Analyzing the Crossdomain Policies of Flash Applications

Dongseok Jang
UC San Diego
d1jang@cs.ucsd.edu

Aishwarya Venkataraman
UC San Diego
avenkata@cs.ucsd.edu

G. Michael Sawka
Topix
sawka@cs.stanford.edu

Hovav Shacham
UC San Diego
hovav@cs.ucsd.edu

Abstract

Adobe Flash is a rich Internet application platform. Flash applications are often deployed to the Web; The Flash Player plugin is installed on a large fraction of all Web-connected PCs. Flash provides a mechanism by which sites can opt in to more expressive information sharing regimes than the same-origin policy for JavaScript allows. A site that wishes to share its content can host a *crossdomain policy file*, `crossdomain.xml`, which lists sites authorized to access the sharing site's content, or even a wildcard to allow all access. Because browsers will typically attach cookies to crossdomain URL requests made by the Flash Player plugin, a site that publishes a crossdomain policy effectively opts out from some of the confidentiality guarantees of the same-origin policy. In some cases, a misconfigured, overly permissive crossdomain policy can expose a site to attacks such as information disclosure or CSRF.

In 2008, Jeremiah Grossman surveyed the crossdomain policies of the Alexa Top 500 sites and the sites of the Fortune 500, and found that 7% hosted crossdomain policy files allowing unrestricted access. In this paper, we repeat Grossman's survey on a larger corpus of sites: the Alexa global Top 50,000 sites. In addition, we use an instrumented Firefox to survey the actual crossdomain requests issued by Flash content hosted on the front pages of the Alexa global Top 50,000 sites. Our survey provides new data about the use of Flash crossdomain policies on popular sites. For example, we find that approximately 6.0% of the surveyed sites allow unrestricted crossdomain access, including 12 sites in the Alexa Top 100, and that, at a minimum, 6.7% of crossdomain requests made by Flash applications we observed were denied by the target site's crossdomain policy.

Our findings suggest that Flash's crossdomain policy mechanism may be liable to misconfiguration in practice. We propose some techniques for mitigating the security problems that might arise from such misconfiguration.

1. Introduction

The same-origin policy. The same-origin policy [12] is at the heart of the browser security model. Without the same-origin policy's confidentiality and integrity guarantees, the Web applications running within a user's browser could interfere with each other. A malicious site a user visits could interact with other sites using that user's credentials, for example reading a user's online bank statements or issuing money transfers from his account.

Technically, the same-origin policy segregates Web resources by origin (hostname, protocol, and port tuple), allowing mutually distrusting applications to run in a user's browser. The isolation provided by the same-origin policy is not absolute: an application running in one domain can import a library from another domain (using the `script` tag) or export data to another domain (by including it in a GET or POST request). However, an application cannot read the response returned by the target of such data export, since this would allow cross-site request forgery attacks [6].

To enable mashups and other sophisticated Web 2.0 applications, browsers have implemented other ways for sites to cooperate beyond the limits of the same-origin policy. Client-side frame communication, like `postMessage` is one way [7]. Another is server-side configuration that allows sites to opt out of (some of) the isolation provided by the same-origin policy.

Flash crossdomain policy files. There are cases in which an application hosted on `a.com` wishes to access content hosted on `b.com` in a manner that would be ruled out by the same-origin policy. Crossdomain access policies allow the `b.com` server to opt in to this access. The client (e.g., the browser or the Flash plugin) downloads a policy from the `b.com` server, and allows the access if this policy authorizes crossdomain access from `a.com`. Flash, Java, and HTML5 all provide such mechanisms. The potentially harmful unintended interaction of these mechanisms with Web infrastructure has already been noted [11]. In this paper, we study the *intended* behavior of one of these mechanisms as deployed in the wild: Flash's crossdomain policy file, `crossdomain.xml`.

The Flash Player plugin is installed in a large fraction of all Web-connected PCs. Flash SWF files are often used as video containers, but can also encode rich application behavior using ActionScript, Flash's ECMAScript-derived scripting language.[1] For example, Flash applications can use the `URLLoader` class to make HTTP requests and read the content returned. By default, such requests must obey the same-origin policy, but servers can choose to allow more access through a crossdomain policy [4]. Flash crossdomain policy is encoded in an XML file named `crossdomain.xml`, hosted in a server's root directory (and, in certain cases, also in other files in subdirectories). The "`allow-access-from`" element is used to specify domains from which crossdomain requests are allowed; one such choice is "`*`", meaning that all requests are allowed.

Crucially, a user's cookies are usually attached to crossorigin HTTP requests that are made through ActionScript and authorized by crossdomain policy. More specifically, when a Flash application hosted at `a.com` makes an authorized crossorigin request to `b.com` by means of an ActionScript class like `URLLoader`, the Flash Player plugin requests the resource from the browser using an NPAPI call. The browser then uses its URL-fetching infrastructure to retrieve the `b.com` resource on behalf of Flash Player. Commonly used browsers attach any applicable cookies in their cookie file to the `b.com` request. This is the documented, expected behavior [1], and allows crossorigin requests to be associated with a session established earlier and identified by a cookie. What's more, this behavior comes about because of the way that browsers handle URL-requesting NPAPI calls, not because of choices made by the Flash Player plugin itself.

Nonetheless, allowing crossdomain access to a site that uses cookies for authentication may leak sensitive user data and may defeat token-based crosssite request forgery defenses. (This observation was already made at least as early as 2006, Shiflett and Couvreur [19, 8], and is not a contribution of our paper.) When `b.com` allows Flash crossdomain access from `a.com`, it gives up the confidentiality guarantees of the same-origin policy with respect to `a.com`. This is a problem if `a.com` is malicious (or compromised by attacks such as cross-site scripting): `a.com` will host a Flash application that makes crossdomain requests and will trick logged-in `b.com` users into rendering this file in their Flash-equipped browser. Note that this attack is possible even if `b.com` is never legitimately accessed through Flash, and hosts a permissive policy file by accident or as a holdover from previous usage.

Even more seriously, when `b.com` allows Flash crossdomain access from "`*`", *it gives up the confidentiality guarantees of the same-origin policy altogether*. Adobe warns that this "is a dangerous practice, and you should only grant access to '`*`' if you are certain that the scope of the policy file does not host any access-controlled, personalized, or private data" [4, Section 1.3.3.1]. From a security standpoint, allowing access from "`*`" is appropriate for specialized domains that act as content repositories (such as YouTube's `*.ytimg.com`), but not for other domains. But real-world configuration of crossdomain policy files may not reflect security best practices.

Our contribution. We perform a large-scale empirical study of crossdomain policy files in use by popular Websites. We retrieved the `crossdomain.xml` files of each of the Alexa global Top 50,000 domains (except where forbidden by `robots.txt`).[2] In addition, we used a modified Firefox browser to observe the actual crossdomain requests issued by Flash applications hosted on the front pages of Alexa Top 50,000 domains.

Speaking formally, our first measurement allows us to construct a portion of the "can-read" graph, in which vertices are origins and a directed edge from `b.com` to `a.com` means that `b.com`'s policy file grants crossdomain access to `a.com`; and our second measurement allows us to approximate the related "does-read" graph, in which a directed edge indicates a crossdomain access that occurs in practice. (In the absence of crossdomain access policies, both graphs would have no edges.) Nodes with high out-degree in the first graph could be attacked by many sites (in particular, sites allowing "`*`" access correspond to nodes with effectively infinite out-degree); nodes with high in-degree, if compromised, could be used to attack many sites. And differences between the two graphs are evidence of policy files that are either overly permissive or overly restrictive.

Our major result is that approximately 17% of sites in the Alexa Top 50,000 host a `crossdomain.xml` policy. Fully 36% of these (i.e., 6.0% of all 50,000 sites) allow "`*`" access. This includes many sites that are not pure content repositories of the sort envisioned for a "`*`" access policy.

We show that the risk of private information disclosure is not merely theoretical: we find sites with a "`*`" access policy that actually do use cookies for authentication and that maintain private information for logged-in users. As one concrete example, `scribd.com` (#234 on the Alexa list) discloses information about logged-in users at `http://www.scribd.com/info`, and `www.scribd.com` hosts a "`*`" crossdomain access policy. It is thus likely that attackers could use Flash crossdomain access to learn sensitive information about Scribd's users.

In addition, we find domains that are allowed to read 100 or more other domains; these typically host advertising or content services, and have access to some high-profile sites; for example, `*.brightcove.com` is allowed by the sites of

[1] See `http://www.adobe.com/devnet/actionscript/references.html` for ActionScript documentation.

[2] Here and elsewhere in this paper we use the Alexa list as of January 29th, 2011

```
<cross-domain-policy>
  <allow-access-from domain="sub.secret.com" />
  <allow-access-from domain="safe.com" />
</cross-domain-policy>
```

Figure 1. An example `crossdomain.xml` file.

the New York Times and ESPN, as well as 171 other sites in the top 50,000.

Related work. Shortly after first noting the possibility of Flash-based crossdomain CSRF, Chris Shiflett noted the possibility of a "witch hunt" of sites with permissive crossdomain access policies [18], and pointed to a site, `crossdomainxml.org` (now offline), that tracked the use of crossdomain policies in popular sites.

That same month (October, 2006), Jeremiah Grossman surveyed the use crossdomain policies in the Alexa Top 100 sites and Fortune 500 company sites [9]. He found that 6% of the Alexa Top 100 and 2% of the Fortune 500 company sites allowed "*" access. In May, 2008, Grossman repeated his survey, extending it to the Alexa Top 500 and Fortune 500 [10]. Grossman's second survey found that 7% of these sites (combined) allowed "*" access. Our study confirms Grossman's findings, showing that, three years later, 10% of the Alexa Top 500 sites allow "*" access.

Simultaneously with our survey, Kontaxis et al. [14] surveyed crossdomain policies for Flash and Microsoft's Silverlight on the Alexa global Top 100,000 sites, as well as the Fortune 500 company sites, and 500 top sites for each of several countries. They find that approximately 7% of the Top 100,000 sites host a "*" policy, and also describe differences in crossdomain policy use between countries and depending on site rank.

Like the survey of Kontaxis et al., ours covers a much larger set of sites than Grossman's surveys. Though we gather much the same data as Kontaxis et al., our analysis focuses on other features; in addition, we use a modified browser to study actual crossdomain policy requests issued by Flash applications.

Threat model. Except where we state otherwise, we assume the standard Web attacker threat model [7]. A Web attacker has a domain from which she can serve HTTP and HTTPS traffic (with a valid SSL certificate), and to which she can drive traffic, for example through ads or spam. A few of our attacks require the attacker to be able to upload certain kinds of content to otherwise-trusted servers. The attacker might have an account on a blogging service, for example.

Lessons. We draw two broader lessons from our experimental study. First, it is dangerous for a single file, if misconfigured, to have a drastic effect on the security of a site — even a site that does not itself rely on Flash for functionality. Accidental misconfiguration may go undetected by everyone but attackers. Second, whitelist access control mechanisms like `crossdomain.xml` will tend towards overpermissive configurations in practice, enabling functionality at the (invisible) expense of security. This is not a new lesson; writing in 2000, Zwicky, Cooper, and Chapman [22, p. 508] describe a similar problem with the X Windowing System's `xhost` mechanism:

> Users tend to forget to preauthorize the host before starting the clients, which are then refused access. After this happens a few times, many users disable the control altogether. For example, they issue an `xhost +` command to allow connections from any and all hosts in the name of convenience (so they can easily run programs on remote systems), without giving any thought to the security implications of their actions.

Another such mechanism is the `rhosts` file that controls access in the `r*` suite of protocols; misconfigured `rhosts` files were used in 1988 by the Morris worm as a propagation vector [17]. Compared to `xhost` and `rhosts`, misconfigured `crossdomain.xml` files have the additional disadvantages of, first, affecting not just the site's administrators but its users, and, second, being easily discoverable, as we have shown in our survey.

Ethics. In both our survey of crossdomain policy files and of Flash applications, we respected sites' `robots.txt` policies, and attempted to avoid placing unnecessary load on the sites. We confirmed that cookies are attached to Flash crossdomain requests by modern browsers using only servers under our control. We did not make any crossdomain requests to any of the sites in our survey or otherwise attempt to probe any vulnerabilities.

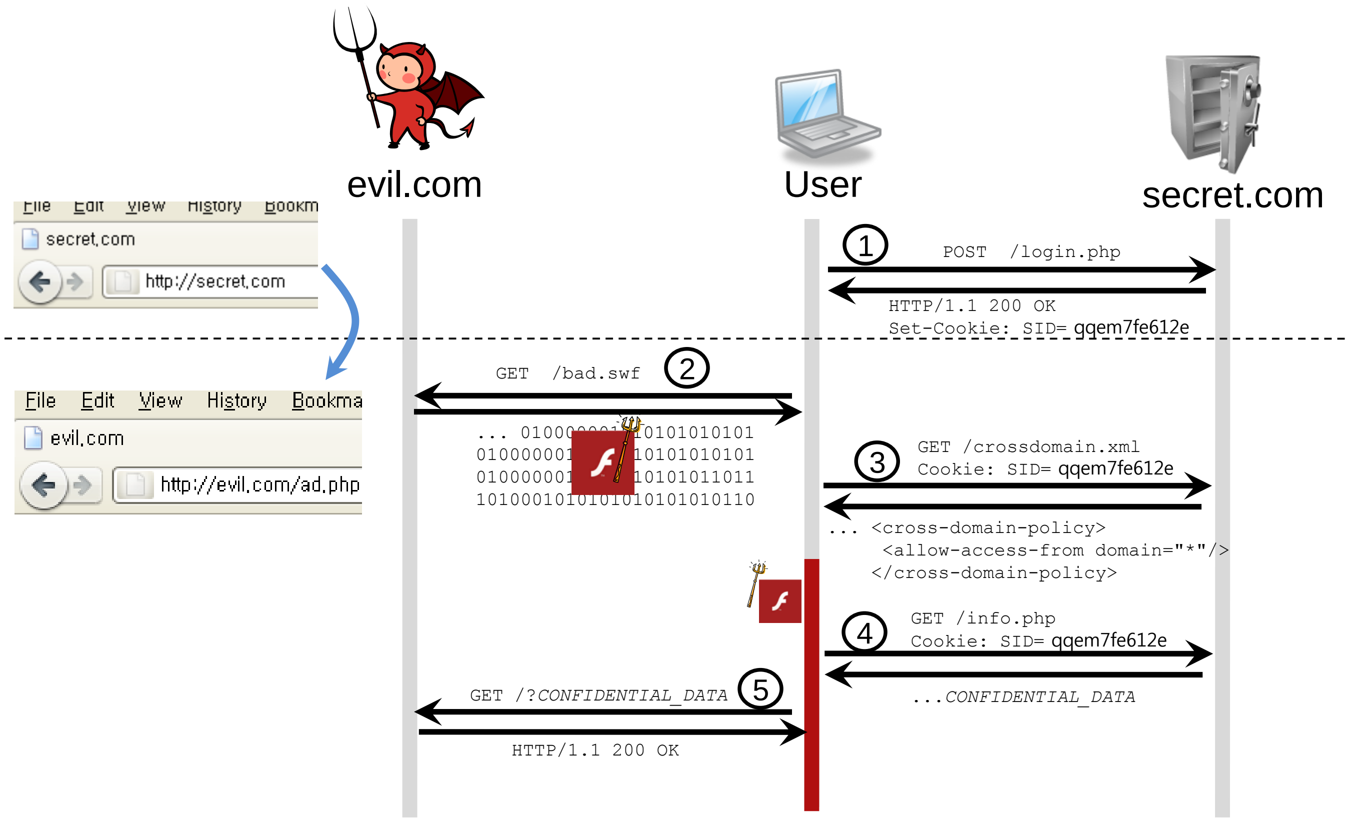
Prior to publication, we discussed our results with Adobe. Where feasible, we also hope to alert sites that host overly permissive crossdomain policy files, though this is difficult given the number of affected sites.

```
0:<cross-domain-policy>
1:  <allow-access-from domain="*.secret.com" />
2:  <allow-access-from domain="0.0.0.0" />
3:  <allow-access-from domain="safe.com"
                                  secure="false"/>
4:</cross-domain-policy>
```

Figure 2. An example `crossdomain.xml` file with various ways of naming domains.

2. Flash Crossdomain Policies

Cross-domain policies. Flash's crossdomain policy was introduced in version 7 of Flash Player. Its goal was to harmonize Flash's default access policy with the JavaScript same-origin policy while allowing sites to opt for more expressive access policies, including the default policy in



```
POST /login.php
HTTP/1.1 200 OK
Set-Cookie: SID= qqem7fe612e
GET /bad.swf
... 0100000001 0101010101
0100000001 10101010101
0100000001 10101011011
1010001010101010101010110
GET /crossdomain.xml
Cookie: SID= qqem7fe612e
... <cross-domain-policy>
     <allow-access-from domain="*"/>
    </cross-domain-policy>
GET /info.php
Cookie: SID= qqem7fe612e
...CONFIDENTIAL_DATA
GET /?CONFIDENTIAL_DATA
HTTP/1.1 200 OK
```

Figure 3. Example attack exploiting overly permissive crossdomain policy. 1: User logs into `secret.com`. 2: Later, the user is tricked to retrieve and execute a malicious Flash application embedded in a Web page on `evil.com`. 3: The Flash application on `evil.com` attempts to read data from `http://secret.com/info.php`. Flash Player fetch the `crossdomain.xml` policy file of `secret.com` to check whether `secret.com` allows for crossdomain requests from `evil.com`. Since the fetched `crossdomain.xml` file specifies unrestricted crossdomain policy, Flash Player allows the Flash application to issue the crossdomain request. 4: The Flash application on `evil.com` issues the crossdomain request to `http://secret.com/info.php` to which the browser attach the user's active session cookie. The requested Web page, `http://secret.com/info.php`, renders content containing the user's confidential information based on the provided session cookie, and the content is sent back to the Flash application. 5: The malicious Flash application sends the confidential data back to `evil.com`

previous versions of Flash Player [5]. In prior versions, Flash Player enforced a variant of the JavaScript same-origin policy under which a Flash application on a subdomain could read data from its parent domain. For example, a Flash application residing on `sub.a.com` could read a Web page residing on `a.com`. By contrast, JavaScript considers `sub.a.com` and `a.com` to be distinct domains.

Versions of Flash Player beginning with Flash Player 7 enforce the same origin separation as JavaScript. For backwards compatibility with Flash applications reliant on the old behavior, and to allow for more expressive crossdomain access policies, Flash Player 7 allowed sites to define custom crossdomain policies. When a Flash application tries to read data from another domain, the Flash Player consults the `crossdomain.xml` file on the domain. This XML file specifies what crossdomain access is allowed; Flash Player allows the access only if it is consistent with policy. With `crossdomain.xml` files, Flash applications can communicate between domains by explicitly specifying crossdomain policies, and can maintain backward compatibility by allowing access from subdomains.

Figure 1 shows an example `crossdomain.xml` residing on the imaginary Website `secret.com`. The policy file authorizes two types of crossdomain access, from `sub.secret.com` and from `safe.com`. The first rule might be in support of legacy Flash applications expecting the access policy from Flash Player 6; the second might allow access from a trusted affiliate. The `crossdomain.xml` file in

Figure 2 illustrates a variety of possible rules. When specifying a rule, one can use a wildcard character (line 1) and even an IP address to name a domain (line 2). Moreover, a site served over HTTPS can allow crossdomain access by Flash applications served over HTTP by using a "`secure`" attribute set to the value "`false`" (line 3).

Attack vectors. Websites might be open to serious attacks if they host overly permissive crossdomain policy files. If `b.com` allows crossdomain access from `a.com`, then a malicious Flash application hosted on `a.com` can mount information-disclosure attacks on `b.com`'s users, by causing those users' browsers to make requests for pages on `b.com` that include sensitive information and exfiltrating that information. Figure 3 shows a concrete example. Consider a Website, `secret.com`, that hosts a `crossdomain.xml` file stating that `evil.com` is allowed to make crossdomain requests to `secret.com`. Suppose there is a Web page, `http://secret.com/info.php`, contains confidential information and hence is only accessible to a user with a valid login session identified by the user's cookie value. If the attacker can trick a user logged in to `secret.com` to visit `evil.com`, she can serve to that user a malicious Flash application that uses Flash's facilities to read the contents of `http://secret.com/info.php`. Flash Player will allow this query, since it is authorized by `secret.com`'s crossdomain policy, and the browser will fulfill it, attaching the user's `secret.com` cookies to the HTTP request. The contents of the sensitive page will be disclosed to the malicious Flash application, which can send it to the attacker's server.

One kind of sensitive information that could be disclosed by this mechanism is the CSRF tokens intended to prevent cross-site request forgery attacks [6]; such an attack would allow the malicious Flash application to manipulate the state of the user's session on `secret.com`. This possibility was noted in 2006 by Shiflett and Couvreur [19, 8].

If an intranet Web server hosts an overly permissive crossdomain policy file, an attacker who convinces someone within that intranet to visit her site can use a Flash application to exfiltrate data from the intranet server. This is similar to DNS rebinding attacks [13]. Unlike the attacks considered above, this attack on intranet servers does not depend on the fact that current browsers attach cookies to crossdomain requests made by Flash applications.

3. Crossdomain Policies on Popular Websites

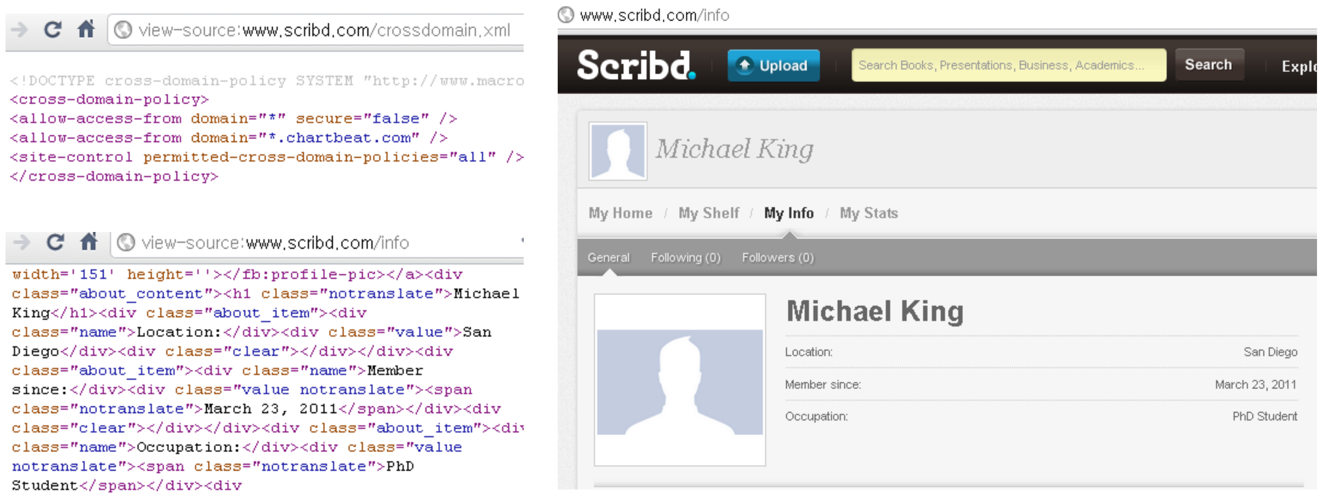
To understand the Flash crossdomain policies that are deployed in the wild, we carried out a survey of the (top-level) crossdomain policy files published on 50,000 high-traffic sites.

To undertake our crawl, we used a machine with a 3.20GHz Pentium 4 processor, running the Ubuntu 10.04.2 ("Lucid Lynx") distribution of Linux. The crawl started on March 23, 2011 and was completed on March 25, 2011, taking 42 hours overall.

Rank	Site	Description	HTTP
9	`qq.com`	Chinese news	-
47	`youku.com`	Chinese video	-
48	`tudou.com`	Chinese video	-
51	`xvideos.com`	adult	-
61	`pornhub.com`	adult	-
65	`about.com`	online resource	-
68	`zedo.com`	advertising	-
77	`youporn.com`	adult	Y
81	`ifeng.com`	Chinese news	-
87	`imageshack.us`	media hosting	-
88	`rapidshare.com`	file sharing	Y
99	`ehow.com`	Q&A	-
115	`cpxinteractive.com`	advertisement	Y
139	`ku6.com`	Chinese news	-
153	`56.com`	news	-
169	`imgur.com`	image sharing	-
179	`reference.com`	online learning	-
184	`liveinternet.ru`	news	-
191	`bild.de`	news	-
192	`onet.pl`	Polish news	-
193	`archive.org`	web archive	-
220	`foxnews.com`	news	-
229	`yfrog.com`	image sharing	-
234	`scribd.com`	online reading	Y
239	`kaskus.us`	forum	-
244	`seesaa.net`	Japanese blog	-
246	`adultfriendfinder.com`	adult	-
249	`soufun.com`	housing	-
252	`hardsextube.com`	adult	-
253	`metacafe.com`	video sharing	-
272	`people.com.cn`	news	-
274	`odesk.com`	job search	-
289	`slideshare.net`	sharing	-
303	`keezmovies.com`	entertainment	-
315	`ign.com`	entertainment	-
339	`verycd.com`	entertainment	-
356	`mashable.com`	social media	-
367	`7k7k.com`	entertainment	-
371	`mercadolivre.com.br`	shopping	-
385	`bigpoint.com`	online gaming	-
389	`ero-advertising.com`	advertisement	-
419	`imagebam.com`	image hosting	-
436	`slutload.com`	adult	-
455	`tradedoubler.com`	marketing	-
469	`cntv.cn`	entertainment	-
472	`vancl.com`	shopping	-
480	`mynet.com`	Turkish news	-
482	`wunderground.com`	weather	-
484	`bloomberg.com`	financial news	-
486	`tnaflix.com`	adult	-

Table 1. Top 50 sites hosting "`*`" crossdomain policies.

Surveyed sites. In order to evaluate the nature of crossdomain policies used in the wild, we collected top-level `crossdomain.xml` files from the Alexa global Top 50,000 Websites as of January 29th, 2011. Each Alexa top-site entry specifies a domain; for example, `google.com` and `facebook.com` are the two top sites. For each Alexa entry "`example.com`" we queried the server "`http://example.com:80/`". An equally valid choice for


view-source:www.scribd.com/crossdomain.xml

```
<!DOCTYPE cross-domain-policy SYSTEM "http://www.macro
<cross-domain-policy>
<allow-access-from domain="*" secure="false" />
<allow-access-from domain="*.chartbeat.com" />
<site-control permitted-cross-domain-policies="all" />
</cross-domain-policy>
```

view-source:www.scribd.com/info

```
width='151' height=''></fb:profile-pic></a><div
class="about_content"><h1 class="notranslate">Michael
King</h1><div class="about_item"><div
class="name">Location:</div><div class="value">San
Diego</div><div class="clear"></div></div><div
class="about_item"><div class="name">Member
since:</div><div class="value notranslate"><span
class="notranslate">March 23, 2011</span></div><div
class="clear"></div></div><div class="about_item"><di
class="name">Occupation:</div><div class="value
notranslate"><span class="notranslate">PhD
Student</span></div><div
```

www.scribd.com/info
Scribd.
Upload
Search Books, Presentations, Business, Academics...
Search
Expl
Michael King
My Home / My Shelf / My Info / My Stats
General
Following (0)
Followers (0)

Michael King

Location:	San Diego
Member since:	March 23, 2011
Occupation:	PhD Student

Figure 4. Crossdomain policy on `www.scribd.com` (Upper left), the content of a user's information page (Lower left), and the screenshot of the page (Right).

host, "`www.example.com`", might give different results, but we do not have data to compare. This is a limitation of our survey. Our crawler did follow any HTTP redirects from `example.com` to `www.example.com`, however. Considering "`www.example.com`" would not have helped if a site's important content is hosted on another subdomain, e.g., "`cdn.example.com`". We did not attempt to make HTTPS connections in addition to HTTP connections. Nor did we attempt to gather sub-path policy files in addition to the root-level policy file.

Policy file crawler. We implemented a Python program that automatically crawls `crossdomain.xml` files on these Websites. The crawler first examines the `robots.txt` file that specifies these sites' robots exclusion policy; it proceeds to request `crossdomain.xml` only if `robots.txt` allows it. We did not examine `crossdomain.xml` files on 1,049 sites out of the 50,000 on the Alexa list because of `robots.txt` restrictions, leaving 48,951 sites to survey. We do not know whether these sites host crossdomain policy files. (Unlike us, of course, an attacker would not be deterred by `robots.txt`.)

Websites with crossdomain policy files. We found 8,264 Websites hosting `crossdomain.xml` files. This is 16.88% of the 48,951 sites for which we attempted to retrieve policy files. Out of the 8,264 Websites having `crossdomain.xml` files 7,686 specified "`allow-access-from`" rules and 1,241 specified "`allow-http-request-headers-from`" rules.

Websites with unrestricted policy. Out of 8,264 Websites with `crossdomain.xml` files, we found that 2,993 Websites allow unrestricted crossdomain requests from any domain by describing permitted domains as a wildcard ("*"). Table 1 shows the top 50 ranked Websites among these sites. The columns in the tables are as follows: "Rank" is the Website's Alexa rank; "Site" is exactly the name of the domain from which `crossdomain.xml` is collected through the HTTP protocol; "HTTP" is whether the policy allows a Flash application served via HTTP from another domain to access the site served in HTTPS: "Y" indicates that it is allowed and "-" indicates that it is disallowed.

Publishing an unrestricted policy is potentially dangerous when Web pages containing users' private information are hosted under the same domain. For example, on `http://www.scribd.com` (which hosts the same unrestricted-access crossdomain policy as does `http://scribd.com`), the page `http://www.scribd.com/info` presents a logged-in user's name, location, and occupation. Since Scribd allows crossdomain Flash requests from any other domain, a user who encounters a malicious Flash application while logged in to Scribd is potentially subject to an information disclosure attack of the sort described at the end of Section 2. Figure 4 shows the unrestricted crossdomain policy of `http://www.scribd.com`, the content of `http://www.scribd.com/info` when a user is logged in, for a sample user account we created. (We stress that we have not attempted to make crossdomain requests to Scribd. There may be server configuration of which we are unaware that defeats the potential attack described above.)

Sites granted access in many `crossdomain.xml` files. We found that a number of domains are allowed to access from many other Websites through Flash crossdomain requests. Table 2 shows the ten domains most frequently listed in `crossdomain.xml` files on the Alexa Top 50,000 sites. Flash applications on the domains in Table 2 can read data from many other Websites. The sites in this table may thus

Domain	Description	Sites and Count
`*.brightcove.com`	advertisement	`espn.go.com`, `nytimes.com`, `weather.com`, `guardian.co.uk`, `wsj.com` + 188
`*.cooliris.com`	content	`cnet.com`, `download.com`, `wsj.com`, `reuters.com`, `t-online.de` + 140
`*.doubleclick.net`	advertisement	`cnn.com`, `espn.go.com`, `doubleclick.com`, `wsj.com`, `livingsocial.com` + 138
`*.2mdn.net`	advertisement	`cnn.com`, `espn.go.com`, `doubleclick.com`, `wsj.com`, `livingsocial.com` + 110
`localhost`	-	`kooora.com`, `justin.tv`, `enet.com.cn`, `nhl.com`, `yellowpages.com` + 100
`*.facebook.com`	social	`espn.go`, `nba.com`, `tripadvisor.com`, `usatoday.com`, `miniclip.com` + 61
`*.doubleclick.com`	advertisement	`espn.go.com`, `wsj.com` ,`livingsocial.com`, `associatedcontent.com` + 54
`*.aol.com`	content	`aol.com`, `cnn.com` `wsj.com`, `engadget.com`, `people.com` + 52
`*.floq.jp`	entertainment	`msn.com`, `ameblo.jp`, `ameba.jp`, `livedoor.biz`, `yaplog.jp`, `sony.jp` + 47
`*.livedoor.com`	content	`livedoor.com`, `livedoor.biz`, `2chblog.jp`, `alfalfalfa.com`, `ldblog.jp` + 44

Table 2. Domains that are allowed to issue crossdomain requests as mentioned in `crossdomain.xml`. “Count” stands for number of Websites that allow these domains to read data from them.

Rank	Site	Domains Allowed and Count
31948	`orange.co.il`	`*.alarab.net`, `*.karemmatar.com`, `*.ispot.co.il` + 414
31773	`shootq.com`	`*.2ndsunphoto.com`, `*.32`, `*.mintandsage.com`, `*.altf.com` + 291
11482	`bestcoolmobile.com`	`6060.com.ua`, `60in60.com.ua` `60v60.com.ua`, `adultinlove.com` + 290
27781	`viamichelin.com`	`*.serving-sys.com`, `a69.g.akamai.net`, `eyeblasterwiz.com`, `www.mcdonalds.com` + 279
12086	`warriordash.com`	`*.ac`, `*.as`, `*.ai`, `*.bf` + 260
5021	`hooverwebdesign.com`	`eliomstudio.com`, `www.taboosf.com`, `sytrex.com` + 217
34277	`qantas.com.au`	`*.yahoo.com`, `*.stuff.co.nz`, `*.weather.com.au`, `*.theage.co.au` + 212
13105	`nissan.co.jp`	`s-miyagi.co.jp`, `akita-nissan.co.jp`, `www.ask-nissan.co.jp`, `www.n-23.com` + 196
26848	`airberlin.com`	`aeroberlin.aero`, `aero-berlin.net` `airberlin.be`, `www.airberlin.at` + 187
12467	`badoo.com`	`*.badoo.ae`, `*.badoo.af` `*.badoo.ch`, `*.badoo.by` + 178

Table 3. Sites that allow crossdomain requests from more than 100 other sites. “Domains Allowed and Count” lists a few domains allowed in `crossdomain.xml` and gives the number of additional domains elided. The entries are sorted by this column.

make attractive targets for attackers: Flash content injected into one of these sites might allow information disclosure or CSRF bypass attacks on the sites that grant it access.

Sites whose `crossdomain.xml` files include many entries. We noticed that some sites host elaborate policies, allowing crossdomain access from hundreds of other sites. Some sites hosting elaborate policies are listed in Table 3. In all, we found 778 sites that allow crossdomain requests from more than 10 sites. While such policies are certainly no worse than “`*`” policies, they may still have risks. For example, `qantas.com.au` allows crossdomain access from 216 sites, one of which is `*.blogspot.com`. If it were possible for an attacker with a Blogger account to upload malicious Flash applications to a `blogspot.com` blog (and have these applications served from `*.blogspot.com`), then Qantas users who visit this blog might be put at risk. To the best of our knowledge, Blogger does not currently allow such SWF uploads.

The `secure` attribute The “`secure`” attribute of the “`allow-access-from`” element specifies whether access to an HTTPS resource should be granted only to Flash applications served over HTTPS (`secure=true`, the default) or also to those served over HTTP (`secure=false`) [5]. An HTTPS site setting `secure=false` for some crossdomain access opts out of SSL’s confidentiality guarantees relative to a network attacker. It appears that there is some confusion about the use of this attribute. For example, we noticed the following comment in the crossdomain policy file of one site setting `secure=false`:

> Note: `secure=false` is confusing, but basically its [*sic*] saying to allow SSL connections. Their reasoning is something about SSL content being made less secure somehow. Go look it up if you want to know more than this.

The “`secure`” attribute is not relevant for `crossdomain.xml` files served over HTTP, like all the ones we surveyed. Nevertheless, 1,424 crossdomain policy files we saw featured the “`secure`” attribute. A list of the top such sites is presented in Table 4. We did not attempt to verify that these sites serve the same policy over HTTPS, or, indeed, that they have HTTPS servers at all.

Port access for socket communication. Flash applications can communicate using raw TCP socket connections as well as HTTP. Such connections must be authorized by a socket

Rank	Site
4	yahoo.com
23	ebay.com
52	cnn.com
59	mediafire.com
66	espn.go.com
67	ameblo.jp
75	ebay.de
77	youporn.com
80	cnet.com
88	rapidshare.com

Table 4. Websites that grant access to HTTPS documents from Flash applications served over HTTP, by setting "secure" for "allow-access-from" to "false".

Rank	Site	Allowed Ports
31	mail.ru	80
36	vkontakte.ru	80
66	espn.go.com	*
67	ameblo.jp	*
88	rapidshare.com	80
123	mixi.jp	80, 8080
171	espncricinfo.com	*
179	reference.com	*
194	ameba.jp	*
246	adultfriendfinder.com	*

Table 5. Types of port access allowed in some Websites.

policy file on the target machine, served either from a master socket policy server (port 843) or from the port being connected to in response to a "<policy-file-request/>" XML request [15]. In either case, the retrieved policy must authorize the connection: Some "allow-access-from" element in the socket policy must name both the source domain (i.e., the domain hosting the Flash application) and, using the "to-ports" attribute, the target port (i.e., the server port being connected to).

Policy files served over HTTP (port 80) are not consulted by Flash Player for authorizing socket connections. Accordingly, the "to-ports" attribute is not relevant for the crossdomain.xml files we surveyed, which were all served over HTTP. Nevertheless, 537 crossdomain policy files we saw had a "to-ports" attribute specified; of these, 326 sites had "to-ports" set to "*", the most permissive value. A list of top sites whose crossdomain.xml files include the "to-ports" attribute is presented in Table 5. In the absence of a robots.txt-like mechanism for excluding unwanted access, we did not attempt to ascertain that these sites actually host socket servers on the named ports.

Header configurations. Prior to version 9, Flash Player allowed Flash applications to set arbitrary custom headers on HTTP requests, except for a short blacklist of dangerous headers. Beginning with version 9, Flash Player allows sites to specify what custom headers they are willing to receive in requests from Flash applications, still subject to a blacklist [3, 2]. The "allow-http-request-headers-from" element in a site's crossdomain.xml file specifies the allowed headers. Though it shares the crossdomain.xml file with "allow-access-from," the "allow-http-request-headers-from" mechanism is orthogonal, and governs the headers that Flash applications can send in requests to the site rather than whether they can read the site's response.

The allowed headers we saw in our survey and their frequency are listed in Table 6. Next to the wildcard "*", the most commonly allowed header is "SOAPAction", used for Web services. We were able to find the description of a public Web service for one site that accepts the "SOAPAction" header: weather.gov provides SOAP API access to the National Digital Forecast Database (cf. http://www.weather.gov/xml/).

The allow-access-identity element. We did not encounter any instances of the "allow-access-identity" element in our survey. This element, which grants read access based on cryptographic identity rather than domain, is supported by Adobe Acrobat but not by Flash Player.

Sub-path crossdomain policy files. In addition to the root policy file at /crossdomain.xml, sites can choose to specify policies that apply to resources within a specific directory and its subdirectories. For example, a sub-path crossdomain policy served from /secret/ controls access only to documents for which /secret/ is a path prefix. Sub-path policy files allow finer-grained control over crossdomain resource sharing than a single root policy. Properly deployed, sub-path policy files reduce a site's attack surface.

To enable sub-path policy files, a site must specify an appropriate metapolicy. Metapolicy is specified in the root policy file, /crossdomain.xml, by means of the "site-control" element's "permitted-cross-domain-policies" attribute. If this attribute is set to "none", then no crossdomain rules are allowed, even in the root policy file. If it is set to "master-only" then rules are allowed in the root policy file, but nowhere else. When "permitted-cross-domain-policies" is set to "by-content-type", sub-path policy files are allowed if served with a Content-Type of text/x-cross-domain-policy; when it is set to "all", sub-path policy files are allowed if served with a text or XML Content-Type. As of Flash Player 10, the default setting for "permitted-crossdomain-policies" is "master-only". (Servers can also use an HTTP header to specify metapolicy.)

Of these options, "all" is the riskiest. An attacker who has a user account on the server might be able to upload a file to the server that will be interpreted by Flash as a sub-path policy file, which would enable crossdomain access to resources within the directory to which the file was

Headers	Sites and Count
*	twitter.com, ebay.com + 1078
SOAPAction	usa.com, smh.com.au + 92
Authorization	dominos.jp, gillete.com + 4
X-Requested-With	demotyvacija.lt, balsas.lt + 1
Accept	paperlesspost.com, official.fm
i.travelpn.com.edgesuite.net	travelocity.com, travelocity.ca
pragma	tpu.ro, gustos.ro
none	thepetitionsite.com, care2.com
GET	svenskfast.se
Content-Type	dr.dk
X-Permitted-Cross-Domain-Policies	sinematurk.com
X-WSSE	multiply.com
http://www.myexperiencephoto.com/	fedex.com
Wanring	uefa.com
*.emusic.com	emusic.com

Table 6. Types of allowed headers, and some sites whose crossdomain policy allows those headers.

Sub-path Policy Attribute	Count
all	896
master-only	706
by-content-type	402
none	38
master_only	1
*.concordia.ca	1

Table 7. Metapolicies and their frequency of occurrence.

uploaded. This threat model does not apply to all sites, of course. Servers can also thwart such an attack, even assuming an "all" metapolicy, by disallowing user uploads that Flash would accept as crossdomain policy files, or by applying the "X-Permitted-Cross-Domain-Policies: none-this-response" header when serving user-uploaded content.

The metapolicies specified in the root policy files of the Alexa Top 50,000 Websites are summarized in Table 7. The column "Sub-path Policy Attribute" gives the value of "permitted-cross-domain-policies". The "site-control" element is not mandatory and thus was not present in the crossdomain.xml file of all Websites. We did not attempt to measure whether servers set metapolicy using HTTP headers, or whether any sites whose metapolicy allows sub-path policy files actually publish such files.

Sloppily configured policy files. We encountered some incorrect or sloppy crossdomain policy files, suggesting that the site's developers misread the crossdomain.xml specification. This state of affairs seems dangerous, given the impact that an incorrect crossdomain policy could have on site security.

A root crossdomain.xml file whose "site-control" element has the "permitted-cross-domain-policies" attribute set to "none" must not specify any access policy, either with "allow-access-from" or with "allow-http-request-headers-from". Nevertheless, we found several sites, such as pixelpipe.com and akihabaranews.com, that do specify crossdomain access rules in their policy files despite having "permitted-cross-domain-policies" set to "none".

In two examples that almost certainly represent misconfiguration, drf.com had the "permitted-cross-domain-policies" attribute set to "master_only" (with an underscore instead of hyphen) and another site, concordia.ca, had this attribute set to "*.concordia.ca".

Some Websites, such as vimeo.com and care2.com, have the "domain" attribute of an "allow-access-from" element and an "allow-http-request-headers-from" element set to "none", which actually implies that only the domain "none" can access the Websites. If the goal was to disallow crossdomain access, omitting "allow-access-from" and "allow-http-request-headers-from" elements altogether would have been a better approach.

An apparently mangled policy file we found was hosted on rollingout.com; it is shown in Figure 5.

4. Crossdomain Requests Issued by Flash Applications

We present an empirical study of the crossdomain requests issued by real Flash applications on popular Websites. Understanding the nature of crossdomain requests issued by real Flash applications can be helpful in interpreting the crossdomain policies published by sites, and in designing realistic defense mechanisms against attacks.

We carried out our study in three steps. First, we modified the Firefox browser to intercept and log external resource requests from the Flash Player plugin running inside the browser. Second, we ran the modified browser on the

```
<?xml version="1.0" encoding="UTF-8"?>
<cross-domain-policy>
  <allow-access-from domain="&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;?&gt;" />
  <allow-access-from domain="&lt;cross-domain-policy&gt;" />
  <allow-access-from domain="&lt;allow-access-from domain=&quot;&quot; /&gt;" />
  <allow-access-from domain="&lt;/cross-domain-policy&gt;" />
</cross-domain-policy>
```

Figure 5. Crossdomain policy file hosted on `rollingout.com`

front pages of the Alexa global Top 50,000 Websites and recorded the crossdomain requests issued by Flash applications embedded on these pages. Finally, we studied the log files to obtain statistical data.

Our modified Firefox. We modified the Firefox 3.6 browser to report every Netscape Plugin Application Programming Interface (NPAPI) call made by the Flash Player plugin. Since the Flash Player plugin makes NPAPI calls whenever it requests to access data through the HTTP or HTTPS protocols, our modified Firefox can intercept and log crossdomain requests by Flash applications. Specifically, Flash Player calls browserside NPAPI interfaces including `NPN_GetURL` and `NPN_GetURLNotify` for reading data through the HTTP GET method, and `NPN_PostURL` and `NPN_PostURLNotify` for HTTP POST.

Our modifications to Firefox consisted of 56 lines of logging code placed at the beginning of these NPAPI interface entrypoints. The logging code records the name of the NPAPI interface called, the URL of the page in which the calling Flash application is embedded, the URL of the Flash application itself, and the URL of the document requested. We did not attempt to evaluate the performance overhead of our logging code. However, there was no noticeable slowdown in interactive use.

A limitation of our approach is that it allows us to observe browser-mediated HTTP requests made by Flash applications but not raw socket communication. Because only the browser-mediated requests have cookies attached by the browser, this is sufficient for our purposes. Studying socket access by Flash applications is also interesting, but would require more intrusive sandboxing of Flash Player.

A more serious limitation of our approach is that the NPAPI calls made by the Flash Player plugin do not have a one-to-one correspondence with crossdomain requests made by Flash applications. On the one hand, if access to some resource is not allowed by a site's crossdomain policy then Flash Player, having downloaded the policy, will not proceed to request that the browser retrieve the disallowed resource. This means that disallowed crossdomain requests do not generate NPAPI calls. On the other hand, the Flash platform includes mechanisms (such as the `flash.display.Loader` class) by which Flash applications can load resources to display without being able to read their contents, much like HTML's `img` tag; Flash Player will request these resources using the same NPAPI calls as for requests that must comply with a crossdomain policy.

We deal with the ambiguities above as follows. First, we consider a request to be crossdomain if Flash Player consults a crossdomain policy file before making the request. That is, if a request for a resource at `example.com` was not preceded by a request for `example.com`'s `crossdomain.xml` file then the request is not crossdomain. Conversely, if Flash Player ever requests `example.com`'s `crossdomain.xml` file on behalf of a Flash application, then that application must have made at least one crossdomain request.

An NPAPI call requesting `example.com`'s crossdomain policy file that is not followed by any NPAPI calls requesting `example.com` resources is evidence that the Flash application requested one or more resources disallowed by `example.com`'s crossdomain policy. An NPAPI call requesting `example.com`'s `crossdomain.xml` file that is followed by n NPAPI calls requesting `example.com` resources, for $n \geq 1$, is evidence that the Flash application made at least one and at most n crossdomain requests that were allowed by policy. (Once a `crossdomain.xml` file is downloaded, it is cached for subsequent crossdomain requests to the Website.) Together, these observations allow us to compute a lower bound on the fraction of crossdomain requests made by Flash applications that were denied by policy.

An alternative way to decide whether a request is crossdomain is to compare the requested URL to the URLs of the Flash application and of the page in which it is embedded. (A Flash application from origin A embedded in a page from origin B is not thereby endorsed into origin B, so origin A must also be considered.) We expect that the this approach and our heuristic approach would give similiar results, but have not attempted to compare them.

Our corpus of Flash applications. To evaluate the behavior of crossdomain requests issued by Flash applications, we ran our modified Firefox on the front pages of the Alexa global Top 50,000 Websites. We automated this visit using a simple bash script that directs Firefox to visit each page. The script detects whether the browser has fully loaded a page by monitoring its network bandwidth usage. Once the network usage drops under 500 bytes per second, our script considers the Web page to be fully loaded and proceeds to the next page. Using the script, we successfully ran our mod-

Ref Count	Site
674	`i.ytimg.com`
601	`i2.ytimg.com`
580	`i4.ytimg.com`
578	`i3.ytimg.com`
550	`i1.ytimg.com`
407	`brightcove.vo.llnwd.net`
387	`c.brightcove.com`
344	`images.kontera.com`
288	`newschool.slideshowpro.com`
261	`api.dimestore.com`
223	`b.vimeocdn.com`
219	`l.player.ooyala.com`
195	`receive.inplay.tubemogul.com`
175	`pixel.quantserve.com`
169	`dc.tremormedia.com`
164	`log.adap.tv`
151	`krocms.kro.nl`
151	`i1.soundcloud.com`
148	`i.cartoonnetwork.com`
147	`caw.sv.us.criteo.com`

Table 8. Sites most frequently referenced by Flash applications we observed. "Ref Count" stands for the number of crossdomain requests targeted for each domain.

ified Firefox on these Websites in a total of about 74 hours on a fast network at a large university.

Crossdomain request statistics. We found that, out of 50,000 Websites, 8,746 (17.5%) host Flash applications that communicate through the network. The Flash applications we examined on these 8,746 Websites issued 102,169 HTTP/HTTPS requests through the NPAPI interface. Here multiple requests to the same URL are counted as distinct requests. Out of these 102,169 requests, 21,430 (21.0%) are crossdomain requests to a Website preceded by a request to the `crossdomain.xml` file of the Website. Even though our survey is limited to Flash applications embedded on the front pages of the top sites, it shows that crossdomain requests are frequently used in the wild.

We evaluated what Websites are most frequently referenced from other domains by the Flash applications we studied. Table 8 lists the 20 Websites that are most frequently referenced. The "Referred" column lists the number of crossdomain requests issued to each Website; "Site" is the name of the Website. Every Website listed in this table was accessed over HTTP protocol. Most of the Websites referenced from other domains are content distribution services like `*.ytimg.com` (for YouTube).

Disallowed crossdomain requests. The Flash applications we examined issued 10,565 requests to `crossdomain.xml` files on Websites in order to consult the crossdomain policy of the Websites before issuing a real crossdomain request. The fact that a request to a `crossdomain.xml` file of a Website is not followed by a subsequent requests to the Website implies that originally issued crossdomain requests were disallowed since they violated the policy described in the `crossdomain.xml` file. Out of the 10,565 requests to `crossdomain.xml` files, 1,545 (14.6%) requests were not followed by any subsequent requests to their target Website. Because there can be more than one request per target site, this 14.6% ratio does not measure the fraction of disallowed *request*. Using the heuristics described above, we can give a lower bound on this fraction. The number of requests to a `crossdomain.xml` file not followed by any further requests to the same domain — 1,545 — is a lower bound on the number of disallowed crossdomain requests. The 9,020 other requests to `crossdomain.xml` files were followed by a total of 21,430 crossdomain requests to documents other than the `crossdomain.xml` file; this is an upper bound on the number of allowed crossdomain requests. The fraction of crossdomain requests disallowed is thus at least $1{,}545/(1{,}545 + 21{,}430)$, or 6.7%. The actual fraction could be higher, but calculating it would require instrumentation of Flash Player rather than NPAPI interposition. Regardless, it appears to be the case that a significant number of crossdomain requests are disallowed in real Flash applications.

These disallowed crossdomain requests might be caused by misconfigured crossdomain policy files. It is also possible that Flash applications originally designed for running on a specific domain are sometimes used elsewhere without sufficiently careful consideration. Finally, it is possible (though unlikely) that these requests represent unsuccessful information disclosure attacks. Our current tool is unable to determine whether any of these reasons was responsible for the disallowed requests we observed. In addition, the nontrivial number of disallowed requests suggests that crossdomain requests are often unimportant or that error reporting facilities for deployed Flash applications are inadequate, since otherwise developers would have taken steps to address the failures.

5. Mitigation

Our survey, described in Section 3 suggests that overly permissive Flash crossdomain policies are quite common among popular sites. When governed by an appropriate policy, crossdomain requests allow for fine-grained information sharing between sites. And, as we showed in Section 4, Flash applications in the wild make frequent use of crossdomain requests. We therefore do not believe that the correct reaction to our findings is to seek to eliminate crossdomain requests. Instead, in this section we consider some mitigations that might make it more likely that site administrators will specify safe crossdomain policies, or else might reduce the harm that arises from any misconfiguration.

Crossdomain policy files should be audited. Most modestly, we suggest that crossdomain policy files should be audited as part of a site's security audits just as a firewall ruleset is (and for the same reason). Flash authoring tools should issue warnings when encountering overly permissive policies (e.g., those that allow "*" access).

Stripped cookies by default. Current browsers will attach cookies to any requests made by the Flash Player plugin on behalf of Flash applications. This is what enables the information disclosure and CSRF attacks discussed in Section 2 on sites with overly permissive crossdomain policies. If browsers were modified so as not to attach cookies to these requests, Flash applications would be able to access only public data for which cookie authentication is not required. (Intranet servers might remain vulnerable.) It is not clear that such a change is practical, however. Flash developers may be relying on the current behavior, in which case changing it will break deployed applications. Moreover, the current NPAPI interface may not allow browsers to reliably distinguish same-origin and crossorigin requests made by Flash Player; changing this interface would be difficult except perhaps where the Flash Player plugin is closely integrated with the browser, as in Google Chrome.

Stripped cookies as an option. As a compromise between always attaching cookies and stripping them, the crossdomain policy file format could be augmented to allow sites to express whether they would like cookies attached or not. This approach, like the one described above, has the drawback of requiring changes to the NPAPI interface. In addition, we have shown that administrators tend to install overly permissive crossdomain policies; given the option, they may opt to have cookies sent.

A more complicated but perhaps feasible approach would be to have Flash Player to bypass the browser and make HTTP and HTTPS requests directly, maintaining its own cookie file independent of the browser's. Where necessary, the browser-side and plugin-side cookies could be associated through communication between JavaScript, ActionScript, and the site. This approach would have serious disadvantages. It would require Flash Player to duplicate the browser HTTP stack; it might exacerbate DNS pinning attacks; and it would make tying a Flash request to a browser session much more difficult than now.

Taint tracking for protecting crossdomain content. To benefit an attacker, sensitive information obtained by means of a crossdomain request must then be sent over the network to a server she controls. Suppressing this sort of information transfer might mitigate information disclosure attacks, though it would not prevent CSRF token bypasses.

Dynamic taint tracking [20] is a promising way of controlling the propagation of information. Using taint tracking, a piece of confidential information is tagged with a special value called *taint*, and the taint is carried along with the information. By blocking tainted values from propagating to those program points that can leak it, we can protect confidential information.

Flash Player could be modified to apply taint to all values obtained through crossdomain requests,[3] and to refuse to transmit tainted values over the network (or at least alert the user, the targeted site, or Adobe). With this approach current `crossdomain.xml` files could be used more safely. However, the modifications required to Flash to support taint would be substantial, and may cause performance degradation. Depending on the taint tracking techniques used, false positive or false negatives may be a problem.

6. Conclusion

We have surveyed the crossdomain policy files on the Alexa global Top 50,000 sites. Furthermore, we have used a modified Firefox browser to survey the actual crossdomain requests made by Flash applications hosted on the front pages of these sites. Our survey provides new data about the use of Flash crossdomain policies on popular sites, suggesting that Flash's crossdomain policy mechanism may be liable to misconfiguration in practice. Such misconfiguration could lead to attacks including information disclosure and CSRF token bypass. We have proposed some techniques for mitigating the security problems that might arise from such misconfiguration.

There are many promising avenues for future work. First, it would be interesting to repeat our survey on a larger number of sites; and to consider sub-path crossdomain policy files as well as root policy files.

Second, it would be worthwhile to determine whether overly permissive configuration on specific sites in fact leads to exploitable vulnerabilities (and to do so automatically, rather than by manual inspection). It is possible that such vulnerabilities would have repercussions beyond the sites hosting the crossdomain policy. For example, some sites in our survey having "*" access policies support "Login with Facebook" authentication. To the extent that such sites reflect Facebook data for logged-in users, that data would potentially be at risk. More generally, as one of our anonymous reviewers asks, what is the interaction between crossdomain access and redirect-based federated identity systems?

Third, more extensive instrumentation of Firefox or Flash Player would allow for greater insight into the reasons that Flash applications make crossdomain requests, why so many appear to be disallowed by policy, and whether stripping cookies from crossdomain requests would break deployed applications.

Fourth, studies of other crossdomain policy mechanisms, such as the W3C's CORS [21] proposal, may be illuminating.

To facilitate future work, we hope to make our code and data available to other researchers.

[3] Tainting all values obtained by crossdomain requests may lead to over-tainting, but there is no reason to believe that developers will be able to specify which portions of their sites are sensitive and should be tainted.

Acknowledgments

We thank Peleus Uhley for his detailed feedback on the paper; Collin Jackson and Eric Rescorla for helpful discussions; and the W2SP reviewers for their comments.

This material is based upon work supported by the National Science Foundation under Grants No. CNS-0831532 and CNS-0964702, and by the MURI program under AFOSR Grant No. FA9550-08-1-0352.

References

[1] L. Adamski. Cross-domain policy file usage recommendations for Flash Player, Mar. 2007. Online: `http://www.adobe.com/devnet/flashplayer/articles/cross_domain_policy.html`.

[2] Adobe Systems. TechNote kb403185: Arbitrary headers are not sent from Flash Player to a remote domain, Mar. 2008. Online: `http://kb2.adobe.com/cps/403/kb403185.html`.

[3] Adobe Systems. TechNote kb403030: Actionscript error when an HTTP send action contains certain headers (Flash Player), Feb. 2009. Online: `http://kb2.adobe.com/cps/403/kb403030.html`.

[4] Adobe Systems. Cross-domain policy file specification, version 2.0, Aug. 2010. Online: `http://www.adobe.com/devnet/articles/crossdomain_policy_file_spec.html`.

[5] Adobe Systems. TechNote 14213: Cross-domain policy for Flash, Sept. 2010. Online: `http://kb2.adobe.com/cps/142/tn_14213.html`.

[6] A. Barth, C. Jackson, and J. Mitchell. Robust defenses for cross-site request forgery. In P. Syverson and S. Jha, editors, *Proceedings of CCS 2008*, pages 75–88. ACM Press, Oct. 2008.

[7] A. Barth, C. Jackson, and J. Mitchell. Securing frame communication in browsers. *Communications of the ACM*, 52(6):83–91, June 2009.

[8] J. Couvreur. Crossdomain.xml security warning, Sept. 2006. Online: `http://blog.monstuff.com/archives/000302.html`.

[9] J. Grossman. Crossdomain.xml statistics, Oct. 2006. Online: `http://jeremiahgrossman.blogspot.com/2006/10/crossdomainxml-statistics.html`.

[10] J. Grossman. Crossdomain.xml invites cross-site mayhem, May 2008. Online: `http://jeremiahgrossman.blogspot.com/2008/05/crossdomainxml-invites-cross-site.html`.

[11] L.-S. Huang, E. Chen, A. Barth, E. Rescorla, and C. Jackson. Talking to yourself for fun and profit. In H. J. Wang, editor, *Proceedings of W2SP 2011*. IEEE Computer Society, May 2011.

[12] C. Jackson and A. Barth. Beware of finer-grained origins. In L. Koved and D. S. Wallach, editors, *Proceedings of W2SP 2008*. IEEE Computer Society, May 2008.

[13] C. Jackson, A. Barth, A. Bortz, W. Shao, and D. Boneh. Protecting browsers from DNS rebinding attacks. In S. De Capitani di Vimercati and P. Syverson, editors, *Proceedings of CCS 2007*, pages 421–31. ACM Press, Oct. 2007.

[14] G. Kontaxis, D. Antoniades, I. Polakis, and E. P. Markatos. An empirical study on the security of cross-domain policies in rich Internet applications. In E. Kirda and S. Hand, editors, *Proceedings of EuroSec 2011*. ACM Press, Apr. 2011.

[15] D. Meketa. Policy file changes in Flash Player 9 and Flash Player 10, Oct. 2008. Online: `http://www.adobe.com/devnet/flashplayer/articles/fplayer9_security.html`.

[16] National Weather Service. SOAP Web service, Jan. 2011. Online: `http://www.weather.gov/xml/`.

[17] J. A. Rochlis and M. W. Eichin. With microscope and tweezers: the worm from MIT's perspective. *Communications of the ACM*, 32(6):689–98, June 1989.

[18] C. Shiflett. The crossdomain.xml witch hunt, Oct. 2006. Online: `http://shiflett.org/blog/2006/oct/the-crossdomain.xml-witch-hunt`.

[19] C. Shiflett. The dangers of cross-domain ajax with Flash, Sept. 2006. Online: `http://shiflett.org/blog/2006/sep/the-dangers-of-cross-domain-ajax-with-flash`.

[20] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas. Secure program execution via dynamic information flow tracking. In *ASPLOS*, pages 85–96, 2004.

[21] A. van Kesteren. Cross-origin resource sharing. Online: `http://www.w3.org/TR/cors/`, July 2010.

[22] E. Zwicky, S. Cooper, and D. B. Chapman. *Building Internet firewalls*. O'Reilly, 2nd edition, 2000.