4 Plug-In APIs
4.5 Tracking Preference Expressed in Other Protocols

Without significant effort, it is unlikely that protocols such as FTP or binary Flash communications will be enhanced to support the use of a DNT header. And unless an actual client-side restrictions are put in place, the DNT header is likely to be ignored.

Tracking Server

In order to support communications from plugins, modern web-browser requests, and tracking and unmasking several protocols need to be implemented in a tracking server.

Protocols

Several specific protocols and services should be configured to provide support for web requests and plugin requests. Which protocols depend on the specific scenario that is being utilized.

For example:

- DNS (tcp/53, udp/53)
- FTP (tcp/21)
- HTTP (tcp/80)
- HTTPS (tcp/443)
- Flash Policy Server (tcp/843)
- Silverlight Policy Server (tcp/943)
- RTMP (tcp/1935)
- XML sockets (arbitrary tcp >1024)

Additional specialized web requests and protocols should be supported as needed.

Plugins

Due to cross-domain restrictions, plugin content often depend on specialized policies. To support the greatest number of possible scenarios, it is important to be overly permissive. If these policies were used in a normal production web server, the applications hosted on the server would be opened to critical vulnerabilities.

Flash

To support web URL requests from Flash (such as LoadVars), a permissive “crossdomain.xml” should be implemented and be available from the root web directory:

```
<?xml version="1.0"?>
<cross-domain-policy>
<allow-access-from domain="*" />
<allow-http-request-headers-from domain="*" headers="*" />
</cross-domain-policy>
```

To support binary communications, a policy server needs to be implemented. This should run on port 843. A permissive policy file should be used:

```
<?xml version="1.0"?>
<cross-domain-policy>
   <site-control permitted-cross-domain-policies="master-only"/>
   <allow-access-from domain="*" to-ports="*" />
</cross-domain-policy>
```

Java

Java network connections depend on a number of factors. For web URL requests, an “crossdomain.xml” file is needed and, any DTD declarations should be excluded to prevent

unneeded network requests (Java blindly fetches any DTD referenced).

Older Java versions only recognized the most rudimentary format:

```
<?xml version="1.0"?>
<cross-domain-policy>
<allow-access-from domain="*" />
</cross-domain-policy>
```

More recent versions appear to support the same values as Flash, but this should not be depended upon.

To establish binary socket communications, several items are considered before allowing the connection:

- location of where the applet was loaded from
- does the PTR record for the domain name match?
- complicated reverse auth DNS validations
- version of the Java plugin (revision and NextGen versus legacy)
- presence of crossdomain.xml file
- the destination port of the connection attempt

Some of these validations are to prevent DNS rebinding attacks or unexpected requests against virtually hosted content. Unfortunately, that can make it challenging to rely upon Java for binary socket connections across of all possible configurations.

Silverlight

The Silverlight runtime uses a “clientaccespolicy.xml” to control URL requests and expects it to appear at the root of the website. Additionally, to support binary communications, a policy server needs to be implemented on port 943.

An permissive file will allow most types of connections and scenarios:

```
<?xml version="1.0" encoding ="utf-8"?>
<access-policy>
  <cross-domain-access>
    <policy>
      <allow-from http-methods="*" http-request-headers="*">
        <domain uri="*" />
        <domain uri="http://*" />
        <domain uri="https://*" />
      </allow-from>
      <grant-to>
        <resource path="/" include-subpaths="true" />
      </grant-to>
    </policy>
    <policy>
      <allow-from>
        <domain uri="*" />
      </allow-from>
      <grant-to>
        <socket-resource port="4502-4534" protocol="tcp" />
      </grant-to>
    </policy>
```

```
  </cross-domain-access>
</access-policy>
```

Additional References

Storage and Persistence

http://www.html5rocks.com/en/tutorials/indexeddb/
https://developer.mozilla.org/en/IndexedDB/
https://developers.google.com/chrome/whitepapers/storage
http://www.w3.org/TR/webdatabase/
http://hacks.mozilla.org/2012/02/storing-images-and-files-in-indexeddb/
http://kevinmontrose.com/2011/09/26/disabling-third-party-cookies-doesnt-meaningfully-improve-privacy/
https://hacks.mozilla.org/2012/03/there-is-no-simple-solution-for-local-storage/
https://blogs.msdn.com/b/ie/archive/2012/03/21/indexeddb-updates-for-ie10-and-metro-style-apps.aspx?Redirected=true
http://klanguedoc.hubpages.com/hub/How-To-Use-HTML5-File-System-Access-API
http://www.w3.org/TR/file-system-api/
http://www.html5rocks.com/en/tutorials/appcache/beginner/
https://developer.mozilla.org/en/DOM/window.navigator.mozIsLocallyAvailable
http://webreflection.blogspot.com/2012/06/asynchronous-storage-for-all-browsers.html
http://www.w3.org/TR/file-system-api/
http://www.html5rocks.com/en/tutorials/file/filesystem/
http://diveintohtml5.info/storage.html

Privacy and Tracking Protection

https://blogs.msdn.com/b/ie/archive/2010/12/07/ie9-and-privacy-introducing-tracking-protection-v8.aspx
http://windows.microsoft.com/en-US/internet-explorer/products/ie-9/features/tracking-protection
https://www.torproject.org/torbutton/en/design/index.html
http://www.w3.org/TR/tracking-dnt/
www.w3.org/2011/track-privacy/papers/Yahoo.pdf
http://www.ftc.gov/os/2012/03/120326privacyreport.pdf
http://www.whitehouse.gov/the-press-office/2012/02/23/we-can-t-wait-obama-administration-unveils-blueprint-privacy-bill-rights
https://blog.torproject.org/blog/improving-private-browsing-modes-do-not-track-vs-real-privacy-design
https://wiki.mozilla.org/Security/Anonymous_Browsing
http://answers.microsoft.com/en-us/ie/forum/ie9-windows_vista/how-to-set-internet-explorer-9-to-start-in/d706fa5d-7d76-e011-8dfc-68b599b31bf5
https://ie.microsoft.com/testdrive/Browser/DoNotTrack/Default.html
http://msdn.microsoft.com/en-us/library/hh273399(v=vs.85).aspx - Creating and Hosting TPLs

Plugins and Protocols

http://support.google.com/chrome/bin/topic.py?hl=en&topic=1678462&parent=1678463&ctx=topic
https://www.adobe.com/devnet/acrobat/javascript.html

http://livedocs.adobe.com/flex/3/progAS_flex3.pdf
https://developer.apple.com/library/mac/#documentation/QuickTime/Conceptual/QTScripting_JavaScript/bQTScripting_JavaScri_Document/QuickTimeandJavaScri.html#//apple_ref/doc/uid/TP40001526-CH001-SW5
https://www.adobe.com/devnet/flashplayer/articles/socket_policy_files.html
https://www.ietf.org/rfc/rfc2246.txt - The TLS Protocol Version 1.0
https://www.ietf.org/rfc/rfc959.txt - FTP
https://www.ietf.org/rfc/rfc1035.txt - DNS
https://www.adobe.com/devnet/articles/crossdomain_policy_file_spec.html
http://msdn.microsoft.com/en-us/library/cc197955%28v=vs.95%29.aspx - Making a Service Available Across Domain Boundaries
http://help.adobe.com/en_US/as3/dev/WSb2ba3b1aad8a27b0-181c51321220efd9d1c-8000.html#WS5b3ccc516d4fbf351e63e3d118a9b90204-7cfb - Flash Sockets
https://www.torproject.org/projects/tordnsel.html.en
http://www.silverlightshow.net/items/Caching-of-in-and-around-your-Silverlight-application-part-1.aspx
http://help.adobe.com/en_US/AS2LCR/Flash_10.0/help.html?content=Part2_AS2_LangRef_1.html
http://livedocs.adobe.com/flex/3/html/help.html?content=17_Networking_and_communications_4.html
https://support.apple.com/kb/DL762 - QuickTime 7.6 for Windows
https://developer.apple.com/library/mac/#documentation/QuickTime/RM/Fundamentals/QTOverview/QTOverview_Document/QuickTimeOverview.html#//apple_ref/doc/uid/TP30000992-CH1g-QuickTimeOverview
https://developer.apple.com/library/mac/#documentation/NetworkingInternet/Conceptual/StreamingMediaGuide/Introduction/Introduction.html
http://wiki.pcbsd.org/index.php/Java,_Flash,_and_Fonts
http://livedocs.adobe.com/flashmediaserver/3.0/hpdocs/help.html?content=00000100.html
http://msdn.microsoft.com/en-us/library/cc645032(v=vs.95).aspx - Network Security Restrictions in Silverlight
http://msdn.microsoft.com/en-us/library/cc296248(v=vs.95).aspx - Silverlight Sockets
http://help.adobe.com/en_US/as2/reference/flashlite/WS5b3ccc516d4fbf351e63e3d118cd9b5f6e-78c7.html - Flash LoadVars
http://help.adobe.com/en_US/AS2LCR/Flash_10.0/help.html?content=Part2_AS2_LangRef_1.html - XMLSocket
https://en.wikipedia.org/wiki/Real_Time_Messaging_Protocol
http://docs.oracle.com/javase/6/docs/api/java/net/URLConnection.html
http://msdn.microsoft.com/en-us/library/system.windows.messaging.localmessagereceiver.anydomain(v=vs.95) - Silverlight AnyDomain
http://msdn.microsoft.com/en-us/library/dd920295(v=vs.95).aspx - Silverlight Client HTTP
http://msdn.microsoft.com/en-us/library/cc838250(v=vs.95).aspx - Silverligth Security
http://timheuer.com/blog/archive/2008/09/24/silverlight-isolated-storage-caching.aspx
http://msdn.microsoft.com/en-us/magazine/dd434650.aspx - Managing Dynamic Content Delivery In Silverlight, Part 2

Web Browser Features and Fingerprinting

http://msdn.microsoft.com/en-us/library/ie/ms531416%28v=vs.85%29.aspx - clientCaps
http://blogs.msdn.com/b/ie/archive/2012/03/02/web-platform-features-in-windows-consumer-preview.aspx

https://ie.microsoft.com/testdrive/Includes/Script/FeatureDetection.js
http://lcamtuf.coredump.cx/p0f3/
http://fingerprint.pet-portal.eu/?menu=1
http://ha.ckers.org/mr-t/mr-t.cgi
http://ip-check.info/
http://anon.inf.tu-dresden.de/help/jap_help/en/help/security_test.html
http://flippingtypical.com/ - CSS Font enumerations
http://msdn.microsoft.com/en-us/library/ie/ms535922%28v=vs.85%29.aspx - addBehavior
http://msdn.microsoft.com/en-us/library/ms531355%28v=vs.85%29.aspx - IE Components
https://developer.mozilla.org/en/DOM/The_structured_clone_algorithm
http://www.w3.org/TR/html5/common-dom-interfaces.html#safe-passing-of-structured-data
https://developer.mozilla.org/en/DOM/window.history
https://developer.mozilla.org/en/DOM/window.postMessage
http://www.whatwg.org/demos/workers/primes/page.html
https://developer.mozilla.org/En/Using_web_workers
http://www.html5rocks.com/en/tutorials/workers/basics/
http://webpolicy.org/2012/02/17/safari-trackers/
http://blog.kotowicz.net/2012/02/intro-to-chrome-addons-hacking.html
http://www.scatmania.org/2012/04/24/visitor-tracking-without-cookies/
https://dvcs.w3.org/hg/content-security-policy/raw-file/tip/csp-specification.dev.html
http://www.w3.org/TR/DOM-Level-3-Core/core.html#DOMFeatures
http://stackoverflow.com/questions/2281613/how-do-i-detect-the-difference-between-adobe-acrobat-versions-higher-than-8-in-n
http://modernizr.com/docs/#features-css
https://blog.torproject.org/blog/firefox-security-bug-proxy-bypass-current-tbbs
http://www.w3.org/TR/cors/ - Cross Origin Resource Sharing
http://www.howtocreate.co.uk/operaStuff/operaObject.html
https://trac.webkit.org/wiki/PrefixedAPIs
http://www.matthewratzloff.com/blog/2007/06/26/detecting-plugins-in-internet-explorer-and-a-few-hints-for-all-the-others/
http://msdn.microsoft.com/library/ie/hh673549.aspx
http://www.w3.org/TR/css3-mediaqueries/
https://developer.mozilla.org/en/Security/CSP/CSP_policy_directives
http://msdn.microsoft.com/en-us/library/ms537509%28v=vs.85%29.aspx - Detecting IE
https://blogs.msdn.com/b/giorgio/archive/2009/04/14/how-to-detect-ie8-using-javascript-client-side.aspx?Redirected=true
https://en.wikipedia.org/wiki/List_of_typefaces#Unicode_fonts
http://www.w3.org/TR/P3P/

Software

https://github.com/facebook/tornado
https://github.com/chr15m/pyPdf/tree/ae4a7bb94f998f9a320129ca289bd552efa2d76c
https://code.google.com/p/rtmplite/
http://sourceforge.net/projects/pydns/files/pydns/pydns-2.3.6/pydns-2.3.6.tar.gz/download?use_mirror=iweb