Computer Communications 34 (2011) 577–590

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

An investigation of hotlinking and its countermeasures

Zi Chu *, Haining Wang

Department of Computer Science, The College of William and Mary, Williamsburg, VA 23187, USA

ARTICLE INFO

Article history:
Available online 25 May 2010

Keywords:
Hotlinking
Resource hosting
Web measurement
Web security

ABSTRACT

Hotlinking is a web behavior that links web resources on a hosting site into a webpage belonging to another site. However, unauthorized hotlinking is unethical, because it not only violates the interests of hosting sites by consuming bandwidth and detracting site visiting traffic but also violates the copyrights of protected materials. To fully understand the nature of hotlinking, we conduct a large-scale measurement study and observe that hotlinking widely exists over the Internet and is severe in certain categories of websites. Moreover, we perform a detailed postmortem analysis on a real hotlink–victim site. After analyzing a group of commonly used hotlinking attacks and the weakness of current defense methods, we present an anti-hotlinking framework for protecting materials on hosting servers based on existing network security techniques. The framework can be easily deployed at the server-side with moderate modifications, and is highly customizable with different granularities of protection. We implement a prototype of the framework and evaluate its effectiveness against hotlinking attacks.

1. Introduction

With the rapid advance of web technologies, websites not only display text-based information, but also host various types of materials including images, media clips, software installation files, and so on. Those hosted materials of great interest and high value help websites attract users and increase site traffic. To date, more and more websites link web elements provided by third parties. Hotlinking, as a web phenomenon, can be defined as including a linked object (often in the form of an image or document) from one site that actually hosts it into a webpage belonging to another site. A hotlinking site has no need to host linked objects itself. In the context of social engineering, hotlinking is a double edged sword. The ethical boundary between benign and malicious hotlinking is whether the linking behavior is authorized by the hosting site or not.

Authorized hotlinking is beneficial to site interaction. For example, a site may include some ad images provided by an advertisement syndicator to make advertising revenue. It does not need to host any ad images by itself, but link them from the syndicator's server.[1] In this way, the syndicator can dynamically change ad contents and obtain the first-hand trace of ad display. Since the above interaction is approved by both parties, it is a benign hotlinking behavior.

By contrast, unauthorized hotlinking is often harmful to hosting sites. There are some common reasons for unauthorized hotlinking listed as follows. First, some web developers are unprofessional. Their laziness makes them to directly link web objects hosted somewhere else. Second, the hotlinking site may not have enough online storage space to host all the materials it wants to display or accommodate bandwidth demands by frequent visits. Third, the hotlinking site attempts to display some "grey materials" such as pirated media. Hotlinking instead of hosting such materials may dodge legal prosecution.

Considering the simplified website operation model from the economical perspective, the site hosting cost is attributed to paying the hosting service provider for storage space and bandwidth quota, while the gain includes website brand effect and online advertisement revenue. As far as the gain outweighs the cost, this economically sustainable model drives the prosperity of the Internet community. However, the rampant unauthorized hotlinking has recently disturbed the harmonious development of websites, because this unethical behavior violates the interests of hosting sites in terms of the following aspects.

- **Bandwidth theft.** Most websites hosted on third-party hosting servers have to pay for a limited amount of traffic delivery. If the bandwidth consumption exceeds the prepaid quota, the website may be charged more or, in the worst-case scenario, shut down temporarily. It is evident that stealing bandwidth increases the site hosting cost. From this perspective, unauthorized hotlinking is also known as leeching or bandwidth theft [1].
- **Visitor traffic loss.** Many websites hosting free materials rely on online advertising revenue. They display ads assigned by syndicators (like Google AdSense [2] and Yahoo! Advertising [3]) on webpages, and are paid for ad impressions or clicks. If the hosted materials are hotlinked, visitors are directly brought

* Corresponding author. Tel.: +1 917 698 5015.
E-mail addresses: zichu@cs.wm.edu (Z. Chu), hnw@cs.wm.edu (H. Wang).

[1] Usually the webpage contains a JavaScript snippet provided by the syndicator; during the execution, JavaScript embeds ad links from the syndicator.

0140-3664/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2010.05.007

to hotlinking pages. Since legal hosting pages are bypassed, no advertising revenue is generated. Visitor traffic loss could also reduce the site brand effect. Users view or download popular hotlinked materials through hotlinking sites without knowing the existence of the hosting site.

- **Copyright infringement.** Many web objects (such as photographs and software programs) are copyrighted or licensed. Owners have exclusive rights to display their works publicly Strickland [4]. Hotlinking, even in the form of inline linking or framing, such objects in the commercial environment may infringe copyright, and is not justified by fair use [5].

The paper focuses on the *malicious* (i.e., *unauthorized*) hotlinking. In the context of a hotlinking attack, we call the site that actually hosts the object *the hosting site* or *victim site*, and the site that hotlinks the object *the hotlinking site* throughout the paper. To fully understand the nature of hotlinking, we conduct a large-scale measurement study over the Internet, targeting two popular types of web materials, images and software packages. In the image-centric measurement, we choose 1453 popular websites and check for images hotlinked in their homepages. We find out that about 75.0% of sites hotlink images. To decide whether such hotlinking behaviors are authorized or not, we further analyze the nature of images and their hosting sites. A great amount of images are hosted on special-purposed sites (such as ad syndicators and traffic monitoring sites) and thus should be considered as hotlinked with authorization. However, we also observe that unauthorized hotlinking widely exists in certain categories of websites like blogging. In the software-centric measurements, we select 100 popular software packages as the targets. We search the Internet for websites linking those software binaries, and find out that most hosting sites are hotlinking victims. Since the download link appears in the hotlinking page, the visitor can directly download the software package without visiting the hosting site.

It is a cat-and-mouse game between hotlinking attack and anti-hotlinking defense. The paper analyzes a series of commonly used hotlinking attacks and corresponding defense methods. Integrating with the existing anti-hotlinking techniques, we present an anti-hotlinking framework for protecting hosting sites. We also implement a prototype of the proposed framework to demonstrate its feasibility, and evaluate its effectiveness against hotlinking attacks in terms of security and usability. The framework can be easily deployed at the server-side with moderate modifications. Moreover, it is highly customizable with different granularities of protection, with which a webmaster can define the different requirements that visitors must complete to qualify for downloading resources.

The remainder of the paper is organized as follows. We present the threat model along with some common hotlinking attacks in Section 2. We describe the large-scale measurement-based study of hotlinking over the Internet in Section 3. We detail our anti-hotlinking framework design in Section 4. The framework implementation and evaluation are given in Sections 5 and 6, respectively. Section 7 surveys some related work. The paper concludes in Section 8.

2. Problem statement

Hotlinking generally involves two parties: hotlinking sites and victim sites (namely original sites or hosting sites). The HTML snippet shown in Fig. 1 gives an example of hotlinking with four types of regular web objects. Suppose the example page URL of the hotlinking site is www.H.com/h.htm. All of the linked objects are hosted by the victim site, V.com. Linking another webpage (shown as L1) is not considered as the hotlinking behavior since visitors will be directly brought to that page. L2–L4 show how to hotlink an image, flash clip and file, respectively.

```
L1:   <a href="www.V.com/b.htm">Go to Page b</a>
L2:   <img src="www.V.com/images/c.jpg"/>
L3:   <object>
      <param name="movie" value="V.com/media/d.swf">
      <embed src="V.com/media/d.swf"
      type="application/x-shockwave-flash"></embed>
      </object>
L4:   <a href="www.V.com/files/e.pdf"/>Download PDF</a>
```

Fig. 1. Basic forms of linking web objects.

```
$server = 'www.H.com';
$host = 'www.V.com';
$target = '/files/install.rpm';
$referer = 'www.V.com/'; // Fake $referer insert here
$fp = fsockopen($server, $port, $errno, $errstr, 30);
fwrite($fp, $out);
```

Fig. 2. Fabricating HTTP_REFERER via PHP.

We assume that the hotlinker owns an independent domain name (such as H.com), and fully operates a web server that hosts his site. We further assume that the hotlinking attack is constrained by the same-origin policy enforced by browsers, since the same-origin policy has been widely-used in modern browsers. The policy isolates web contents from different schemes, hosts or ports Jackson et al. [6]. For example, the scripts from http://H.com cannot access or modify the content of http://V.com on the same page.

The objectives of a hotlinker include: (1) hotlinking web objects hosted on other servers into webpages of his own, and (2) ensuring hotlinked objects to be normally displayed or accessed on the client-side. The first goal is easy to achieve by using similar codes shown in Fig. 1. The second goal is more challenging to reach, since the hotlinker has to detect and bypass the anti-hotlinking mechanisms enforced by hosting sites. In the following, we list some common hotlinking techniques, and analyze how they evade the corresponding defense countermeasures.

2.1. Exsiting hotlinking techniques

Directly hotlinking web objects into hotlinkers' webpages (known as *Direct Hotlinking*) is the basic form of hotlinking attacks. It can simply use HTML codes displayed in Fig. 1. As a counterattack against *Direct Hotlinking*, many hosting sites currently use a straightforward Referer-based technique. It judges the Referer field in the incoming HTTP request header. If the Referer does not belong to its own domain, the hosting site determines that the requested material is linked from another domain (namely hotlinked) and thus refuses to respond. According to the HTTP protocol specification [7], the Referer field allows the client to identify the URI of the resource from which the Request-URI was obtained. Therefore, checking the Referer field is an effective way to prevent hotlinking as long as the integrity of Referer can be guaranteed.

Hotlinking sites can fabricate the Referer field to bypass the widely used Referer-based defense. Fig. 2 shows an example of using PHP code to fake HTTP_REFERER for outgoing HTTP request headers[2]. An arbitrary fake Referer could be injected at the statement $Referer='www.Any.com/'. When a user clicks on the hotlinked download link, the browser generates an HTTP request for that file (www.V.com/files/install.rpm), and HTTP_REFERER is changed into 'V.com/', instead of the real one (H.com/h.htm). Because the victim

[2] Tampering with HTTP request header can also be achieved using other scripts or tools.

server always allows webpages in its own domain to link hosted materials, this type of attacks evades the Referer-based defense. Chen and Henry [40] and Chou et al. [41] confirm in their research of web-based identification theft again that, the Referer field is easy to fabricate.

Many websites have realized the vulnerability of HTTP Referer, and have improved anti-hotlinking methods with the help of cookie and session mechanisms. Cookie is a piece of data that a server stores on the browser to maintain the HTTP communication state between the client and server-sides, since the HTTP protocol itself is stateless [7]. If the server receives the cookie issued by itself, it can tell that the user has already visited the site before and thus grants the file download.

Different from cookie, the session mechanism stores information on the server-side except a session ID on the client-side [8]. To implement the session tracking, there are two methods to store session id on the browser: (1) using cookie to record session ID, or (2) URL Rewriting. Since the former method works similarly with the cookie mechanism, we use the latter one as an example to describe how the session mechanism work. When the browser visits www.V.com/v.htm, the server creates a session with a unique ID. Suppose that the server implements URL Rewriting via PHP [9]. The session ID will be automatically appended to the links towards the same domain as a URL parameter (such as www.V.com/v.htm?PHPSESSID=5k32d0). When the server receives the request, it authenticates the session ID and returns the file if the id is valid.

However, the defense framework purely based on cookie or session mechanism can still be evaded by more sophisticated hotlinking attacks, which are described as follows. In the hotlinking attack against cookie protection as shown in Fig. 3, the hotlinker can include a hidden iFrame on the hotlinking page to bypass the cookie check. When the browser parses the HTML document of h.htm, the iFrame will load the page v.htm from V.com. V.com then stores a cookie in the browser. When a user clicks the download link, an HTTP request (GET /files/install.rpm HTTP/1.1) is sent to V.com, and the cookie issued by V.com is also included in the request header. After V.com receives its cookie, it returns an HTTP response containing the file install.rpm to the browser. Note that the file should be considered as *hotlinked* in h.htm because the legal page v.htm is hidden in the iFrame and never displayed to the user. Park and Sandhu [42] and Li et al. [44] furthermore discuss securing and extending cookie with security properties.

Simple HTML tricks can also be used to bypass the session-based defense (namely Hotlinking against Session Protection). Fig. 4 shows such an instance. We assume that a session ID is passed to the browser via URL Rewriting. After the browser parses the HTML document of the page h.htm, it sends an HTTP request (GET www.V.com/v.htm HTTP/1.1) attempting to obtain the fictitious image via the URL specified by the href attribute of the <img> tag. When V.com establishes the session with the browser and assigns a session ID, the malicious JavaScript code, append_sid(), can be triggered to retrieve the session ID and then append it to the href attribute of the file download link with the JavaScript functions getAttribute() and setAttribute(). Thus, the request URL for the file will be artificially rewritten with the legal session ID. Once V.com receives the HTTP request and validates the session ID, it grants the file request and the anti-hotlinking defense becomes void. The legal page v.htm is not displayed because it is included in an <img> tag, and of course fails to render. The naughty script exists in the hotlinking page, and it is beyond the ability of script purifying techniques deployed by benign servers (Ter Louw and Venkatakrishnan [10]). Noiumkar and Chomsiri [43] listed some more complicated session hijacking cases as the authors evaluated the security level of some popular free web-mail.

```
<iframe src="V.com/v.htm" style="display:none">
<a href="V.com/files/install.rpm">
  Download RMP Package</a>
```

Fig. 3. HTML snippet exploiting cookie vulnerability.

```
<script>function append_sid(){
   retrieves SID;
   var fileLinkObj =
   document.getElementById("fileLink");
   var newURL = fileLinkObj.getAttribute("href")
      + "?&PHPSESSID=SID";
   fileLinkObj.setAttribute("href", newURL);
}</script>
<img src="www.V.com/v.htm" style="display:none">
<a id="fileLink" href="www.V.com/files/install.rpm"
   onclick="append_sid();">Download RMP Package</a>
```

Fig. 4. HTML snippet exploiting session vulnerability.

2.2. Defense against hotlinking

There are two places to enforce anti-hotlinking, the client-side and server-side. Since end users do not have direct motivation to defend against hotlinking, the client-side is not the ideal place for anti-hotlinking. First, hotlinking is transparent to end users. Very few users would even notice or care whether embedded web materials are from other domains or not. Second, anti-hotlinking may deteriorate user-perceived performance and even interfere with users' browsing activities. For example, some defense methods simply refuse HTTP requests for hotlinked objects, and thus involved web contents fail to load on the browser. By contrast, unauthorized hotlinking steals server resources and costs revenue loss to victim sites. Thus, we believe that the server-side is the appropriate place to deploy the anti-hotlinking framework, and it is the responsibility of hosting sites to protect their materials from being hotlinked. Our proposed anti-hotlinking framework adopts this design principle (see more details in Section 4).

3. Hotlinking measurement

In this section we first conduct a large-scale measurement study to understand the nature of hotlinking in a quantitative way. Among the regular types of hotlinked web objects, we choose image and software installation packages (a typical representative of large-sized files) as the measurement targets. Then we observe a hotlinking attack on a web server and perform a detailed analysis based on the logs of the hotlinking victim. Our analysis is focused on the negative effect on the hotlinking victim in terms of computation and communication costs.

3.1. Measurement of hotlinked images

With the development of web 2.0, web sites interact with each other much closer than ever before. Webpages of a site may link a variety of materials hosted on third-party sites, such as scripts and ad contents. Among them, images are the most hotlinked. In this part of image-centric measurements, we choose some representative sites, and record images hotlinked in site homepages.

Hotlinking is a simple web technique. Whether it is ethical or not to hotlink depends on how it is used. Hotlinking an image with the authorization from its hosting site is benign and acceptable. For example, the site homepage links an image hosted by a traffic monitor site. Each request for the homepage triggers a request for the hotlinked image. In this way, the monitor site traces visitor traffic for the client site. On the other hand, intentional hotlinking without

Table 1
Category breakdown by top-level domain.

Category	com	net	org	gov	edu	cc	Other	Total
Arts	85	3	4	0	0	7	1	100
Business	90	2	0	1	0	7	0	100
Computers	91	1	6	0	0	2	0	100
Games	96	2	2	0	0	0	0	100
Health	61	2	15	14	2	5	1	100
Home	89	1	3	5	0	2	0	100
News	85	0	2	0	0	13	0	100
Recreation	88	2	3	3	0	3	1	100
Reference	37	0	10	6	38	8	1	100
Regional	64	0	2	3	0	31	0	100
Science	49	1	15	19	6	10	0	100
Shopping	95	0	1	0	0	4	0	100
Society	68	1	11	10	1	7	2	100
Sports	86	0	2	0	0	12	0	100
World	38	4	4	0	0	53	1	100
Blogging	319	17	11	9	1	47	5	400
Total	1441	36	91	61	48	211	12	1900
Unique	1115 (76.7%)	34 (2.3%)	72 (5.0%)	28 (1.9%)	43 (3.0%)	151 (10.4%)	10 (0.7%)	1453 (100%)

authorization is unethical. For instance, a blog hotlinks a copyrighted photograph from the official site of a celebrity to attract click traffic. In this case, displaying the image is against the owner's will, and incurs additional transfer traffic for the hosting site.

We cannot determine hotlinking is authorized or not merely based on the URL of the hotlinked image. We analyze the characteristics of images and their hosting sites to classify authorized and unauthorized hotlinking with the help of a set of pre-defined rules. We focus more on unauthorized hotlinking.

3.1.1. Chosen websites

The target websites for measurement are chosen as follows. We select 15 categories listed by Alexa [11], and take the top 100 sites from each category. These categories are mainly divided based on site content. Alexa is a well-known web archiving site that consistently monitors web traffic and site popularity ranking. We also add a 16th category — blogging. From ranking lists published by [12,13], we include the top 400 blog sites, as many of such sites greatly hotlink images because of their shortages in hosting storage space and bandwidth quota.

Table 1 lists the breakdown summary of all the 16 categories by DNS Top-Level Domain (TLD) for comparison. Note that some sites are listed in multiple categories. For example, www.google.com appears in both the Computers and World categories. Besides, some sites have multiple domain (or sub-domain) names listed (i.e., news.bbc.co.uk and bbc.co.uk). We run the analysis toolkit to eliminate the duplicates to have the unique site numbers listed at the bottom of Table 1. We can see that 1115 (77%) unique sites belong to the .com TLD, which is the dominant domain in our chosen sites, followed by the country code (denoted as cc) TLD that contributes 151 (10%) unique sites. The rest of TLDs takes up 13% of the chosen sites.

3.1.2. Data collection

We develop a Firefox extension along with some script commands to automatically visit the homepages of target sites one by one. Displaying a webpage on a browser generally involves the following two steps: (1) after the browser loads the page, it parses the HTML document; and then (2) the browser performs a series of actions to display the page on a best-effort basis, including executing dynamic codes (like JavaScript), downloading embedded images, and so on. After the browser loads and parses the page, our Firefox extension logs all the outgoing requests for web objects (like images, JavaScript snippets, .css files, etc.) made by the browser in real-time. We believe that this dynamic logging mechanism is more accurate than the traditional static content check over web objects pre-included in the HTML document, since many contents are dynamically generated such as ads inserted by third-party JavaScript. The extension stays at each site homepage for 45 s before switching to next site.[3] After collecting all the homepage logs, we run a toolkit mainly written in Java to retrieve image object information from each log based on extension type and to decide whether an image object is hosted locally or hotlinked from other sites based on URI.

3.1.3. Data analysis

We first describe the analysis and processing of image measurement results, which are summarized in Table 2. For the homepage of each website, it may include two types of images: those hosted by the site itself and those hotlinked from other sites. We can distinguish these two types of images by comparing URIs of the site and images. If the homepage of a site links any images hosted by other sites, the site is categorized as *Site with Hotlinking Behavior* (Column 2 in Table 2). There is nothing wrong with using hotlinking. However, the key issue here is to differentiate between authorized and unauthorized hotlinking. Our large-scale measurement involves 1453 sites, and the majority of these sites link images from many other sites. It is very time-consuming to manually check whether a site obtains the authorization from another to hotlink an image. Based on our observations, we apply the following three rules for processing the measurement results.

- Rule 1. If the page of site A contains any iFrames or scripts from site B, and B includes images in iFrames or dynamically inserts via scripts, it is considered that A is authorized to hotlink those images from B. This fact implies the cooperation relationship between the two sites. In this case, B plays the role of content (image) provider. Take online ad assignment as an example. A includes an iFrame from syndicator B, and B puts links of ad images in the iFrame, and dynamically changes them to update ad content. This rule also applies to many other web applications, such as site traffic monitoring, and webpage tagging.
- Rule 2. A white list is created to cover the popular sites in the categories which generally authorize hotlinking from themselves. The representative categories include advertising syndi-

[3] We measure the load time of 100 sample site homepages. Most of them do not exceed 20 s. Thus, we set a safe threshold value here, 45 s, which is long enough for fully displaying a page.

Table 2
Image hotlinking distribution per site home.

Category	Sites w/HL behavior	Sites w/unauthorized HL	Unauth-hotlinked image ave.	Sites w/authorized HL	Auth-hotlinked image ave.
Arts	70	19 (27.1%)	2.79	69 (98.6%)	10.55
Business	36	5 (13.9%)	1.60	34 (94.4%)	14.41
Computers	40	7 (17.5%)	2.29	39 (97.5%)	16.15
Games	57	11 (19.3%)	3.00	56 (98.2%)	11.45
Health	50	6 (12.0%)	2.00	49 (98.0%)	10.47
Home	64	18 (28.1%)	1.78	61 (95.3%)	11.10
News	78	16 (20.5%)	1.88	78 (100%)	13.22
Recreation	39	8 (20.5%)	3.38	39 (100%)	9.54
Reference	35	6 (17.1%)	3.17	34 (97.1%)	11.74
Regional	26	8 (30.8%)	1.50	25 (96.2%)	19.32
Science	29	13 (44.8%)	2.15	26 (89.7%)	7.19
Shopping	56	23 (41.1%)	3.22	52 (92.9%)	11.96
Society	35	13 (37.1%)	3.46	28 (80.0%)	13.82
Sports	41	18 (43.9%)	3.72	36 (87.8%)	8.31
World	39	12 (30.8%)	2.50	31 (79.5%)	10.48
Blogging	395	273 (69.1%)	9.00	360 (91.1%)	32.35
Total w/o blogging	695	183 (26.3%)	2.55 (avg)	657 (94.5%)	11.98 (avg)
Total	1090	456 (41.8%)	2.96 (avg)	1017 (93.3%)	13.25 (avg)

cators (such as Google Syndication [14], 2mdn [15] and DoubleClick [16]), web performance accelerators (such as Akamai and Speedera [17][4]), and image hosting/cloud service (such as Flickr [18] and CacheFly [19][5]). Moreover, the white list includes a few partnerships between specific sites, such as yahoo.com and yimg.com, where the latter hosts images for the former. This partnership suggests authorized hotlinking.

- Rule 3. Webpages often include a large amount of very small images mostly in the format of GIF and PNG. These small images are mainly used as tags of social networks (such as Facebook, Dig and Twitter), toolbar logos, face expression symbols, and so on. Generally hosting sites of such small images authorize and solicit hotlinking to spread their brand names. In our measurement, we set the threshold as 10 KB.[6] Any images whose sizes are less than the threshold are classified as authorized hotlinking.

A program equipped with the above rules is used to automatically distinguish unauthorized hotlinking behaviors from authorized ones. If an image follows any of the rules, it is labeled as an *authorized hotlinked image*. Otherwise, it is labeled as an *unauthorized hotlinked image*. The automatic determination saves us from burdensome manual check over hotlinked images. However, the side-effect is the possible false negatives (unauthorized hotlinking misjudged as authorized hotlinking) and false positives (authorized hotlinking misjudged as unauthorized hotlinking). We randomly select 20% of the sites with hotlinking behavior to perform the manual check. For each hotlinked image, we examine the characteristics of hotlinking and victim sites, their relationship, and the content and size of the image. Then we determine whether hotlinking the image is authorized or not, and compare with the decision made by the program. The false negative ratio is 0.1%. Very few images should have been judged as unauthorized hotlinking based on their contents. However, due to their sizes smaller than the threshold value in Rule 3 (namely 10 KB), they are misjudged as authorized hotlinking by the program. The false positive ratio is 3.7%, which is caused by the following two reasons.

[4] These sites let normal sites to hotlink special-purposed images, and trace image requests to gather statistical information about normal sites.

[5] These sites provide online storage services most of which are free, and their policies explicitly allow hotlinking.

[6] According to the measurement results in our crawl experiment, the majority of this type of images are smaller than 10 KB.

- The incompleteness of the white list in Rule 2. For example, it does not contain a site (imagevenue.com) providing free image hosting. Hotlinking images from this site should be considered as authorized. Besides, the list misses specific relationships between some sites. For example, our manual check discovers the redirection from bdd.com to randomhouse.com, which suggests the site partnership. Thus hotlinking is authorized between the two sites.
- The setting of image size threshold in Rule 3. Our manual check observes that some images whose sizes are greater than the 10 KB threshold. However, their contents of ads strongly suggest the authorized hotlinking.

Our manual check confirms the accuracy of the automatic classification of authorized and unauthorized hotlinking. Now we analyze the statistical data listed in Table 2. A site containing such images is categorized as a *site with authorized hotlinking behavior* (Column 5 in Table 2). We divide the sum of authorized hotlinked images by the number of sites with authorized hotlinking behavior to obtain the average of authorized hotlinked images per site (Column 6 in Table 2). Similarly, we count the *sites with unauthorized hotlinking behavior* (Column 3 in Table 2) and compute the average of unauthorized hotlinked images per site (Column 4 in Table 2), respectively. If a site hotlinks both types of images, it is counted in both Column 3 and Column 5. In the first 15 categories without blogging, 46.3% of the sites (namely, 695 out of 1500) hotlink images. In other words, more than half of the sites (53.7%) host all the images by themselves and do not hotlink. Among the hotlinking sites, most sites (94.5%) manifest authorized hotlinking behaviors, and the average of authorized hotlinked images per site is 11.98. Only a minority of sites (26.3%) have unauthorized hotlinking behaviors, and the average of unauthorized hotlinked images per site is 2.55. Such measurement results are expected. Those top sites from the 15 categories are mostly owned by large organizations. They have adequate online storage space and traffic quota, and can host any images by themselves. They do not have strong motivations to "steal" images from others. They perform hotlinking mainly for site interaction. We manually check the nature of authorized hotlinked images, and find out that most of them are for online advertising, webpage tagging, and site partnership displaying.

However, the blogging category is an exception. Most blogging sites (395 out of 400) hotlink images. Among them, 273 blogging sites have unauthorized hotlinking behaviors. The unauthorized

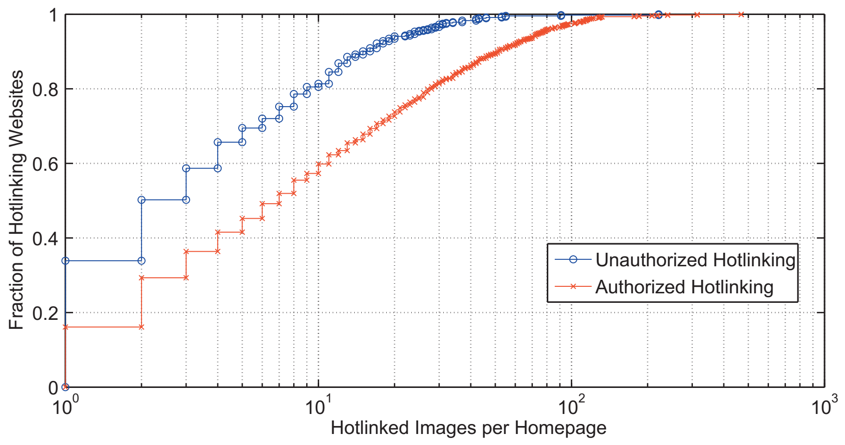


Fig. 5. CDF of the number of hotlinked images per homepage.

ratio of 69.1% is much higher than the average ratio (26.3%) of the other 15 categories. The average of unauthorized hotlinked images per site is 9.00, much higher than the average (2.55) of the other 15 categories. It is not surprising to observe that hotlinking is severe in the blogging category. The reasons are: (1) many blog sites are operated by individuals who are not professional web developers. They tend to use search engines to look for pictures and then directly link them instead of hosting them. Besides laziness, they do not have the clear conscience on copyright infringement their (unauthorized) behaviors have done. (2) many blog sites do not have enough online storage and traffic quota, and have to intentionally hotlink from other sites. On the other hand, the proportion of blogging sites that have authorized hotlinking behavior is 91.1%, which is very close to the average (94.5%) of the other 15 categories. This is because blog sites also have needs of regular site interaction, such as online advertising and traffic monitoring. However, the average of authorized hotlinked images per site is 32.35, much higher than the average (11.98) of the other 15 categories. Blog sites link a large amount of small images for user avatars and face expressions. Furthermore, blog pages display more ad images than regular webpages.

For those sites with unauthorized hotlinking behaviors (456 in our measurements), Fig. 5 shows the cumulative distribution function (CDF) of the number of unauthorized images hotlinked per homepage. Around 81.2% of the site homepages hotlink at least 10 images. While the maximum value of outdegree is 221, the mean is 6.52, and the standard deviation is 13.36. The fact suggests that, if a site conducts unauthorized hotlinking, the behavior may be regular instead of occasional. The corresponding CDF curve of authorized hotlinking is also shown in Fig. 5 as a reference. The maximum value of outdegree in the authorized curve is 469, the mean is 18.63, and the standard deviation is 31.90.

Since a great amount of images are hotlinked all over the Internet, besides "hotlinking culprit", we also want to know the general distribution of victim sites. For images hotlinked without authorization in our measurement, we get URLs of their hosting sites. Fig. 6 shows Top-Level Domain (TLD) distribution of victim sites. The majority of victims belong to the .com domain (78.8%). The second largest victim domain is the .cc domain (11.0%). The remaining domains account for a small portion (10.2%).

Since image hotlinking is severe in the blogging category, we conduct a deep measurement which targets images hotlinked by the site, not merely by the homepage. More specifically, we pick up top 10% sites (namely, 40 out of 400) in the blogging category based on the number of hotlinked images per homepage. We modify wget [20] to crawl these top 40 sites. For each site, starting from the homepage, the crawler visits the internal pages linked in the homepage in a recursive way, with the recursive depth set as three[7]. After acquiring an (incomplete) internal page list of a site, we still use our Firefox extension to visit all these pages to accumulatively log images hotlinked by the site. Fig. 7 shows the total number of hotlinked images for the top 40 blogging sites we measured. A site surprisingly hotlinks 222 images. Averagely speaking, each of these 40 blog sites hotlinks about 25 images. It confirms again that these target blog sites do hotlink a large amount of images without authorization.

Top-Level Domain	No. of Unique Victim Sites
com	2,453 (78.8%)
cc	343 (11.0%)
net	161 (5.2%)
org	118 (3.8%)
gov	6 (0.2%)
misc	31 (0.9%)
Total	3,112 (100%)

Fig. 6. Unique victim site distribution by TLD (16 categories).

Based on the above measurement results, we can draw the following conclusions. (1) The behavior of hotlinking images is very common over the Internet. (2) For the majority of top sites in the first 15 categories except blogging measured by us, they have their own servers for image storage. Linking images hosted by other sites is usually for web interaction (such as advertising and traffic tracing). Such behaviors are authorized by hosting sites. (3) A small proportion of sites (such as blog sites measured above) hotlink a large amount of images without authorization. This unethical behavior infringes the interest and rights of hosting sites.

3.2. Measurement of hotlinked software packages

Among the various types of files hotlinked over the Internet, the software installation packages (such as .exe and .rpm) are more prone to becoming hotlinked. Due to the large file size, hotlinking may cause significant resource consumption to hosting sites. Therefore, software package is selected as the measurement target in the file category of hotlinking. In the previous image-centric measurements, we start from hotlinking sites and trace back to victim sites. In this part, we reverse the order and trace from victims to hotlinkers. The measurement procedure is explained as follows.

Based on download times and popularity, we choose 100 top free software packages as suggested by [21,22]. Our software set

[7] We believe that the 3-depth-level is a good balance between site coverage and measurement time. A larger depth will incur much greater running time in a nearly exponential way and also generate more duplicate pages.

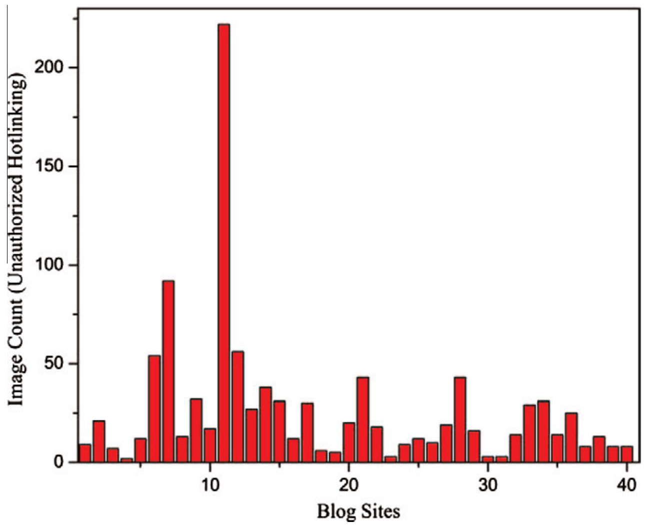


Fig. 7. Count of hotlinked image without authorization for top 40 blog sites.

covers various categories ranging from security software to developer tools. For each software package, we manually obtain its official download URL that usually belongs to its author/publisher site. We use a PHP-written script to search webpages that contain official download URLs via the standard query API provided by Google. We argue that, including the official download URL on a third-party page is a kind of hotlinking, because the visitor can directly download the software by clicking on the hotlinked URL without visiting the official download page. In most cases, such hotlinking behaviors are not authorized or expected by software owners. Software owners may use CDN (content delivery network) sites to spread files for faster download. However, this case is not hotlinking at all, since CDN sites do host software packages by themselves.

For each package, we generate a log including the first 100 results (namely, URLs of hotlinking pages) returned by Google. The result ranking is decided by search relevance and popularity. Thus, we believe that the top 100 results are appropriate for the measurement sampling purpose. If the number of the results returned is less than 100, we take the actual number. After collecting logs of these 100 software packages, we use our Java-written toolkit to process them and do a comprehensive analysis presented as follows.

In our measurement result set, the 100 software packages are hotlinked by 3020 sites in 7539 unique webpages. For the above hotlinking sites, Fig. 8 shows the CDF of the number of hotlinked software packages per site. Around 33.8% of the sites hotlink two or more software packages (may in different pages). The maximum value of the CDF curve is 128. It is a forum-style site where "warm-hearted" users publish 128 hotlinking download URLs (including duplicates) in multiple posts. The curve mean is 2.50 with the standard deviation of 5.79.

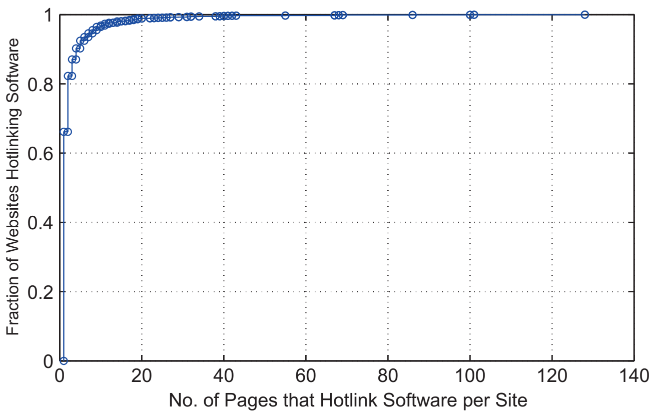


Fig. 8. CDF of the number of hotlinked software packages per site.

Table 3
TLD distribution of unique sites that hotlinking software packages.

Top-level domain	No of unique hotlinking sites
com	1859 (61.6%)
cc	481 (15.9%)
net	372 (12.3%)
org	202 (6.7%)
info	42 (1.4%)
misc	63 (2.1%)
Total	3020 (100%)

Moreover, Table 3 lists the distribution of the 3020 hotlinking sites by Top-Level Domain. The majority are from the .com domain (61.6%). Subsequently, the .cc and .net domains contribute 15.9% and 12.3%, respectively. The remaining domains account for 10.2%.

The above analysis manifests that hotlinking software packages is a common problem over the Internet. It also happens to other types of file resources, such as documents and audio/video clips. Because of the large file size, frequent hotlinking incurs significant consumption in network bandwidth and computing resources. It is essential to deploy anti-hotlinking methods to protect hosting sites.

3.3. Postmortem analysis of a hotlinking attack

To fully understand the damages caused by hotlinking towards hosting sites, such as system burden and traffic theft, we collected raw traces from a victim server and performed a forensic-style postmortem analysis on a real hotlinking attack.

The hotlinking attack is briefly described as follows. One user of the victim server hosted a folder of many images of a popular commercial product under his web directory. He posted a lot of articles containing some of the above images at a few third-party sites. Due to the attractive contents and images, those articles were frequently referred by others. Eventually they drew a large amount of click traffic. Since the images were hosted at the victim server, numerous image requests were redirected to it, consuming many computing and network resources. Finally, the victim server was overwhelmed and crashed. The user was not conscious of his hotlinking behavior and the serious damages induced to the server. It is the system administrator who noticed the crash of the victim server and blocked the public Internet access to that user's directory, which ends the hotlinking event lasting in a continuous period of 40 days from the first week December, 2008 to the early January, 2009.

The victim server is installed with Apache 2.2.4 running on an Intel Xeon 64-bit workstation, which is equipped with quad-processors of 3 GHz, 12 MB L2 cache, 16 GB memory, 300 GB hard disk and 1 Gbps LAN connection. The victim server grants the direct HTTP access to any files in a user's web directory if the file path is given correctly. Namely, hotlinking is allowed, and no anti-hotlinking defense is deployed. The image folder contains around 1000 images in the format of JPEG. The size of a single image varies from 50 to 300 KB, and the total size of the image folder is about 130 MB. Each entry in the raw server log records the response to an HTTP request, including fields like the client IP address, timestamp, the file path of the requested object at the server, and the Referer of the HTTP request. The total size of the 40-days logs is 400 MB. After removing those entries irrelevant to image hotlinking, we reduce the total log size to around 200 MB.

With the help of the Referer field, we trace back to the hotlinking source. A set of articles were posted at some third-party sites,

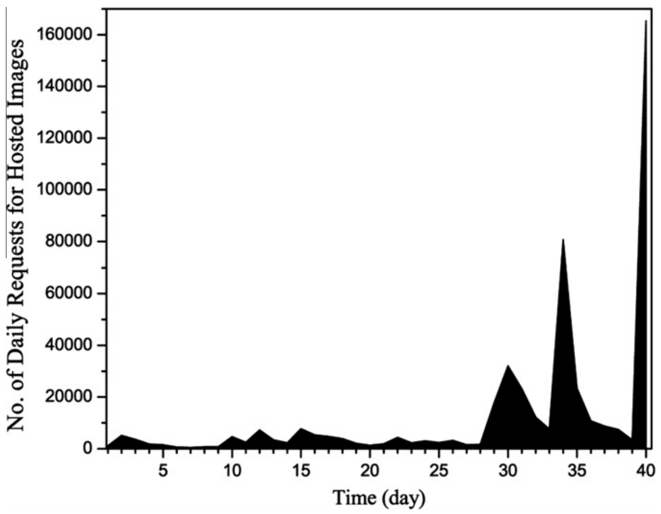


Fig. 9. Daily traffic in terms of image requests.

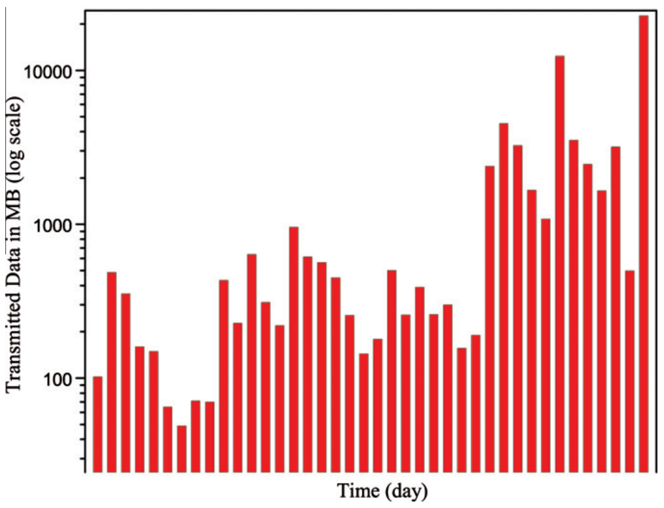


Fig. 10. Daily data transmission (in MB) caused by hotlinking.

more specifically, nine BBS-style forums and one EBay-style shopping site. The article pages included images hosted on the victim server in the form of <img src="apache/imgs/sample.jpg">. In this way the hotlinking relationship forms between those third-party sites and the victim server. Most of those sites have a large user population, and generate a large trace of user requests. The number of images embedded in the article pages ranges from 5 to 80. One page hotlinks 80 images with the total size of 12 MB. When the browser displays the page, it sends an HTTP request for each embedded image to the victim server. The server returns the image.

Fig. 9 shows the number of daily requests for hosted images. The average number of daily requests over the 40-day window is 11,872. The daily number gradually increases, and reaches the summit of 165,430 on the last day when the victim server was crashed. The curve development can be plausibly explained as follows. With the increase of user views and replies, the rank of the article boosts. As a case of Matthew Effect (i.e., the rich get richer and the poor get poorer), it draws more users, bringing more click traffic. There are three spikes over the last 10 days. We manually checked the posts during that period, and found the obvious evidence of scripts[8] that automatically post replies to articles to keep them staying in the first few pages of the forums, thus attracting many more viewers.

Fig. 10 shows the daily data transmission caused by hotlinking. We first compute the data transmission, *Ti*, caused by a single file, as $Ti = Fi^{*}Ri$, where *Fi* is the size of the file *i*, and *Ri* is the number of requests for the file *i*. The daily data transmission is calculated as the sum of that of all the files requested that day, i.e., $Td = \sum_i Ti$. The curve in Fig. 10 is similar to that in Fig. 9. The data transmission increases gradually over the first 30 days, and surges drastically over the last 10 days. The average daily data transmission caused by image hotlinking is around 1.7 GB. The maximum occurs on Day 40 at 22.7 GB, followed by Day 35 at 12.3 GB.

Fig. 11 illustrates the number of daily client IP addresses, which can be used to roughly estimate the number of daily visitors. The average of daily IP addresses is 490, while the maximum of 2215 appears on Day 35 followed by the second highest of 1950 on Day 40. Over the 40 days, on average, each IP address requested 24 hotlinked images.

[8] For example, the reply interval is short and almost fixed. Furthermore, many recently registered accounts are used.

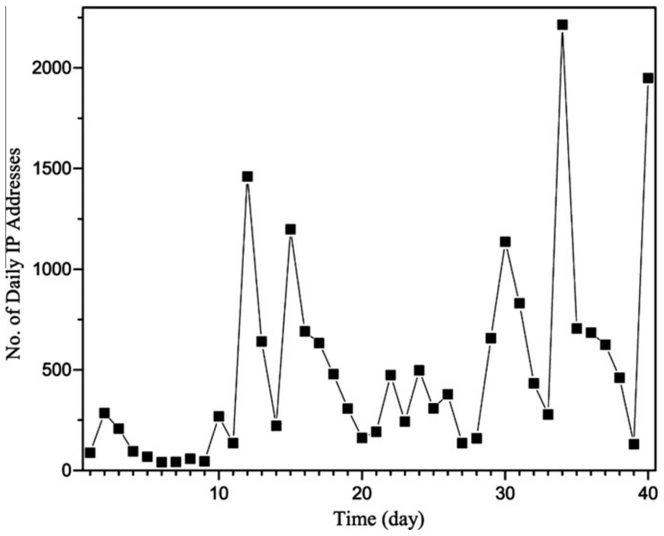


Fig. 11. Daily traffic in terms of client IP addresses.

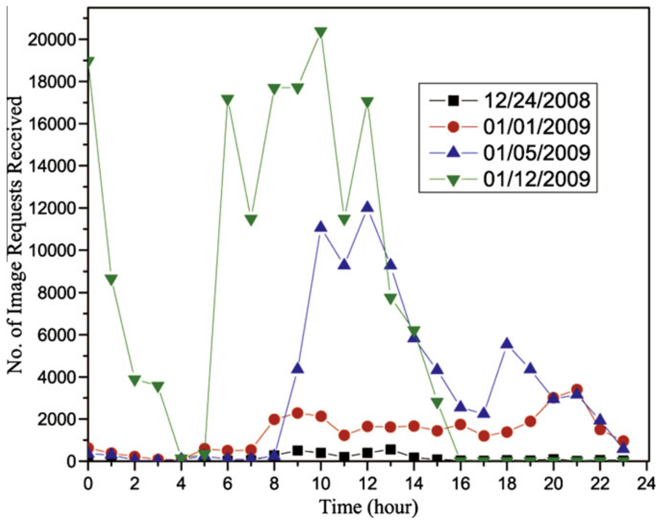


Fig. 12. The hourly traffic distribution of four selected days.

Among the 40-day logs, we choose 4 days for more detailed analysis. The result is presented in Fig. 12. Days 30, 35, and 40 are selected due to their top data transmissions. The number of the daily IP addresses of Day 24 (497) is the closest to the average (490), and thus that day is used as the average case. The curve shapes of the first 3 days are similar. The peak appears from 9:00 to 15:00, followed by another wave from 17:00 to 21:00. This matches the regular timetable of human visitors, since more people tend to surf the Internet during working hours and in the evening. Day 40 is a special case, where starting from 8:00, the hourly traffic reaches 18,000 requests with the summit at 20,000 (namely 5.6 requests per second). The victim server is overwhelmed by the large amount of incoming HTTP requests. In the following period, its performance drops sharply and the response becomes extremely slow. The server crashes around 16:00. The system administrator restarts the server manually and blocks the public access to the user image folder afterwards.

4. Framework design

Based on the existing network security techniques, we present an anti-hotlinking web framework for hosting sites. Our design goal is to greatly increase the hotlinking difficulty and to defeat most common forms of hotlinking with easy deployment. In reality, webmasters can control the granularity of anti-hotlinking by defining different protection policies and applying them to the framework based on the requirements of their sites. Currently, the framework provides two policies, *Strict Policy* and *Loose Policy*, for the demonstration purpose. In this section, we first describe the design details of the framework and then present the two protection *policies*.

4.1. Design details and modules

Fig. 13 illustrates the anti-hotlinking framework that consists of three major modules. The *HTTP Request Filtering Module* filters incoming HTTP requests and blocks direct access to hosted resources. The file entrance page contains the other two modules. The *Session Creation/Authentication Module* creates and manages sessions to maintain the HTTP communication status between the server and client. Different protection policies may require different steps, and the user must complete them to become eligible for downloading. The *Download Authorization Module* checks the download authorization log and determines whether the download request can be granted. If so, the server will return the file to the client. The intra-page communication of the file entrance page is supported by the AJAX techniques. The page address in the browser does not change, and only the page content in the inner window updates smoothly. The detailed description of these modules are given as follows.

4.1.1. HTTP request filtering module

The main function of this module is to transform the incoming HTTP request into the legal form required by the framework and to block the direct access to hosted resources. The module uses the following three functional blocks.

4.1.1.1. Unique file ID and entrance page. The site assigns each of its hosting files a unique file ID, and this solves the problem of duplicate file names. Currently, our framework uses three types of information to generate the unique file ID: original file name, file upload timestamp and user ID (either the user account name or the IP address) where appropriate. The site then hashes the above information into the file ID and maintains a file storage structure table as shown in Fig. 14. A unique entrance page can be created for each file based on its file ID.

4.1.1.2. Limiting HTTP requests. Most sites without anti-hotlinking defense directly grant HTTP requests for hosted resources as long as the resource path is given correctly. Limiting HTTP requests helps the site block the direct access to hosted materials. As an option, the site can use URL Rewriting of HTTP requests to achieve it. Suppose now the framework only directly grants HTTP requests for webpages (whose extensions are .htm and .php in our prototype) and some accessory web objects required for normal page display (such as background images), while requests for all other types of objects are redirected to corresponding entrance pages based on unique file ID. For example, the direct HTTP request for the file install.rpm in the form of www.site.com/files/install.rpm is prohibited and rewritten into www.site.com/download.php?fid=T85X4PNS.

4.1.1.3. Handling different HTTP requests. The module may receive three types of HTTP requests for a file, and finally it transforms them into Legal Request, the only request type accepted by the framework. The three types of HTTP requests are listed as follows. (1) *Legal Request* (in the form of www.site.com/download.php?fid=T85X4PNS, with a valid fid) is directly brought into the file entrance page. This format is always used by all internal pages of the hosting site. (2) *Hotlinking Request* (in the form of www.site.com/files/install.rpm, without a fid) that directly requests files is rewritten into Legal Request based on the unique file ID mapping and then redirected to the file entrance page. (3) *Invalid Request* (in the form of www.site.com/download.php?fid=S85X4PNS, with an invalid fid) is redirected to a file-list page that shows legal download links of hosted files on

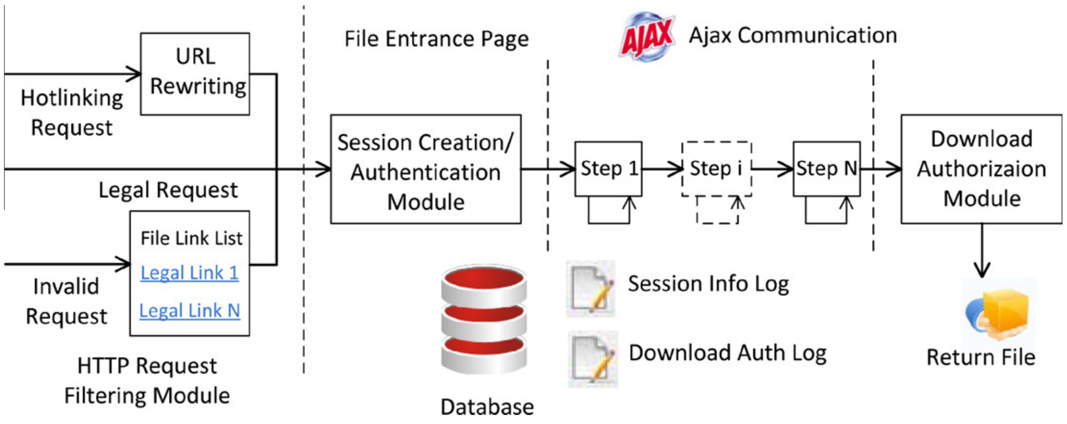


Fig. 13. Anti-hotlinking framework overview.

Unique File_ID	File Name	Storage Path
T85X4PNS	install.rpm	root/files/

Fig. 14. File storage log.

the server. When a user clicks on a legal download link, it generates a Legal Request and the user can continue to download.

4.1.2. Session creation/authentication module

The anti-hotlinking framework needs to maintain the interaction status between the client and server to determine whether the client becomes qualified to download the requested file. Both cookie and session mechanisms can be used to maintain HTTP communication states. Our framework chooses to use the session mechanism. The main reason is that, the session mechanism only has to store session ID on the client-side, and maintains most session information on the server-side. It greatly reduces the risk of the client-side forging and hacking. Session ID can be stored on the client-side in the form of cookie or URL parameter. In either way, the server must authenticate the received session ID to make sure it is not only valid, but also originally assigned to the client. Because HTTP sessions build upon TCP connections, this module uses the essential information extracted from the underlying TCP connection along with browser signature to perform session ID authentication. The TCP information mainly includes the IP addresses and port numbers of source and destination. For every session it initiates, the server creates an entry in the form of [session ID, TCP info, client browser signature] in the *Session Info Log*.

When the HTTP request arrives at the file entrance page, the module determines whether the client has an existing valid session by authenticating the session ID. The authentication procedure is shown as Fig. 15. If the authentication succeeds, the module creates an entry in the *Download Authorization Log* and the client can start to execute the required download steps. Requirements specified by different protection policies may vary. We give an example set of steps in Section 4.2. After the steps are fulfilled, *Download Authorization Module* decides whether the download request will be granted or not.

4.1.3. Download authorization module

The *Download Authorization Log* is a core log that stores information used by the site to decide whether the file request can be granted or not. Table 4 lists a simplified log entry. Session_ID is associated with the client-side information (mainly about TCP connection and browser signature), and File_ID is unique to each file. The first two fields together show a specific client requests to download a specific file. The Step_i field is set as true after the corresponding step is completed. After all the required steps are finished, the last three fields are set as follows. The *Authorization* field is turn into true, the *Authorization Code* field is filled with a random string, and the *Expire Timestamp* field carries a timestamp that specifies when the *Authorization Code* expires. The *Download Authorization Log* uses Session_ID and File_ID to locate the corresponding entry in the *Download Authorization Log*. If (1) the *Authorization* field is true, (2) the *Authorization Code* field is valid, and (3) the *Expire Timestamp* field is greater than the current timestamp, then the module authorizes the download request.

The module uses File_ID to locate the associated file entry in the *File Storage Log*, and reads the file via the full file path. The site constructs an HTTP response containing the requested file and returns to the client. The framework provides the following two options to handle the *Authorization Code* field after the file request is granted. The first is the *One-time Code Use* option. Namely the code may only be used once and expires after then. If the same user wants to download the same file again, he will be redirected to Step_1 to start over even though the session is still alive. The second is the *Repeated Code Use* option. The code can be repeatedly used untill it expires. This option enables the user to download the same file multiple times during a certain time window. To prevent the code from being abused, the web server can limit both the maximum number of simultaneous download connections per IP address and the download speed per process.

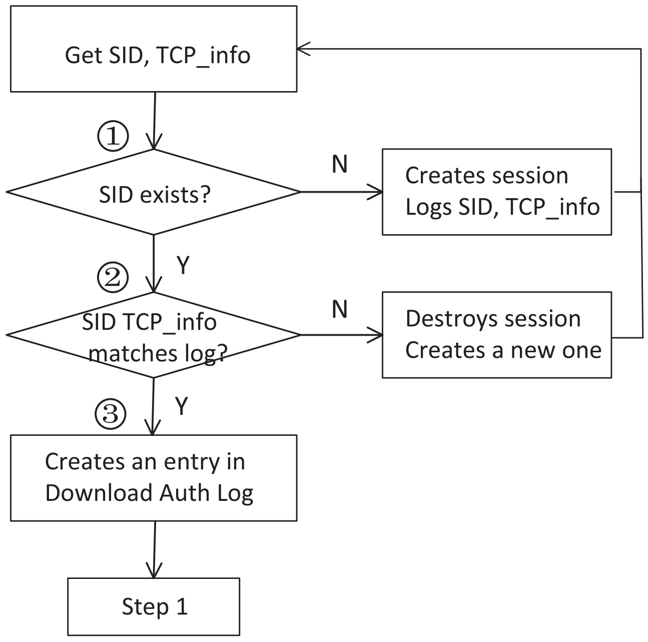


Fig. 15. Session creation and authentication.

4.2. Strict policy

Hosting sites often accommodate high-importance and large-size resources, such as software packages and documents. Hotlinking such resources may cause serious damages to hosting sites. We introduce *Strict Policy* to protect this type of resources as follows.

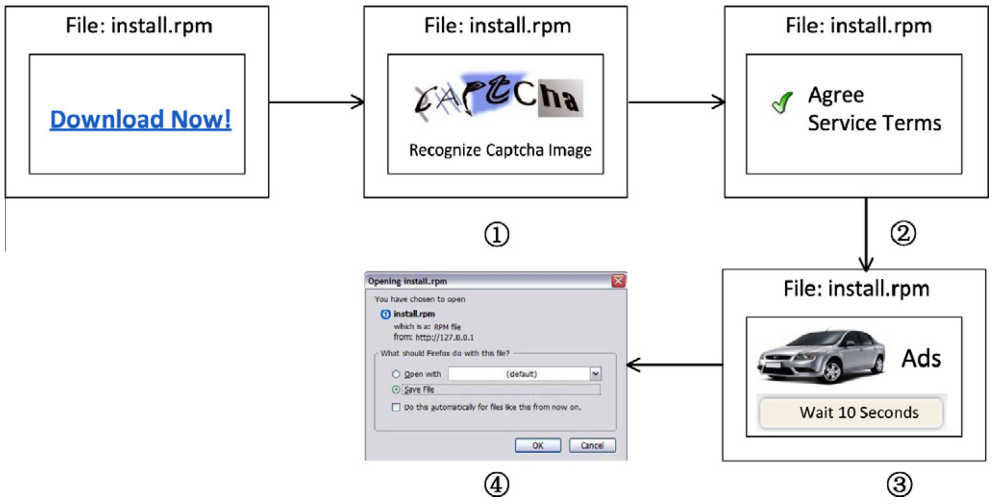
The *HTTP Request Filtering Module* filters the incoming resource request into the form of Legal Request. When the request is directed to the *File Entrance Page*, the *Session Creation/Authentication Module* guarantees an authentic session between the server and client.

The key feature of *Strict Policy* is that, it specifies a set of steps the user must fulfill to become eligible for downloading. We emphasize that, the policy can generate as many steps as needed and customize requirements for each step. These steps can serve different goals for hosting sites in reality, like distinguishing human users from machines, displaying sponsors ads, and encouraging users to buy premium download accounts, etc. We leave up to the webmaster the details of designing steps. Our framework implements a combination of three simple steps as shown as Fig. 16 purely for the demonstration purpose. The user must (1) correctly recognize a CAPTCHA image, which distinguishes human users from machines,[9] (2) check the Terms of Service checkbox, and (3) wait a short time to activate the download link. The *Strict Policy* sets the download authorization code to be one-time use. If the user wants to download the same file again, it has to repeat the steps to generate another authorization code.

[9] Using CAPTCHA is an option of distinguishing human users from machines. Its effectiveness against automated clicks is out of the scope of the paper.

Table 4
Download authorization log.

SID	FID	Step_1	Step_i	Step_N	Auth	Auth_code	Expire_TS
5k0642	T85X4PNS	True	True	True	True	d383e3	1227213501



Opening install.rpm
You have chosen to open
install.rpm
which is a: RPM file
from: http://127.0.0.1
What should Firefox do with this file?
Open with (default)
Save File [✓]
Do this automatically for files like this from now on.

Fig. 16. User download procedure.

4.3. Loose policy

While the *Strict Policy* is stringent and suitable for protecting the resources of great importance, it is too cumbersome for users to download these widely-used web objects such as images. As an alternative, we present a light-weight policy, *Loose Policy*, for protecting common web objects of less importance. Its design principle is to strike a good balance between resource protection and anti-hotlinking cost.

Suppose hosted objects (such as images) are included in some webpages of the hosting site. After a browser loads and parses the webpage, it sends a request for the embedded object to the hosting site. According to Fig. 13, the server rewrites the incoming HTTP request into Legal Request (like www.site.com/loose_download.php?fid=57OUK14N). The download control page (loose_download.php) calls the *Session Creation/Authentication Module* to judge whether the browser has an active session with the server or not. If so, the *Download Authorization Module* will return the object to the browser. Note that the process of executing steps is turn off in the *Loose Policy*. The whole download control procedure does not involve any user interaction, and is totally transparent to users. This is a big difference from the *Strict Policy*. Furthermore, the *Download Authorization Module* takes the option of *Repeated Code Use*. This allows the browser for multiple downloads towards the same object during the session lifetime.

If there is no active session between the browser and hosting site, it is very likely the direct object request is triggered by a hotlinking webpage.[10] The control page will redirect the browser to a default page that builds an active session and displays some notification information like *the requested object is originally hosted at* www.site.com (*click here to visit the genuine hosting site*). The page stays for a short time period (like 3 s) customizable by the site. After that, the request will be granted and object will be sent to the browser to appear in the page.

[10] It is almost impossible for a user to know the object URL that is long and contains random strings without visiting the hosting page. There is a slim chance that the user enters the object URL on the browser to directly visit it. In this situation, we think that the user intends to visit the hosting site. The redirection action will happen, but it does not hinder the user's browsing experience.

5. Prototype implementation

In this section, we describe the implementation details of our anti-hotlinking framework prototype. It is deployed on the server-side and does not require modifications on the client-side. It only requires browsers to run a small amount of assistant JavaScript codes that are supported by modern out-of-box browsers.

5.1. Web server setup

We choose Apache (version 2.2.8) [23] as the web server based on which our anti-hotlinking prototype is deployed. It runs at a workstation with Intel dual-processor 2.2 GHz and 2 GB of RAM. Three additional modules, mod_limitipconn, mod_bandwidth and mod_rewrite, are added to the server. The mod_limitipconn module [24] limits the maximum number of simultaneous download connections per IP address. Similarly, the mod_bandwidth module [25] limits the download speed of a single connection. The mod_rewrite module [9] rewrites the requested URL on-the-fly based on configuration rules with the style of Perl formal expressions. In our framework, rewriting HTTP requests is achieved by modifying .htaccess file and putting it into proper directories.

5.2. Technical details

We use PHP 5.2.5 [26] as the back-end programming language. Most webpages in our framework are written in PHP. Our framework uses PHP functions to implement session communication, and to store related variables in the $_SESSION array. Session_ID can store on the browser in the form of cookie. If the browser disables cookies, PHP can detect it and transfer Session_ID to the browser as a query parameter of URIs. The webmaster can enable the PHP Transparent Session Support function by turning on the statement *session.use_tran_sid*=1 in the php.ini configuration file.

In the current PHP page, Session_ID will automatically append to all the internal URIs of the hosting domain. Another important function module, CAPTCHA [27], is also implemented in PHP. We use a pseudo-random algorithm to generate a string consisting of a fixed number of random characters. They are drawn on the image in a random font with the help of the GD library [28]. Some random background noise is also added to thwart character recognition algorithms.

Our framework uses a small amount of JavaScript codes that are embedded in webpages and run at the client-side. It provides a few additional functions to enhance the user experience, and does not affect the core download control procedure of the framework at all. If the browser disables running JavaScript, a client can still finish the download procedure. The framework provides Ajax-style (Asynchronous JavaScript and Xml) [29] intra-page communication based on prototype.js [30]. The advantage brought by Ajax is that, it retrieves data from the server asynchronously in the background without interfering with the display of the current page.

6. Evaluation

In this section, we present the system evaluation of the proposed framework in terms of security and usability, with the focus on the *Strict Policy*.

6.1. Security analysis

From the security perspective, the *Strict Policy* provides a much stronger defense against hotlinking attacks than the *Loose Policy*. We use the *Strict Policy* to demonstrate the effectiveness of our anti-hotlinking framework against the four different types of hotlinking attacks presented in Section 2.1, respectively.

6.1.1. Effectiveness against Direct Hotlinking

Since the site applies URL Rewriting towards incoming HTTP requests, the request for a file will be redirected to the download entrance page. The server does not directly return the file in response to the HTTP request. Therefore, the attack of *Direct Hotlinking* will be defeated.

6.1.2. Effectiveness against hotlinking via referer fabrication

The site does not use the HTTP_REFERER field in the received HTTP request header to determine whether it should return the file or not. Instead, it uses the control procedure shown in Fig. 13 to make the decision. Thus, fabricating an HTTP_REFERER by using the domain of the hosting server does not help hotlinking at all.

6.1.3. Effectiveness against hotlinking via cookie vulnerabilities

To demonstrate the effectiveness against a hotlinking attack that explores cookie vulnerabilities, we suppose Session_ID is stored on the browser in the form of cookie. The hotlinking page, H.com/h.htm, can contain an iFrame from the victim site V.com to make the browser establish a session with V.com. When the browser requests a file that is hotlinked in h.htm and hosted by V.com, V.com will call the *Session Creation/Authentication Module* to validate the current session. The session validation will pass. However, the hotlinking will fail to execute the required steps (shown in Fig. 13). The reason is that, the same-origin policy prevents the malicious code on h.htm to access the iFrame of V.com. Therefore, the malicious code cannot complete the required steps, such as CAPTCHA image recognition, even if it has the ability of automated scripting. Since the hotlinking page disables the user to view the legal page by setting the iFrame hidden, the user will not help finish the required steps either.

6.1.4. Effectiveness against hotlinking via session vulnerabilities

Suppose the malicious code shown in Fig. 4 manages to steal the session ID, and append it to the file download hotlink. The last two HTML statements in Fig. 4 can be modified as Fig. 17 shows. Now the legal file request along with the valid session ID and file ID is sent to the web server. It can pass the check of *Session Creation/Authentication Module*. However, the *File Entrance Page* (namely, download.php) cannot be displayed on the client-side. It is discarded by the browser because the returned file type is HTML document rather than image expected by the <img> tag. The download steps on this page cannot be completed by the user. Thus, this type of hotlinking attacks will also be defeated.

6.2. Usability analysis

After the *Strict Policy* is enforced on the hosting site, the download request for the hotlinked file will be redirected to the specific entrance webpage. First, the site creates a session with the browser or validates the existing session. This procedure is totally transparent to the user, and the induced time delay is hardly to be noticed. Second, the user needs to fulfill some steps required by the hosting site before he becomes qualified for downloading. The time consumption of this procedure is determined by the details of the requirements. In our prototype, the three example steps take around 20 s to complete.[11] On one hand, this is unavoidable trade-off a hosting site has to make; on the other hand, the hosting site must design the download steps carefully to maintain the users' interests of the site.

The *Loose Policy* is responsible of delivering the requested object (often images) to the client browser. It does not involve any user interaction. When the server receives an object request, the Loose Policy performs additional steps listed as follows. In comparison with the direct HTTP response, it causes some minor delays.

- If there is no active session, the server needs to create a session for the client, and logs related information to the session file. The time consumption is labeled as Delay_1.
- If the session ID rendered by the client does not match the server log (i.e., it is stolen from others), the server will delete the current session and create a new one. The time consumption is labeled as Delay_2.
- If the session ID is legal, it will pass the server authentication. The time consumption is labeled as Delay_3.

To measure the delay in each of the above cases, we use a Firefox browser to visit a Apache server protected by the *Loose Policy*. We conduct 50 visits for each case. The server records the timestamps (in millisecond) of the event start and end, and calculates the delay. On average, Delay_1 is 10.0 ms, Delay_2 is 7.3 ms, and Delay_3 is 11.1 ms. Overall, the additional delays induced by the *Loose Policy* are minor and hard to be noticed by the client.

7. Related work

We have already discussed several common hotlinking attacks in Section 2, and in this section we survey related work in a boarder scope.

Hotlinking against session protection shares much in common with web session hijacking. Session hijacking [31] is an attack of taking over a user session by stealing the session ID and impersonating the authorized user. After that, the attacker gains access to the sensitive information stored in the session. The main counter-

[11] The third step requires the user to wait 10 s, which can be skipped by the webmaster in reality.

```
<img src="www.V.com/download.php" style="display:none">
<a id="fileLink" href="www.V.com/files/download.php?fid=T85X4PNS&sid=5k32d0"
    >Download RMP Package</a>
```

Fig. 17. Modified snippet exploring session vulnerability.

measure is improving session management on both server-side and client-side to protect session ID. End-to-end secure channels like SSL can prevent session ID from being intercepted by passive eavesdroppers. The drawback of SSL is that, it brings much additional cost to the communication [32]. As a result, many sites only use SSL to protect the initial login page, and the following communication is conducted over plain HTTP. The session ID may be exposed in the insecure network again.

Sessionlock [33] provides a light-weighted approach to securing web sessions against eavesdropping. After the server sets up the session with the client, it stores the session ID in the client browser as the form of fragment identifier. The browser signs outgoing HTTP requests with the fragment identifier via HMAC to present its identity to the server. Since the browser never sends fragment identifier over the Internet, the eavesdropper cannot intercept the session ID. However, this method is not appropriate for fighting against hotlinking that explores session vulnerability. The hotlinking site can set up a legal session with the hosting site, and embeds the assigned session ID into hotlinking pages for the user to share. In other words, the user can send requests signed by the legal session ID without visiting the hosting site.

Cookie vulnerabilities exploited by hotlinking is in the field of cookie security. HTTP cookies have serious security and privacy concerns [34]. Cookie theft is the act of intercepting cookies by an unauthorized party. Cookies may be stolen via packet sniffing over the Internet. This scenario is similar to session hijacking. Cross-site scripting attack is another way to steal cookies Wassermann et al. [35], Jim et al. [36]. Usually the attacker posts malicious code into a webpage, and by running it, the browser itself will send cookies to the attacker. HttpOnly flag is the countermeasure that protects cookies from cross-site scripting, and it makes cookies inaccessible to client-side programs (such as JavaScript). It was first introduced by Microsoft [37] and supported in PHP since version 5.2.0. However, it has not become the industry standard.

Currently, there are some web sites that provide online storage and file delivery service, such as [38,39]. They have also deployed anti-hotlinking measures. However, their implementation details remains sealed.

8. Conclusion

In this paper we investigate the hotlinking phenomenon with the focus on unauthorized hotlinking. We perform a series of large-scale measurements targeting two types of hotlinked objects, images and software packages. Our measurement results show that hotlinking widely exists over the Internet and is severe in some categories of websites like blogging. We also conduct a detailed postmortem analysis on a real hotlink–victim site, which shows that unexpected large amount of hotlinking traffics can easily overwhelm the victim server. Moreover, we analyze a set of regular hotlinking attacks that explore the weakness of current defense methods. To defend against hotlinking attacks, we present an anti-hotlinking framework based on the existing network security techniques. The framework is highly customizable with different granularities of protection that webmasters can specify. A prototype of the framework is implemented with the support of the two download policies, *Strict Policy* and *Loose Policy*. Its effectiveness against hotlinking attacks is evaluated in terms of security and usability.

References

[1] Inline linking (leeching, bandwidth theft), Wikipedia. Available from: <http://en.wikipedia.org/wiki/Inline_linking>.
[2] Google AdSense. Available from: <www.google.com/adsense>.
[3] Yahoo! Advertising. Available from: <http://advertising.yahoo.com/>.
[4] Lee S. Strickland, Copyright's Digital Dilemma Today: Fair Use or Unfair Constraints? Part 2: The DMCA, the TEACH Act and Other E-Copying Considerations. Available from: <http://www.asis.org/Bulletin/Dec-03/strickland.html>.
[5] The Digital Millennium Copyright Act of 1998. Available from: <www.copyright.gov/legislation/dmca.pdf>.
[6] Collin Jackson, Andrew Bortz, Dan Boneh, John C. Mitchell, Protecting browser state from web privacy attacks, in: WWW '06: Proceedings of the 15th International Conference on World Wide Web, 2006, pp. 737–744.
[7] The Hypertext Transfer Protocol (HTTP), RFC2616. Available from: <www.w3.org/Protocols/rfc2616/rfc2616.html>.
[8] HTTP State Management Mechanism, RFC2109. Available from: <www.ietf.org/rfc/rfc2109.txt>.
[9] The Apache Module mod_rewrite URL Rewriting Engine. Available from: <httpd.apache.org/docs/1.3/mod/mod_rewrite.html>.
[10] Mike Ter Louw, V.N. Venkatakrishnan, Blueprint: precise browser-neutral prevention of cross-site scripting attacks, in: 30th IEEE Symposium on Security and Privacy, Oakland, CA, USA, 2009.
[11] Alexa. Available from: <www.alexa.com/>.
[12] Blog Top Sites Ranking. Available from: <http://www.blogtopsites.com/>.
[13] BlogFlux Top Blog Sites Overall Statistics. Available from: <http://topsites.blogflux.com/stats.php>.
[14] Google Syndication. Available from: <http://googlesyndication.com/>.
[15] 2mdn. Available from: <http://2mdn.net/>.
[16] DoubleClick. Available from: <www.doubleclick.com/>.
[17] Akamai, Web application acceleration and performance management. Available from: <www.akamai.com/>.
[18] Flickr. Available from: <http://www.flickr.com/>.
[19] CacheFly, Content delivering. Available from: <www.cachefly.com>.
[20] GNU Wget. Available from: <www.gnu.org/software/wget/>.
[21] Free Software Downloads and Reviews by Download.com. Available from: <http://download.cnet.com/Best-Free-Software/1200-20-5154518.html>.
[22] The Best Free Software of 2009, Features by PC Magazine. Available from: <www.pcmag.com/article2/0,2817,2338803,00.asp>.
[23] The Apache HTTP Server Project. Available from: <httpd.apache.org/>.
[24] The Apache module mod_limitipconn.c. Available from: <http://dominia.org/djao/limitipconn.html>.
[25] The Apache module mod_bandwidth. Available from: <www.cohprog.com/mod_bandwidth.html>.
[26] PHP, Hypertext preprocessor. Available from: <www.php.net/>.
[27] CAPTCHA: Telling Humans and Computers Apart Automatically. Available from: <www.captcha.net/>.
[28] GD Graphics Library. Available from: <www.boutell.com/gd/>.
[29] AJAX, Asynchronous JavaScript and Xml. Available from: <https://developer.mozilla.org/en/AJAX>.
[30] Prototype JavaScript framework. Available from: <www.prototypejs.org/>.
[31] Robert C. Newman, Cybercrime, identity theft, and fraud: practicing safe internet - network security threats and vulnerabilities, in: InfoSecCD '06: Proceedings of the 3rd Annual Conference on Information Security Curriculum Development, 2006, pp. 68–78.
[32] Homin K. Lee, Tal Malkin, Erich Nahum, Cryptographic strength of ssl/tls servers: current and recent practices, in: IMC '07: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, 2007, pp. 83–92.
[33] Ben Adida, Sessionlock: securing web sessions against eavesdropping, in: WWW '08: Proceeding of the 17th International Conference on World Wide Web, 2008, pp. 517–524.
[34] Hal Berghel, Hijacking the web, Commun. ACM 45 (4) (2002) 23–27.
[35] Gary Wassermann, Zhendong Su, Static detection of cross-site scripting vulnerabilities, in: ICSE '08: Proceedings of the 30th International Conference on Software Engineering, 2008, pp. 171–180.
[36] Jim Trevor, Swamy Nikhil, Hicks Michael, Defeating script injection attacks with browser-enforced embedded policies, in: WWW'07: Proceedings of the 16th International Conference on World Wide Web, 2007, pp. 601–610.
[37] Mitigating Cross-site Scripting With HTTP-only Cookies. Available from: <http://msdn.microsoft.com/en-us/library/ms533046.aspx>.

[38] RapidShare: 1-click Web hosting – Easy Filehosting. Available from: <www.rapidshare.com/>.
[39] Megaupload: the leading online storage and file delivery service. Available from: <www.megaupload.com/>.
[40] Thomas Chen, Peter Henry, Phishing and countermeasures: understanding the increasing problem of electronic identity theft, in: Journal of Digital Forensic Practice, vol. 1, issue 2, 2006, pp. 147–149.
[41] N. Chou, R. Ledesma, Y. Teraguchi, D. Boneh, J.C. Mitchell, Client-side defense against web-based identity theft, in: 11th Annual Network and Distributed System Security Symposium, 2004.
[42] Joon S. Park, Ravi Sandhu, Secure cookies on the Web, in: IEEE Internet Computing, vol. 4, issue 4, 2000, pp 36–44.
[43] Preecha Noiumkar, Thawatchai Chomsiri, Top 10 free web-mail security test using session Hijacking, in: 3rd International Conference on Convergence and Hybrid Information Technology, 2008.
[44] F. Li, W. Wang, J. Ma, H. Su, Action-based access control for Web services, in: Proceedings of the 2009 5th International Conference on Information Assurance and Security, vol. 2, 2009, pp 637–642.