The State of the Cross-domain Nation

Sebastian Lekies, Martin Johns and Walter Tighzert
SAP Research Karlsruhe
{firstname.lastname}@sap.com

***Abstract*—By deploying a configuration that allows the creation of client-side, cross-domain HTTP requests, a Web application weakens the same-origin policy. This enables sophisticated browser-based interaction which is not possible in the standard model, but also may lead to insecurities. In this paper, we briefly cover the technical background of client-side, cross-domain requests and explore the resulting potential security problems. Then, we present an extensive empirical study on observable cross-domain configurations and conduct an analysis of the collected data to assess the fraction of potentially vulnerable sites. For this purpose, we collected the cross-domain policies of 1.093.127 Web sites. The results of our analysis show that 2,8% of all examined sites are potentially insecure, including 15.060 sites for which an exploitable condition can be predicted with a high level of confidence.**

I. Introduction

The Web is ever changing. This constant evolution is driven by a steady stream of innovative Web application types or novel use-cases for existing functionality. In this progression of the field, upcoming application scenarios occasionally are hindered by the limits of the underlying technology and standards. This in turn fosters the further development of the Web server and browser model. Several times in the past, new capabilities were added to the mix via Web browser extensions or plug-ins, such as Adobe Flash or Google Gears, before they were adopted to become native browser features.

One example of such a development was the introduction of capabilities for client-side, cross-domain HTTP requests, which enable the cooperation of more than one Web application within the browser without server-side involvement. Such capabilities were first introduced by the Adobe Flash plug-in.

The security implications of this extension of the same-origin model of the web browser were discussed in the security community early on [5, 15, 16]. A small study of Jeremiah Grossman revealed in 2008 a percentage of 7% of the top 500 Web sites issue probably insecure cross-domain policy configurations [7]. Since then, only little evidence is present to assess to which degree the awareness of the potential security implications has reached the operators of modern Web sites.

In this paper, we present an empirical study on the current deployment of cross-domain policies and conduct an extensive analysis in respect to the observable level of security of these deployments. For this purpose, we analyze the cross-domain policies of the top 1.000.000 sites listed in the Alexa [3] index.

II. Technical background

In this section, we briefly revisit the technical background of client-site cross-domain request and explore the motivation that has led the introduction of this technique.

A. Client-side cross-domain HTTP requests

In general, the same-origin policy (SOP) [14] is the main security policy for all active content that is executed in a Web browser within the context of a Web page. This policy restricts all client-side interactions to objects which share the same origin. In this context, an object's origin is defined by the URL, port, and protocol, which were utilized to obtain the object. While this general principle applies to all active client-side technologies (e.g., JavaScript, Java applets, Flash, or Silverlight), slight variations in the implementation details exist. Please refer to [20] for further reference.

Based on the SOP, the initiation of network connections from active content in the browser is restricted to targets that are located within the origin of the requesting object[1]. This means, a JavaScript that is executed in the browser in the context of the origin `http://www.example.org` is only permitted to generate HTTP requests and read the response (via the XMLHttpRequest-object [17]) from URLs that match this origin. The same general rule exist for Flash, Silverlight, and Java.

However, in the light of increasing popularity of multi-domain, multi-vendor application scenarios and ever growing emphasis on client-side functionality, a demand for browser-based *cross-domain* HTTP requests arose (e.g., in the field of Web2.0 Mashups), as such request have the ability to mix data and code from more than one authorization/authentication context (see Sec. II-B).

Following this demand, the ability to create cross-domain HTTP requests from within the browser has been introduced by Adobe Flash [1], later followed by Sliverlight and the browser's native JavaScript (see Sec. II-C for technical details).

B. Use cases for client-side cross-domain HTTP requests

The need for client-side cross-domain requests is not immediately obvious. Alternatively, the Web application could offer a server-side proxying service that fetches the cross-domain content on the server-side, instead of requiring the client-side browser to issue the cross-domain requests. Such a service can be accessed by the client-side code via standard, SOP-compliant methods. This technique is offered for example by Google's Gadget API [6].

However, this technique is only applicable in cases in which the Web application's server-side code is able to access the

This work was in parts supported by the EU Project WebSand (FP7-256964), http://www.websand.eu.

[1]NB: This restriction only applies to HTTP requests that are created by active code. HTML elements, such as `IMG` or `IFrame` are unaffected and can reference cross-domain objects.

requested content, e.g, when requesting publicly available internet resources. Server-side proxies might not be applicable whenever the access to the requested content is restricted. Examples for such scenarios include situations in which the requested content is unavailable to the server because of network barriers (e.g., an internet Web application interacting with intranet data) or cases in which the demanded information is only available in the current authentication context of the user's Web browser (e.g., based on session cookies, see Sec. III for details). Such use cases can only be realized with client-side cross-domain capabilities.

C. Techniques for implementing client-side cross-domain requests

Since cross-domain requests can be abused for various different attacks (see Sec. III for details), Adobe introduced a policy-based security mechanism which was imitated by several other cross-domain technologies. This mechanism can be described as a server-side opt-in approach, where the server has to explicitly allow cross-domain communication to requesting domains. In order to do so, the server keeps a whitelist of domains in his policy file. When an application initiates a cross-domain request the responsible browser plugin downloads the policy file from the server and checks whether the calling domain is whitelisted or not. Only if this check is successful the request is conducted, otherwise the request is dropped. Besides full domain names (e.g. sub.domain.example.org) these policy files also use wildcards to grant cross-domain communication to several domains at once. *.example.org for example grants cross-domain access to all subdomains of example.org and a single "*" grants access to any domain. This is especially helpful for domains offering an API which is intended to be used by any other website. Under certain circumstances, however, having a "*" in a policy file can have severe security implications (see section III).

1) Adobe Flash: In order to allow cross-domain request of remote flash applets a server has to place a file called tt crossdomain.xml into the root folder of the web server. Listing 1 shows an exemplary cross-domain policy which contains the two main directives supported by flash. The allow-access-from-tag grants a domain to pull data from the web server, while the allow-http-request-header-tag allows a flash applet to push data onto the server in the form of http-request headers. The policy file at the server root is also called master-policy file and is valid for all files on the same domain. Sometimes it is desirable to grant access only to subfolders of a domain, thus, a flash-applet is able to specify the location of additional meta policies. A meta-policy can be placed anywhere on the server and is valid only for the directory it is situated in and all its subfolders. In order to avoid abuse of meta-policies the master policy has to grant the usage of such additional policies. The file in Listing 1 explicitly forbids other policies besides the master policy by specifying a site-control element.

2) Silverlight: Following Flash's example, Silverlight also implemented a policy-based recipient-opt-in approach to avoid the misuse of its cross-domain capabilities. It even uses Flash's `crossdomain.xml` as a fallback mechanism in case it cannot find its own policy file which is called `clientaccesspolicy.xml`. In contrast to Flash Silverlight only relies on one policy file and waives meta policies. This file however, can make use of more fine-grained policies and thus is also able to set access restrictions on the sub folder level. Listing 2 shows an exemplary client access policy. Each policy file can specify multiple policies consisting of a set of domains (<allow-from>...<\allow-from>) to which access is granted and a set of resources (<grant-to>...<\grant-to>) to which the cross-domain request can be send.

Listing 1 Exemplary crossdomain.xml file

```
<cross-domain-policy>
    <site-control
    permitted-cross-domain-policies=
                              "master-only" />
   <allow-access-from domain="a.net" />
   <allow-http-request-headers-from
    domain="a.net" headers="SOAP" />
</cross-domain-policy>
```

Listing 2 Exemplary clientaccesspolicy.xml file

```
<access-policy>
  <cross-domain-access>
    <policy>
      <allow-from http-request-headers="*">
        <domain uri="a.net" />
      </allow-from>
      <grant-to>
        <resource path="/"
          include-subpaths="true"/>
      </grant-to>
    </policy>
  </cross-domain-access>
</access-policy>
```

3) CORS: In the course of expanding JavaScript's networking stack, the ability to create cross-domain HTTP requests have been added. JavaScript's native implementation is called Cross-origin Resource Sharing (CORS) [18]. CORS utilizes JavaScript's XMLHttpRequest-object for this purpose. Instead of following Flash's example of utilizing policy files, CORS utilizes HTTP response headers to allow or deny the requests. Only if the HTTP response carries an allowing HTTP header, the received data is passed on to the calling script. This behavior allows much more fine-grained access control compared to the established policy-based mechanisms.

III. Security implications of client-side cross-domain HTTP requests

The security implications of client-side cross-domain capabilities are known in the security community ever since the introduction of this mechanism in Adobe Flash [5]. In this section, we briefly discuss the general security problems that can arise to motivate our survey.

A cross-domain policy expands the security perimeter of the issuing Web site through relaxing the applicable origin-based restrictions for the domain listed in the policy (this

Listing 3 Insecure crossdomain.xml file

```
<cross-domain-policy>
  <site-control
    permitted-cross-domain-policies="all" />
  <allow-access-from domain="*" />
  <allow-http-request-headers-from domain="*"
        headers="*" />
</cross-domain-policy>
```

topic is discussed further in Sec. III-D). In consequence, it can be stated that the likelihood of potential security implications grows with the number of domains that are permitted elevated cross-domain privileges. Therefore, policies that whitelist all other domains, using the "*"-wildcard, provide the most potential for abuse, as in such cases all Web sites, including sites controlled by an adversary, are outfitted with the relaxed restrictions.

A. Attacker model:

From now on, we consider the following scenario: The victim visits a Web site `b.net`, which is under the control of the attacker. This Web site is granted the right to create cross-domain requests to a second Web site `c.net`, for instance because `c.net` has an overly permissive `crossdomain.xml` policy file that whitelists ("*") all foreign domains. Furthermore, the victim currently possesses an authenticated session state with `c.net`, i.e., in the victim's browser exists a session cookie for this domain.

This setting allows the attacker to create arbitrary HTTP requests to `c.net` from within the victim's browser and read the corresponding HTTP responses. The victim's cookies for `c.net`, including the authenticated session cookies, are attached to these requests automatically by the browser, as the cookies' domain matches the target domain of the outgoing HTTP request. These requests are received and handled by `c.net` in the same fashion as regular requests coming from the victim, hence, they are interpreted in the victim's current authentication context.

B. Resulting malicious capabilities:

We can deduct the several potential attack vectors, based on the scenario discussed above.

1) Leakage of sensitive information [8]: The adversary can obtain all information served by `c.net` which the victim is authorized to access by simply requesting the information via HTTP and forwarding the corresponding HTTP responses to the attacker's server.

2) Circumvention of Cross-site Request Forgery protection [15]: The security guarantee of nonce-based Cross-site Request Forgery (CSRF) protection [4] is based on the assumption that the attacker is not able to obtain the secret nonce which is required for the server to accept the request. These nonces are included in the HTML of `c.net`. As the adversary is able to read HTTP responses from `c.net`, he can request the page from `c.net` that contains the nonce, extracting the nonce from the page's source code, and using it for the subsequent HTTP request.

3) Browser hijacking [12]: As discussed, the adversaries has the ability to create arbitrary HTTP requests that carry the victim's authentication credentials and read the corresponding HTTP responses. In consequence, this enables him to conduct attacks that are, for most purposes, as powerful as session hijacking attack which are conducted via cookie stealing: As long as the targeted application logic remains in the realm of the accessible domain (in our case `b.net`), the attacker can chain a series of HTTP requests to execute complex actions on the application under the identity of the user, regardless of CSRF protection or other roadblocks. This can happen either in a fully automatic fashion, as seen in the payloads of XSS worms [9, 11], or interactive to allow the attacker to fill out HTML forms or provide other input to his attacks. Frameworks such as BeEF [2] or MalaRIA [12] can be leveraged for the interactive case.

C. On the insecurity of "*"-policies

Unlike the widespread notion, a wildcard ("*") policy alone is not a sufficient precondition to classify a cross-domain configuration to be insecure. For a Web site which sole purpose is to serve public information. without the need for any state-full session management or user authentication, a general permissive policy is completely adequate.

Also, for sites which partition their functionality over several subdomains, having a permissive *-policy for some of the subdomains is legitimate (e.g., a site that serves static content via `static.b.net` and isolates its administrative functions on `admin.b.net` does not cause any security problem if `static.b.net` provides a permissive policy file).

Only cases, in which an Web application combines authentication tracking (or at least state-full session tracking) with an over-allowing "*"-policy, the cross-domain configuration has to be regarded as potentially harmful. In consequence, for the remainder of this paper we exclusively classify policies to be *insecure* if both indicators could be detected by our crawler (see Sec. IV-B4 for details).

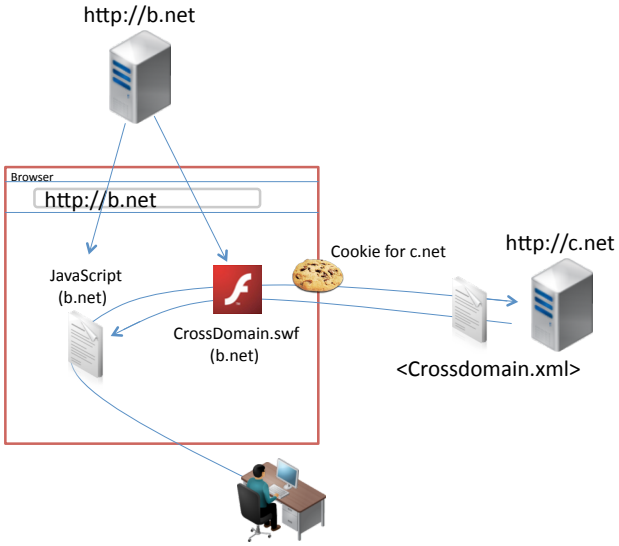

Fig. 1: Attacker Model

D. Transitivity of insecurity

As stated above, the inclusion of a domain `a.net` in the cross-domain policy of the site `b.net` is an expression of `b.net`'s trust in the operators of `a.net` to handle the capability to create authenticated HTTP requests responsible. Hence, in parts the security perimeter of `b.net` is expanded into the realm of `a.net`. This in turn means, that a potential compromise of any site which has been included into `b.net`'s trust domain this way, may also directly affect `b.net`'s security.

The type of enabling vulnerability, that allows the adversary to exploit this circumstance, differs depending on the utilized techniques for granting client-site requests:

1) CORS: As CORS is implemented in the browser's native JavaScript, it is sufficient for the attacker to be able to execute JavaScript under the domain of `a.net`, i.e., a simple XSS flaw is sufficient.

2) Flash and Silverlight: Unlike injected JavaScript, plug-in content, such as Flash or Silverlight objects, retain their origin when being included in foreign web pages. Hence, to abuse a cross-domain trust relationship, the adversary is required to cause `a.net` to serve the attackers Flash/Silverlight object. Hence, in this case an arbitrary file-upload vulnerability has to be present [8].

E. Case study:

To exemplify the malicious capabilities an adversary may gain through a liberal `crossdomain.xml` file, we briefly document security issues that we encountered during our study at a site image sharing site[2].

The affected is a popular image and video-hosting website which offers the possibility for its users to upload and share media files. At the time this paper was written the site was ranked in the top 1000 of the most popular websites in the world according to the Alexa [3] index. After logging in the user is presented with a list of his media files where he can conduct several actions, like editing the images or marking them as private or public. This personal list can be accessed on the site.

The site serves a `crossdomain.xml`file from its Web root. An excerpt of the file can be seen in Listing 4. As it can be seen in the excerpt, the site issues a wildcard policy ("*"), thus, allowing any foreign domain to read data by the use of a cross-domain request. Such cross-domain requests are outfitted by the browser with the website's cookie. Therefore, a flash applet's request appears to the site as if the user itself conducted the request within his authenticated session.

An adversary could, therefore, setup a webpage where he embeds an invisible flash file. If he is able to get a user who is concurrently logged in in the site to visit this website he is able to conduct arbitrary actions in the name of the user. As a prove of concept we implemented a flash applet which reads out the list of images and changes the visibility of all photos which are marked private to public. Besides this obviously very undesired attack a lot of other scenarios are conceivable, like the stealing of personal account data or the upload virus-infected files.

[2]We notified the operators of the site about our findings. The issue has been resolved and the site does not use a wildcard policy anymore.

Listing 4 Excerpt of vulnerable site's crossdomain.xml file

```
<cross-domain-policy>
<allow-access-from domain="*"/>
    [...]
</cross-domain-policy>
```

IV. Assessing the current practice of client-side cross-domain policies

The last public examination of deployed cross domain policies that received considerable attention was done by Jeremiah Grossman in 2008. He examined the policy files of the Alexa Top 500 and Fortune 500 websites [7]. He found that at this point in time 7% of the examined websites had a policy that granted every domain full access via the $*$-wildcard. In this paper, we give a more recent and more complete account on the current situation.

A. Research questions

In this study, we are not only interested in the total number of perceived insecure sites. Furthermore, the goal of the study is to provide a better understanding on the details of currently deployed policies. More precisely, we target to address the following research questions:

(R1) *Penetration*

This study is aimed to provide insight on the current level of usage of client-side cross-domain interaction. As providing a cross-domain policy file is an active act of the serving site, it is safe to state, that each of these files was created to enable at least one specific technical requirement, which is not solvable with the full same-origin restrictions in place. Hence, quantitative data is a valid indicator. In this context, we identified the following points of interest:

- How prevalent is the usage cross-domain policies?
- Which techniques are utilized for this purpose?
- Can a trend towards native JavaScript techniques (i.e., CORS) be observed?
- What kind of sites do issue cross-domain policies?

(R2) *Security*

As already mentioned, the driving force of this survey is the assessment in respect to currently deployed insecure configurations. Hence, we explore:

- How high is the ratio of potentially insecure policies?
- Can we observe a connection between the likelihood of insecure policies and category of the corresponding Web site? I.e, are certain types of sites "more secure" than others in the context of cross-domain policies?
- Is there a correlation between security awareness and non-technical factors such as *popularity*? I.e., does the ratio of insecure configurations raise when the Alex rang declines?

- Which sites are listed in the most policy files and hence attractive targets for attacks that try to leverage the transitivity of insecurity problem (see Sec.III-D)?

B. Methodology

In this section, we briefly document the setup of our experiment. Furthermore, we explore selected aspects of our data collection methodology which we consider worth mentioning.

1) General approach: In order to gain meaningful insights into the usage of cross-domain requests on the web, we decided to crawl as many websites as possible. Therefore, we choose the Alexa top 1.000.000 Websites as it is one of the biggest and most popular rankings. For each website in the ranking we first checked the availability of the website by sending a request to the main page. So if the domain was example.org we first sent an HTTP request to `http://example.org/`. If the website was not available we also called `http://www.example.org`. After having received the response, we stored the returned response headers (containing the CORS and the cookie headers), checked if the main page contained any password fields and downloaded `crossdomain.xml` and `clientaccesspolicy.xml` files. With our crawler infrastructure we crawled the websites in a time frame of about 5 days. As many web server are not using standard HTTP response codes, we also downloaded a lot of invalid files which were mainly error messages served with a 200 response code. Therefore, we conducted some cleansing and preprocessing on the data.

2) Classifying domain values based on topics: Research question (R2) includes the objective to gain insight, whether certain classes of Web sites are more prone to insecure configurations than others.

For this purpose, it is essential to have a notion on the business area of the involved sites (both the provider of the policy file as well as the domains listed). Hence, a method for mapping domain names to their respective Web application category is needed.

Our first stab at this problem was to use the categories which are provided by the Alexa service [3]. Unfortunately, first manual tests revealed that the Alexa categories do not have the level of quality which would be needed for meaningful results. For instance, the provided categories for the video site Youtube.com (at time of writing with an Alexa rank of 3) did not even contain the term "video" at all (see Listing 5). We experienced similar problems with the Google Web directory[3] and the open directory DMOZ[4].

For this reason, we resorted to the wisdom of the crowd approach. Instead of using a directory of Web sites, we utilized the social bookmarking service Delicious.com [19] for information gathering. For each examined domain value, we looked up the top tags which have been assigned to this domain by the Delicious.com users. As social tagging is a decentralized, unorganized process, we had to execute normalization on the data (e.g., identifying terms given in singular and plural form). Furthermore, with sites of big brands, often the actual name of the brand is the most often assigned tag (for youtube.com the second top tag was, for instance, indeed "youtube"). Hence, such tags had to be identified and removed from the index. Furthermore, to end up with a set of categories that has a manageable size, we choose the 100 most frequently used tags from the cleaned set and manually assigned these as sub-categories to 15 top-categories, which were selected according to the top-categories of the other Web directories.

After these cleaning steps, we merely took a site's top tag (i.e., the tag that was used by the largest number of Delicious.com users to characterize the site) to serve as the Website's sub-category, thus, implicitly determining the site's top-category. Applying this for youtube.com, the resulting top tag is "video" which is linked to the *video* sub-category and thus to the *art* top-category.

However, using a social bookmarking service as a data source has one noteworthy disadvantage: Only sites that reach a certain level of popularity are posted to the bookmarking service by enough individual users so that the assigned tags provide reliable information on the nature of the site. For this reason, only a subset of 16.467 of the examined domain values in our test set carry category information. Nonetheless, we believe this number of categorized sites is large enough to gain first insights.

Listing 5 Alexa category results for youtube.com

```
World > Cesky > Kultura > Zabava
World > ... Espana > Guias y directorios
World > ... Suchen > Suchmaschinen > Google
World > ... Moteurs de recherche > Google
Computers > ... > Internet Traffic
World > ... > Wyszukiwarki > Google
World > ... Internet > Ricerca > Motori
```

Listing 6 Delicious.com top tags for youtube.com

```
video (25972), youtube (18248), videos (16876),
entertainment (9200), media (7544), web2.0
(6743), social (4646), fun (4624), music (3378),
community (3141)
```

3) Probing for CORS adoption: As part of research question (R1), we were interested if indicators on technology adoption can be collected. As described in section II-C CORS utilizes HTTP header fields instead of policy files. In order to receive such a CORS header one has to send a header called "Origin:" with the HTTP request. The value of this header has to be the domain from which the cross-domain request originates. If the header is set and the domain is on the whitelist of the receiver, the server adds an additional header named "Access-Control-Allow-Origin:" to the response. The value of the header contains either a wildcard or a list of domains which are allowed to conduct cross-domain requests to the server. If the domain which was sent to the server by the use of the origin header is on that list or a wildcard is received, the request is granted. If this domain is not whitelisted the

[3]Google Web directory: `http://www.google.com/dirhp`
[4]DMOZ: `http:www.dmoz.org`

Flash	Total	Flash only	with SL	with CORS
	82.052	81.439	555	58
Silverlight	**Total**	**SL only**	**with Flash**	**with CORS**
	995	436	555	0
CORS	**Total**	**CORS only**	**with Flash**	**with SL**
	215	153	58	0

TABLE I: Total numbers of collected policy files and CORS headers

server does simply not add the response header and the request will be dropped by the browser.

This fact makes it quite difficult to detect the presence of CORS capabilities, as one must know a whitelisted domain in order to get the correct response containing a response header which indicated CORS readiness. In order to get a realistic view on the dissemination of CORS we therefore utilized two different approaches to receive CORS headers. For one, we used the domain value of the target of the HTTP request as the origin of the request (in lack of a better alternative).

Furthermore, we checked the target domain for existing Flash or Silverlight cross-domain policies. If such policies existed, we compiled the set of whitelisted domains in these policies and used these values within the `origin`-header of the test requests. The underlying reasoning is that, if a server allows cross-domain requests through Silverlight or Flash it is very likely that it also allows requests through CORS to the same domain.

4) Identification of insecure policies: As explained in Sec. III-C, there are legitimate cases for using a general wildcard policy without compromising the issuing site's security. Thus, the mere existence of such a policy is not a sufficient indicator for a present insecurity.

However, as soon as such a policy is combined with authentication tracking, the discussed attacks (see Sec. III-B) gain severity. For this reason, we tested all examined sites for indicators of authentication dialogues. More precisely: We did a shallow crawl of the Web application's UI and attempted to locate either password fields or links to login pages.

Please note: For such authentication forms, the target domain of the form action can differ from the domain value for which the from originally has been served. Hence, whenever we encountered a authentication form that leads to submission of the entered data to a different domain, we tried to obtain the cross-domain policy file for the form's target domain. In such cases, this policy file was the basis for our security assessment.

If for a given site both a wildcard "*" policy and a login dialogue could be found, we labeled the site as *insecure*.

V. Results and analysis

In this section, we present the results of our survey and discuss these results in the context of the identified research questions.

A. (R1) Penetration

In total we probed 1.093.127 individual domains for cross-domain policy files and CORS headers. This number resulted from the first one million domain names included in the Alexa index plus a set of additional domains, which were added to our list because the action-attribute of a login from pointed to these domain (as pointed out above, the security properties depend on the cross-domain configuration of the domain that received the login data, not the one providing the login field).

Out of these domains, a total of 82.052 served valid `crossdomain.xml` files. In addition, we were able to find 995 `clientaccess.xml` files and detect CORS headers in the responses of 215 sites (see Table I for details on combined occurrences). Given the numerical dominance of Flash policies, for now we limit the further discussion to these policies.

While each website serving a valid policy file can be seen as a supplier for cross-domain services, the websites specified within the policies can be seen as consumers of cross-domain services. For the flash policies, we were thus able to identify 82.052 unique suppliers and 67.974 unique consumers of cross-domain services. (Note: In practice the number of consumers could be much higher than the number stated above, as policies only containing a wildcard do not reveal information about the consumers).

In average approximately 8% of all examined sites provide a `crossdomain.xml` policy file. It is noteworthy, that the sites which are within the top 1000 Alexa positions expose a considerable higher probability of providing cross-domain policy files (see Fig. 2). For instance, 70 out of the Alexa top 100 sites provide a policy file and within the Alexa top 1000, 48% of the sites. For the "long tail" of sites, the 8% number is surprisingly stable.

Hence, compared with the results of Grossman's 2008 study, we can observe an increase in the uptake of this technology: Grossman examined the top 500 Alexa and Fortune 500 and observed a ratio of 28% of sites providing policy files.

The majority of analyzed policy files contain references to five or less external domains (see Fig. 3). The largest number

#	Domain	Alexa	#Ref.	Category	Subcategory
1	ning.com	249	1188	Society	Social
2	cooliris.com	2231	1076	Computers	Software
3	mochiads.com	23560	726	Business	Advertisement
4	brightcove.com	5125	718	Arts	Video
5	mochimedia.com	2822	405	Games	Video Games
6	2mdn.net	1892	394	Business	Advertisement
7	facebook.com	2	347	Society	Social
8	amazonaws.com	157	305	Computers	internet
9	weebly.com	462	267	Computers	internet
10	userplane.com	24671	255	Society	Social

TABLE II: Top ten most trusted urls

#	Domain	Entries	Category	Subcategory
1	youronlineagents.com	980	Home	Real Estate
2	grand-casino.com	951	Games	Gambling
3	www.stormpulse.com	848	News	Weather
4	comned.com	443	Computers	Internet
5	orange.co.il	421	Business	Telecommunication
6	www.ivgstores.com	398	Shopping	Furniture
7	www.pointpoker.com	305	Games	Gambling
8	www.mychallenges.net	304	Society	Social
9	timwe.com	294	Business	Advertisement
10	fr.viamichelin.com	283	Reference	Maps & Views

TABLE III: Biggest Policy Files

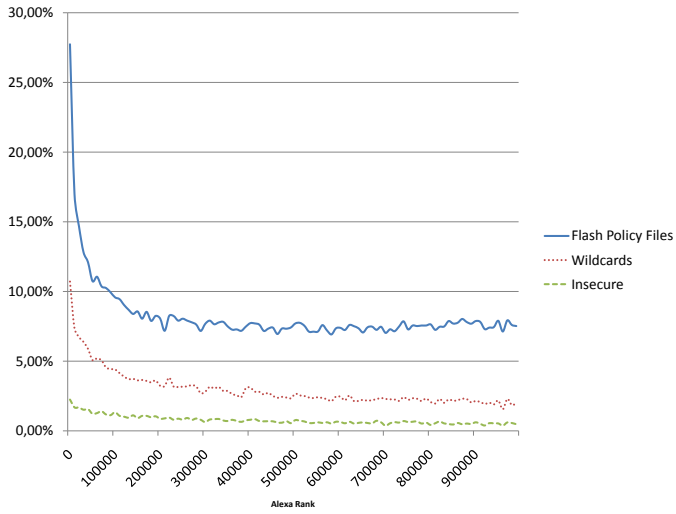

Fig. 2: Collected `crossdomain.xml` files

of entries in a single `crossdomain.xml` file that we could observe was 980 domains (see Tab III).

B. (R2) Security

Out of the 82.052 `crossdomain.xml` files that we collected, 31.011 utilized a general wildcard (a *star policy*), which enables all sites to conduct client-side cross-domain interaction. This results in 37,7% of potentially insecure policies and, thus, a total of 2,8% potentially insecure site in our dataset of 1.093.127 examined domains.

However, as discussed in Sec. III-C, a wildcard policy alone is not a sufficient indicator, that the site is indeed insecure. Hence, we utilized our method of associating evidence for login/session-tracking and wildcard policies (see Sec. III-C) to establish a more conservative number of sites which have to be considered to be insecure.

We identified 15.060 sites in which a wildcard policy is combined with evidence of authentication tracking, resulting in a percentage of 1,3% of sites which are very likely to be susceptible to the attacks, that we outline in Sec. III-B.

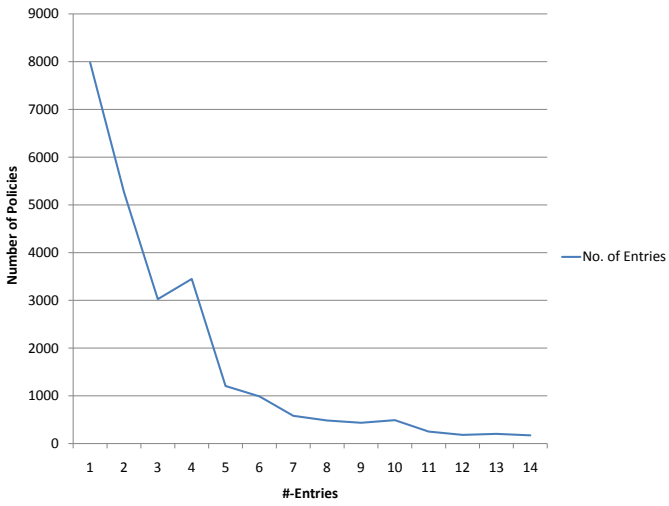

Fig. 3: Hosts Per Policy

Based on the collected data, we could not identify a correlation between Alexa rang and observed percentage of insecure policies. In fact, both the ratio of deployed policies to wildcard policies as well as deployed policies to insecure wildcard policies remains relatively stable over the whole dataset with a slight decrease in the percentage of wildcard policies for lower ranked sites. For illustrations of this circumstances refer to Fig. 2 which visualizes the total numbers, Fig. 4 which shows the percentage of wildcard and insecure policies within the set of all collected policies, and Fig. 5 which plots the ratio of insecure policies to wildcard policies.

To a certain degree, this result surprised us. We expected to observe a "long tail" phenomena, i.e., an increase in insecure policies as the importance (measured via the Alexa rang) of the site decreases.

When it comes to map the sites to their respective categories (see Fig. 6), it can be observed that Web sites of a more causal nature (such as members of the *games*, *adult*, or *sports* category) have the tendency to be more insecure, than their more serious counterparts (e.g., *health* or *business* sites).

Very telling data points are provided by the *education* and *reference* categories. Even though the percentage of wildcard policies compared to the total number of cross-domain policies on the higher end of the spectrum, the ratio of insecure policies is low. Such sites often only provide information without sophisticated interaction capabilities or user management. Thus, they serve as good examples for cases in which wildcard policies are completely legitimate.

Even not having a wildcard policy but allowing specific domains may still be at a risk: as described in Sec. III-D, the inclusion of a domain in the cross-domain policy may increase the overall vulnerability of the trusting domain. An attacker may have a look at the trusted domains in the policy files and check those for vulnerabilities. If we consider it the other way round, an attacker may crawl the web, collect the policy files and then make a list of the most trusted urls as those urls are a target of choices for him (see Tab II). If he manages to exploit a vulnerability in one of those, the number of domains that trust it is high.

VI. Securely using cross-domain policies

The results of our study suggest, that a significant number of Web site operators endanger their users through deploying insecure policies. There a two potential explanations for this: Either the operators are not aware of the security risks that are connected to allowing client-side cross-domain requests. Or, the operator's knowledge on how to securely offer public cross-domain interfaces is limited to a degree, that they resort to deploying general wildcard policies due to a perceived lack of alternatives.

For the former case multiple sources are available, that discuss the security issues (see Sec. III). However, the latter case is much more sparsely documented. Especially, the scenario in which a Web site administrator wants to open a selected subset of his application's interface to cross-domain requests

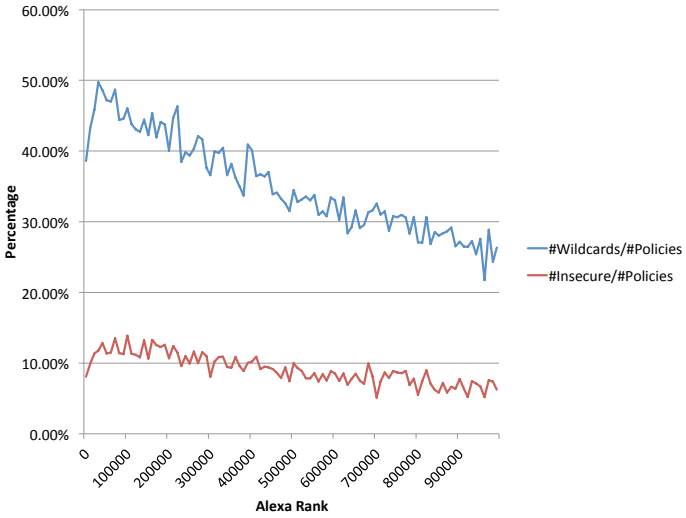

Fig. 4: Percentage of wildcard and insecure policies within the set of collected policies

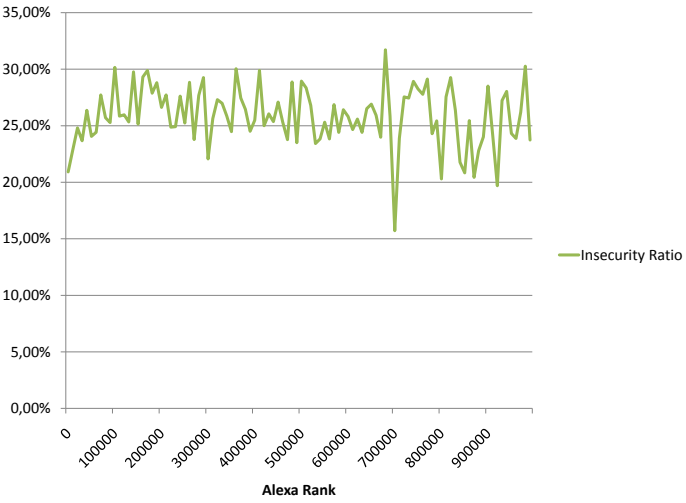

Fig. 5: Ratio insecure policies in the set of all wildcard policies

from arbitrary sources while denying them for the rest of the application is not well covered.

In this section, we give recommendation for such situations, through discussing how the public content can be separated from the restricted interfaces with properly configured policy files. There are two approaches to achieve this goal:

1) Usage of different subdomains: Each type of content can be stored an a specific subdomain like `api.a.net` for the public one and `secret.a.net` for the restricted one. In this case, each subdomain will have its own Flash/Silverlight policy files. The policy file of `api.a.net` could then contain a wildcard as it is only used for public content whereas the policy file of `secret.a.net` only contains domains that are trusted, for example partners or other sites under the control of `a.net` owner.

2) Usage of different subfolders: If creating a new subdomain is not possible, the content can be separated using subfolders, like `a.net/api` and `a.net/restricted`. As described in Sec. II-C1, it is possible to define Flash meta-policies that apply to specific folder. For this, the master policy has to allow it by using the site-control `permitted-cross-domain-policies` element. Given such a master policy, further policy files can be served on subpaths of the application, which govern the cross-domain access for this path and all further subpaths.

Please note: When setting `permitted-cross-domain-policies` to `all`, any file can be used as a policy file. This can be dangerous when not all the files on the server are under the control of the site's owner. If a user can upload files on the server, an attacker could upload its own Flash policy file [5]. It is therefore recommended to set this attribute to `by-content-type`. Only policy files that are served with Content-Type: `text/x-cross-domain-policy` are allowed.

For Silverlight, this is not a problem as there is only one policy file with multiple policies that apply for different resources.

VII. Related work

In concurrent and independent work, Kontaxis et al. have published a study on Flash policy files closely related to our efforts [10]. The authors examine the cross-domain policy files of the top 100K site listed in the Alexa index. The authors observe that the frequency of insecure policies increases with decreasing popularity. This observation differs from our results, as we did not observe such a "long tail" phenomena. The reason for this conflicting outcomes lies in two facts: For one, their definition of an insecure policy is broader than ours, as they classify all wildcard policies to be insecure and all partial wildcards to be "weak" policies. Furthermore, their data basis is significantly smaller than ours (100K vs 1.1 million examined sites). As it can be seen in Figure 4, their observed increase in insecurity only applies to the first 10K of examined sites and after the first 100K sites the percentage of wildcard policies is actually decreasing while the ratio of insecure polices (in our definition) is relatively stable over the full data set (see Fig 5).

In addition, the recent history has shown that the introduction of capabilities for client-side, cross-domain HTTP requests is prone to abuse. In the remainder of this section we list documented cases that relate to the issues discussed in this paper. The potential security issues with Flash's cross-domain capabilities have received attention from the applied security community [5, 16] early on. Public documentation of real issues were given by, e.g., Rios [13] who compromised Google's mail service GMail by attaching a forged `crossdomain.xml` to an email, and by Grossman [8] who accessed private information on `youtube.com` by uploading a swf-file to a host which was whitelisted in YouTube's policy. In 2010, the tool MalaRIA (short for 'Malicious Rich Internet Application') [12] was released. The tool provides a graphical user interface to interactively conduct session hijacking attacks, as outlined in Section III.

VIII. Conclusion

In this paper, we presented an empirical study on currently deployed cross-domain policy files to assess the level of adop-

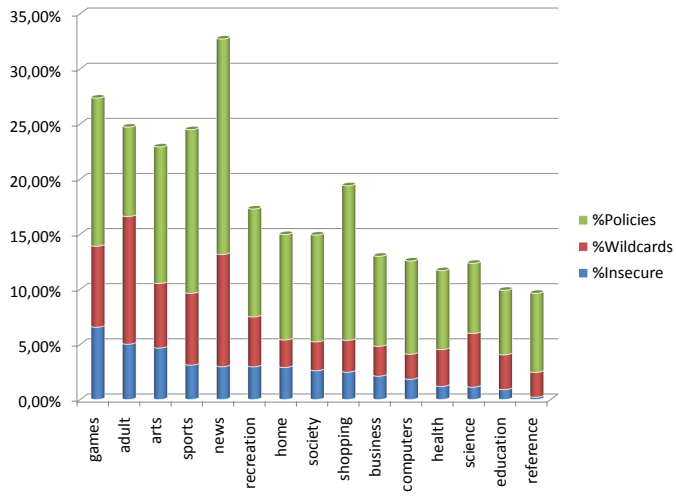

Fig. 6: Mapping policy files to the top-categories

tion of this Web application technique and the corresponding ratio of potentially insecure configurations. For this purpose, we collected the policy files of the top 1.000.000 domains listed in the Alexa index, along with a set of policies of domains which are directly connected to these site through login processes.

We found a total of 82.052 valid `crossdomain.xml` policies. Out of these policies, 31.011 implement a general wildcard policy, hence, allowing all external sites to generate client-side, cross-domain requests to the site. Furthermore, through associating indicators for authentication tracking with wildcard policies, we deducted that for 15.060 of these sites the likelihood of resulting vulnerabilities is very high.

References

[1] Adobe Coperation. Adobe flash. [online] http://www.adobe.com/products/flash/flashpro/.

[2] Wade Alcorn et al. Browser Exploitation Framework (BeEF. [software], http://code.google.com/p/beef/, accessed in January 2011, 2011.

[3] Alexa Internet, Inc. Alexa Top 500 Global Sites. Website, http://www.alexa.com/topsites, accessed in March 2010.

[4] Jesse Burns. Cross Site Request Forgery - An introduction to a common web application weakness. Whitepaper, https://www.isecpartners.com/documents/XSRF_Paper.pdf, 2005.

[5] Stefan Esser. Poking new holes with Flash Crossdomain Policy Files. [online], http://www.hardened-php.net/library/poking_new_holes_with_flash_crossdomain_policy_files.html, Accessed in January 2011, October 2006.

[6] Google inc. Google Gadgets API: Working with Remote Content. [online], http://code.google.com/apis/gadgets/docs/remote-content.html, accessed in January 2011.

[7] Jeremiah Grossman. Crossdomain.xml Invites Cross-site Mayhem. [online], http://jeremiahgrossman.blogspot.com/2008/05/crossdomainxml-invites-cross-site.html, Accessed in January 2011, May 2008.

[8] Jeremiah Grossman. I used to know what you watched, on YouTube. [online], http://jeremiahgrossman.blogspot.com/2008/09/i-used-to-know-what-you-watched-on.html, Accessed in January 2011, September 2008.

[9] Samy Kamkar. Technical explanation of the MySpace worm. [online], http://namb.la/popular/tech.html, accessed in January 2011, October 2005.

[10] Georgios Kontaxis, Demetris Antoniades, Iasonas Polakis, and Evangelos P. Markatos. An Empirical Study on the Security of Cross-Domain Policies in Rich Internet Applications. In *Proceedings of the 4th European Workshop on System Security (EuroSec 2011)*, 2011.

[11] Benjamin Livshits and Weidong Cui. Spectator: Detection and Containment of JavaScript Worms. In *Usenix Annual Technical Conference*, June 2008.

[12] Erlend Oftedal. Malicious rich internet application (malaria). [software], http://erlend.oftedal.no/blog/?blogid=107, accessed in January 2011, April 2010.

[13] Billy Rios. Cross Domain Hole Caused By Google Docs. [online], http://xs-sniper.com/blog/Google-Docs-Cross-Domain-Hole/, Accessed in January 2011, 2007.

[14] Jesse Ruderman. The Same Origin Policy. [online], http://www.mozilla.org/projects/security/components/same-origin.html (01/10/06), August 2001.

[15] Chriss Shiflett. Cross-Domain Ajax Insecurity. [online], http://shiflett.org/blog/2006/aug/cross-domain-ajax-insecurity, Accessed in January 2011, August 2006.

[16] Chriss Shiflett. The Dangers of Cross-Domain Ajax with Flash. [online], http://shiflett.org/blog/2006/sep/the-dangers-of-cross-domain-ajax-with-flash, Accessed in January 2011, September 2006.

[17] Anne van Kesteren. The XMLHttpRequest Object. W3C Working Draft, http://www.w3.org/TR/XMLHttpRequest, April 2008.

[18] Anne van Kesteren (Editor). Cross-Origin Resource Sharing. W3C Working Draft, Version WD-cors-20100727, http://www.w3.org/TR/cors/, July 2010.

[19] Yahoo Inc. Delicious.com - Social Bookmarking. [Web application], http://delicious.com/, accessed in January 2011.

[20] Michal Zalewski. Browser Security Handbook. Whitepaper, Google Inc., http://code.google.com/p/browsersec/wiki/Main, accessed in January 2011, 2008.