Securing Web Service by Automatic Robot Detection

KyoungSoo Park, Vivek S. Pai
Princeton University

Kang-Won Lee, Seraphin Calo
IBM T.J. Watson Research Center

Abstract

Web sites are routinely visited by automated agents known as Web robots, that perform acts ranging from the beneficial, such as indexing for search engines, to the malicious, such as searching for vulnerabilities, attempting to crack passwords, or spamming bulletin boards. Previous work to identify malicious robots has relied on ad-hoc signature matching and has been performed on a per-site basis. As Web robots evolve and diversify, these techniques have not been scaling.

We approach the problem as a special form of the Turing test and defend the system by inferring if the traffic source is human or robot. By extracting the implicit patterns of human Web browsing, we develop simple yet effective algorithms to detect human users. Our experiments with the CoDeeN content distribution network show that 95% of human users are detected within the first 57 requests, and 80% can be identified in only 20 requests, with a maximum false positive rate of 2.4%. In the time that this system has been deployed on CoDeeN, robot-related abuse complaints have dropped by a factor of 10.

1 Introduction

Internet robots (or bots) are automated agents or scripts that perform specific tasks without the continuous involvement of human operators. The enormous growth of the Web has made Internet bots indispensable tools for various tasks, such as crawling Web sites to populate search engines, or performing repetitive tasks such as checking the validity of URL links.

Unfortunately, malicious users also use robots for various tasks, including (1) harnessing hundreds or thousands of compromised machines (zombies) to flood Web sites with distributed denial of service (DDoS) attacks, (2) sending requests with forged referrer headers to automatically create "trackback" links that inflate a site's search engine rankings, (3) generating automated "click-throughs" on online ads to boost affiliate revenue, (4) harvesting e-mail addresses for future spamming, and (5) testing vulnerabilities in servers, CGI scripts, etc., to compromise machines for other uses.

In this paper, we describe our techniques for automatically identifying human-generated Web traffic and separating it from robot-generated traffic. With this information, we can implement a number of policies, such as rate-limiting robot traffic, providing differentiated services, or restricting accesses. Such identification can help protect individual Web sites, reduce the abuse experienced by open proxies, or help identify compromised computers within an organization.

Distinguishing between humans and robots based on their HTTP request streams is fundamentally difficult, and reminiscent of the Turing test. While the general problem may be intractable, we can make some observations that are generally useful. First, we observe that most Web browsers behave similarly, and these patterns can be learned, whereas the behaviors of specialized robots generally deviate from normal browsers. Second, the behavior of human users will be different from robots – for example, human users will only follow visible links, whereas crawlers may blindly follow all the links in a page. Third, most human users browse Web pages using the mouse or keyboard, whereas robots need not generate mouse or keyboard activity. From these observations, we propose two effective algorithms for distinguishing human activities from bots in real time: (1) human activity detection, and (2) standard browser detection.

In our study, we test the effectiveness of these algorithms on live data by instrumenting the CoDeeN open-proxy-based content distribution network [9]. Our experiments show that 95% of human users can be detected within the first 57 requests, 80% can be identified within 20 requests, and the maximum false positive rate is only 2.4%. Four months of running our algorithms in CoDeeN indicates that our solution is effective, and reduces robot-related abuse complaints by a factor of 10.

2 Approach

In this section, we describe how to identify human-originated activity and how to detect the patterns exhibited by Web browsers. We implement these techniques in Web proxy servers for transparency, though they could also be implemented in firewalls, servers, etc. We use the term "server" in the rest of this paper to designate where the detection is performed.

2.1 Human Activity Detection

The key insight behind this technique is that we can infer that human users are behind the Web clients (or browsers) when the server gets the evidence of mouse movement or keyboard typing from the client. We detect this activity by embedding custom JavaScript in the pages served to the client. In particular, we take the following steps:

```
<html>
...
<script language="javascript"
 src="./index_0729395150.js"></script>
<body onmousemove="return f();">
<script>
function getuseragnt()
{ var agt = navigator.userAgent.toLowerCase();
  agt = agt.replace(/ /g, "");
  return agt;
}
document.write("<link rel=\'stylesheet\'"
               + "type=\'text/css\'"
               + "href=http://www.example.com/"
               +  getuseragnt() + ">");
</script>
...
</body>
...
</html>
```

```
<!-- ./index_0729395150.js -->

var do_once = false;
function f()
{
 if (do_once == false) {
   var f_image = new Image();
   do_once = true;
   f_image.src = 'http://www.example.com/0729395160.jpg';
   return true;
 }
 return false;
}
```

Figure 1: Modified HTML and its linked JavaScript code. <script>...</script> and "onmousmove=..." is dynamically added in the left HTML. The second <script>..</script> sends the client's browser agent string information back to the server.

1. When a client requests page 'foo.html', the server generates a random key, $k \in [0, \ldots, 2^{128} - 1]$ and records the tuple <foo.html, k> in a table indexed by the client's IP address. The table holds multiple entries per IP address.

2. The server dynamically modifies 'foo.html' and delivers it to the client. It includes JavaScript that has an event handler for mouse movement or key clicks. The event handler fetches a fake embedded object whose URL contains k, such as http://example.com/foo_html_k.jpg. Suppose the correct URL is U_0. To prevent smart robots from guessing U_0 without running the script, we obfuscate the script with additional entries such that it contains $m(>0)$ similar functions that each requests U_1, ..., U_m, where U_i replaces k with some other random number $k_i(\neq k)$. Adding lexical obfuscation can further increase the difficulty in deciphering the script.

3. When the human user moves the mouse or clicks a key, the event handler is activated to fetch U_0.

4. The server finds the entry for the client IP, and checks if k in the URL matches. If so, it classifies the session as human. If the k does not match, or if no such requests are made, it is classified as a robot. Any robot that blindly fetches embedded objects will be caught with a probability of $\frac{m-1}{m}$.

Figure 1 shows an example of dynamic HTML modification at www.example.com. For clarity of presentation, the JavaScript code is not obfuscated. To prevent caching the JavaScript file at the client browser, the server marks it uncacheable by adding the response header line "Cache-Control: no-cache, no-store".

The sample code shows the mouse movement event handler installed at the <body> tag, but one can use other tags that can easily trigger the event handler, such as a transparent image map (under the <area> tag) that covers the entire display area. Other mouse related events such as "mouseup" and "mousedown" can also be used as well as keyboard events. Alternatively, one can make all the links in the page have a mouse click handler. For example, the following code

```
<A HREF=somelink.html onclick=''return f();''>
Follow me</A>
```

will call the function f() when a human user clicks the "Follow me" link. The function f() contains

```
f_image.src
    ='http://www.example.com/0729395160.jpg';
```

which has the side effect of fetching the image, so the server will receive the mouse movement evidence with k = 0729395160 in the URL. The server can respond with any JPEG image because the picture is not used. The parameter k in the URL prevents replay attacks, so the server should choose k at random for each client/page.

The script below the <body> tag in Figure 1 sends the client's browser string to the server. This embedded JavaScript tells the server whether the client enables JavaScript or not. If the client executes the code, but does not generate the mouse event, the server can infer that it is a robot capable of running the JavaScript code.

2.2 Browser Testing

In practice, a small fraction of users ($4 - 6\%$ in our study) disable JavaScript on their browsers for security or other reasons. To avoid penalizing such users, we employ browser detection techniques based on the browsing patterns of common Web browsers. The basic idea behind this scheme is that if the client's behavioral pattern deviates from that of a typical browser such as IE, Firefox, Mozilla, Safari, Netscape, Opera, etc., we assume

it comes from a robot. This can be considered a simplified version of earlier robot detection techniques [6], and allows us to make decisions on-line at data request rates. Basically, we collect request information within a session and try to determine whether a given request has come from a standard browser. This provides an effective measure without overburdening the server with excessive memory consumption. An obvious candidate for browser detection, the "User-Agent" HTTP request header, is easily forged, and we find that it is commonly forged in practice. As a result, we ignore this field.

On the other hand, we discover that many specialized robots do not download certain embedded objects in the page. Some Web crawlers request only HTML files, as do email address collectors. Referrer spammers and click fraud generators do not even need to care about the content of the requested pages. Of course, there are some exceptions like off-line browsers that download all the possible files for future display, but the goal-oriented robots in general do not download presentation-related information (e.g., cascading style sheet (CSS) files, embedded images), or JavaScript files because they do not need to display the page or execute the code.

We can use this information to dynamically modify objects in pages and track their usage. For example, we can dynamically embed an empty CSS file for each HTML page and observe if the CSS file gets requested.

```
<LINK REL="stylesheet" TYPE="text/css"
 HREF="http://www.example.com/2031464296.css">
```

Since CSS files are only used when rendering pages, this technique can catch many special-purpose robots that ignore presentation-related information. We can also use silent audio files or 1-pixel transparent images for the same purpose. Another related but inverse technique is to place a hidden link in the HTML file that is not visible to human users, and see if the link is fetched.

```
<A HREF="http://www.example.com/hidden.html">
<IMG SRC="http://www.example.com/transp_1x1.jpg">
</A>
```

Because the link is placed on a transparent image which is invisible to human users, humans should not fetch it. However, some crawlers blindly follow all the links, including the invisible ones.

3 Experimental Results

To evaluate the effectiveness of our techniques in a real environment, we have implemented them in the CoDeeN content distribution network [9]. CoDeeN consists of 400+ PlanetLab nodes and handles 20+ million requests per day from around the world. Because CoDeeN nodes look like open proxies, they unintentionally attract many robots that seek to abuse the network using the anonymity provided by the infrastructure. While our previous work on rate limiting and privilege separation [9] prevented much abuse, we had to resort to manual pattern detection and rule generation as robots grew more advanced.

Description	# of Sessions	Percentage(%)
Downloaded CSS	268,952	28.9
Executed JavaScript	251,706	27.1
Mouse movement detected	207,368	22.3
Passed CAPTCHA test	84,924	9.1
Followed hidden links	9,323	1.0
Browser type mismatch	6,288	0.7
Total sessions	929,922	100.0

Table 1: CoDeeN Sessions between 1/6/06 and 1/13/06

3.1 Results from CoDeeN Experiments

We instrumented the CoDeeN proxies with our mechanisms and collected data during a one-week period (Jan 6 - 13, 2006), with some metrics shown in Table 1. For this analysis, we define a session to be a stream of HTTP requests and responses associated with a unique <IP, User-Agent> pair, that has not been idle for more than an hour. To reduce the noise, we only consider sessions that have sent more than 10 requests.

Of the 929,922 sessions total, 28.9% retrieved the empty CSS files we embedded, indicating that they may have used standard browsers. On the other hand, we have detected mouse movements in 22.3% of the total sessions, indicating that they must have human users behind the IP address. Considering that some users may have disabled JavaScript on their browsers, this 22.3% effectively is a lower bound for human sessions.

We can gain further insight by examining the sessions that have executed the embedded JavaScript, but have not shown any mouse movement – these definitely belong to robots. We can calculate the human session set S_H by:

$$S_H = (S_{CSS} \cup S_{MM}) - (S_{JS} - S_{MM})$$

where S_{CSS} are sessions that downloaded the CSS file, S_{MM} are sessions with mouse movement, and S_{JS} are sessions that executed the embedded JavaScript. We consider the sessions with CSS downloads and mouse movements to belong to human users except the ones that have executed JavaScript without reporting mouse movement. We label all other sessions as belonging to robots. Using the data collected from CoDeeN, we calculate that 225,220 sessions (24.2% of total sessions) belong to S_H.

Note that the above equation gives us an upper bound on the human session set because this set has been obtained by removing from the possible human sessions any that clearly belong to robot sessions. However, the difference between the lower bound (22.3%) and the upper bound (24.2%) is relatively tight, with the maximum false positive rate(# of false positives/# of negatives) = 1.9%/77.7% = 2.4%.

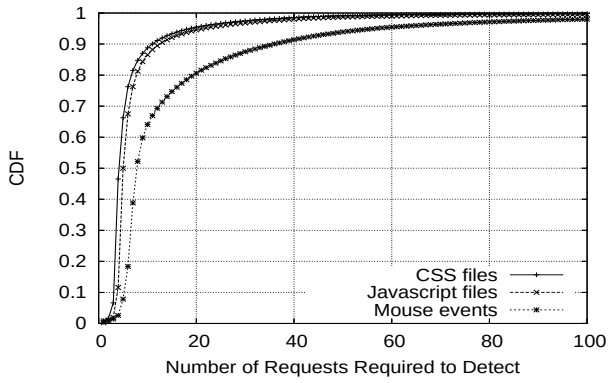

Figure 2: CDF of # of Requests Needed to Detect Humans

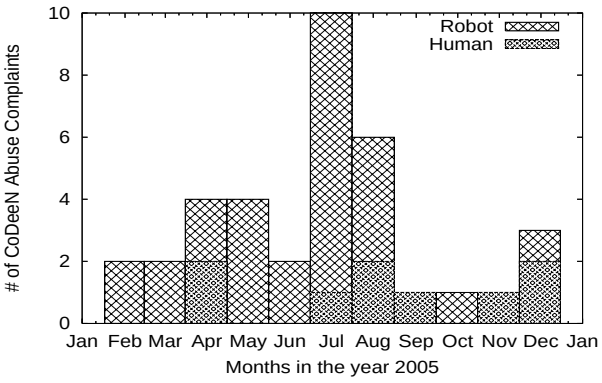

Figure 3: # of CoDeeN Complaints Excluding False Alarms

Our mouse detection scheme relies on the widespread use of JavaScript among the users. To understand how many users have disabled JavaScript on their browsers, we employed a CAPTCHA test [8, 1] during the data collection period. Users were given the option of solving a CAPTCHA with an incentive of getting higher bandwidth. We see that 9.1% of the total sessions passed the CAPTCHA, and we consider these human users.[1] Of these sessions, 95.8% executed JavaScript, and 99.2% retrieved the CSS file. The difference (3.4%) are users who have disabled JavaScript in their browsers, which is much lower than previously reported numbers ($\sim$10%). This can explain the gap between the lower bound and the upper bound of human sessions in our experiment. We also note that most standard browsers request CSS files, suggesting that our algorithm based on CSS file downloads is a good indicator for fast robot detection.

Figure 2 shows how many requests are needed for our schemes to classify a session as human or robot. 80% of the mouse event generating clients could be detected within 20 requests, and 95% of them could be detected within 57 requests. Of clients that downloaded the embedded CSS file, 95% could be classified within 19 requests and 99% in 48 requests. The clients who downloaded JavaScript files show similar characteristics to the CSS file case. Thus, the standard browser testing is a quick method to get results, while human activity detection will provide more accurate results provided a reasonable amount of data. We revisit this issue in Section 4 when we discuss possible machine learning techniques.

3.2 Experience with CoDeeN's Operation

During the four months this scheme has been deployed in CoDeeN, we observed that the number of complaints filed against CoDeeN has decreased significantly. Figure 3 presents the number of complaints filed against the CoDeeN project in 2005. In February, we expanded the deployment of CoDeeN on PlanetLab, from 100 US-only nodes to over 300 total nodes worldwide. As CoDeeN became widely used, the number of complaints started to rise (peaking in July, when most of the complaints were related to referrer spam and click fraud). In late August, we deployed the standard browser test scheme on CoDeeN, and enforced aggressive rate limiting on the robot traffic. After we classify a session to belong to a robot, we further analyzed its behavior (by checking CGI request rate, GET request rate, error response codes, etc.), and blocked its traffic as soon as its behavior deviated from predefined thresholds. After installing this mechanism, we observed the number of complaints related to robot activities have decreased dramatically, to only two instances over four months. During this period, the other complaints were related to hackers, who tried to exploit new PHP or SQL vulnerabilities through CoDeeN. The mouse movement detection mechanism was deployed in January 2006, and we have not received any complaints related to robots as of April 17^{th}.

We also investigate how much additional overhead these schemes impose, and we find it quite acceptable. A fake JavaScript code of size 1KB with simple obfuscation is generated in 144 μseconds on a machine with a 2 GHz Pentium 4 processor, which would contribute to little additional delay in response. The bandwidth overhead of fake JavaScript and CSS files comprise only 0.3% of CoDeeN's total bandwidth.

4 Discussions and Future Work

In this section, we discuss limitations of our current system and possible improvements using machine learning.

4.1 Limitations of Our Approach

Our proposed detection mechanism is not completely immune to possible countermeasures by the attackers. A serious hacker could implement a bot that could generate

[1] While some CAPTCHA tests can be solved by character recognition, this one was optional, and active only for a short period. We saw no abuse from clients passing the CAPTCHA test, strongly suggesting they were human.

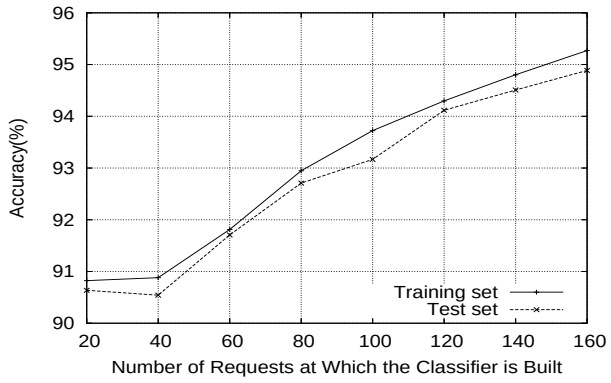

Figure 4: Machine Learning Performance to Detect Robots

Attribute	Explanation
HEAD %	% of HEAD commands
HTML%	% of HTML requests
IMAGE %	% of Image(content type=image/*)
CGI %	% of CGI requests
REFERRER %	% of requests with referrer
UNSEEN REFERRER %	% of requests with unvisited referrer
EMBEDDED OBJ %	% of embedded object requests
LINK FOLLOWING %	% of link requests
RESPCODE 2XX %	% of response code 2XX
RESPCODE 3XX %	% of response code 3XX
RESPCODE 4XX %	% of response code 4XX
FAVICON %	% of favicon.ico requests

Table 2: 12 Attributes used in AdaBoost

mouse or keystroke events if he or she knows that a human activity detection mechanism has been implemented by a site. Although we are not aware of any such intelligent bots today, we may need to address such issues in the future. One possible way to address the problem is to utilize a trusted underlying computer architecture to guarantee that the events have been truly generated by the physical devices [7]. A more practical solution may combine multiple approaches in a staged manner – making quick decisions by fast analysis (e.g., standard browser test), then perform a careful decision algorithm for boundary cases (e.g., AI-based techniques). Our goal in this paper was to design a fast and effective robot detection algorithm that could be deployed in CoDeeN to effect practical benefits, which we seem to have achieved. However, we do not feel the work is complete; on the contrary, it has just started.

4.2 Detection by Machine Learning

From the above discussion, the following question naturally follows: "How effective is a machine learning-based technique, and what is the trade-off?" The main forte of machine learning is that if we can characterize the typical features of human browsing, we can easily detect unwanted traffic by robots. Conceivably, it is very hard to make a bot that behaves exactly like a human.

To test the effectiveness of a detection algorithm using machine learning, we collected data by running CAPTCHA tests on CoDeeN for two weeks, and classified 42,975 human sessions and 124,271 robot sessions using 12 attributes shown in Table 2. We then divided each set into a training set and a test set, using equal numbers of sessions drawn at random. We built eight classifiers at multiples of 20 requests, using the training set. For example, the classifier at the request number 20 means that the classifier is built calculating the attributes of the first 20 requests, and the 40-request classifier uses the first 40 requests, etc. We used AdaBoost [5] with 200 rounds.

Figure 4 shows the accuracy of classification with respect to the number of requests. The result shows the classification accuracy ranges from 91% to 95% with the test set, and it improves as the classifier sees more requests. From our experiment, RESPCODE 3XX%, REFERRER % and UNSEEN REFERRER % turned out to be the most contributing attributes. Basically, robots do not frequently make requests that result in redirection; many bot requests do not have a valid referrer header field; and finally, referrer spam bots frequently trip the unseen referrer trigger.

Although this approach is promising, it has a few drawbacks. First, it requires significant amount of computation and memory, which may make the server susceptible to DoS attacks. Second, in order to make accurate decisions, it needs a relatively large number of requests, making it difficult to apply in a real-time scenario (in our experiment, it takes 160 requests to achieve 95% accuracy). Third, the human browsing pattern may change with the introduction of a new browser with novel features. Finally, attribute selection must be done carefully. In theory, the learning algorithm can automatically determine the most effective attributes, but in practice, bad choices can decrease the effectiveness of the classifier.

5 Related Work

Despite the potential significance of the problem, there has been relative little research in this area, and much of it is not suitable for detecting the robots in disguise. For example, Web robots are supposed to adhere to the robot exclusion protocol [4], which specifies easily-identified User-Agent fields, with contact information. Before crawling a site, robots should also retrieve a file called "robots.txt", which contains the access permission of the site defined by the site administrator. Unfortunately, this protocol is entirely advisory, and malicious robots have no incentive to follow it.

Tan *et al.* investigated the navigational pattern of Web robots and applied a machine learning technique to exclude robot traces from the Web access log of a Web site [6]. They note that the navigational pattern of the Web crawlers (e.g., type of pages requested, length of a session, etc.) is different from that of human users, and

these patterns can be used to construct the features to be used by a machine learning algorithm. However, their solution is not adequate for real-time traffic analysis since it requires a relatively large number of requests for accurate detection. Robertson *et al.* tried to reduce administrators' effort in handling the false positives from learning-based anomaly detection by proposing the generalization (deriving anomaly signatures to group similar anomalies) and characterization (to give concrete explanation on the type of the attacks) techniques [3]. Using generalization, one can group similar anomalies together, and can quickly dismiss the whole group in the future if the group belongs to false positives. In contrast to these approaches, our techniques are much simpler to implement yet effective in producing accurate results for incoming requests at real time. Moreover, our proposed solution is robust since it does not have any dependency on specific traffic models or behavior characterizations, which may need to change with the introduction of more sophisticated robots.

CAPTCHA (**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part) is a test consisting of distorted images or sounds, sometimes with instructive description, that are designed to be difficult for robots to decipher [8]. These tests are frequently used by commercial sites which allow only human entrance or limit the number of accesses (e.g. concert ticket purchasing). Kandula *et al.*, used CAPTCHA tests to defend DDoS attacks by compromised robots that mimic the behavior of flash crowds [2]. They optimize the test serving strategy to produce better goodput during the attack. Although CAPTCHA tests are generally regarded as a highly effective mechanism to block robots, it is impractical in our scenario, since human users do not want to solve quiz every time they access a Web page. In comparison, our techniques do not need explicit human interaction, and can be used on every page, while producing highly accurate results. Also we are more concerned in providing a better service under a normal operation rather than special situations such as during denial of service attacks.

6 Conclusion

While Web robots are an important and indispensable part of the modern Web, the subset of malicious robots poses a significant and growing concern for Web sites, proxy networks, and even large organizations. We believe the first step to deal with this is to accurately classify the traffic source, and we present two novel techniques for discriminating humans from robots.

Our experience with CoDeeN shows that our solution is highly effective in identifying humans and robots. 95% of humans are detected within 57 requests with less than a 2.4% false positive rate. The integration of the techniques in CoDeeN's operation also greatly reduced the number of abuse complaints caused by robots. Furthermore, we believe that our solution is quite general – not only does it apply to the safe deployment of open proxies, but it can be used to identify streams of robot traffic in a wide variety of settings. We believe that this approach can be applied both to individual Web sites, and to large organizations trying to identify compromised machines operating inside their networks.

Acknowledgments

We thank Felix Holderied and Sebastian Wilhelmi [1] for providing the CAPTCHA image library, and Peter Kwan for useful discussion and feedback. This work was supported in part by NSF Grants ANI-0335214, CNS-0439842, and CNS-0520053.

References

[1] captchas.net. http://www.captchas.net/.

[2] S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation(NSDI '05)*, 2005.

[3] W. Robertson, G. Vigna, C. Krugel, and R. A. Kemmerer. Using generalization and characterization techniques in the anomaly-based detection of web attacks. In *Proceedings of The 13th Annual Network and Distributed System Security Symposium(NDSS '06)*, 2006.

[4] Robot Exclusion Protocol. http://www.robotstxt.org/wc/exclusion.html.

[5] R. Schapire. The boosting approach to machine learning: An overview. *Nonlinear Estimation and Classification*, 2003.

[6] P.-N. Tan and V. Kumar. Discovery of web robot sessions based on their navigational patterns. *Data Mining and Knowledge Discovery*, 6:9–35, 2002.

[7] Trusted Computing Group. http://www.trustedcomputinggroup.org/.

[8] L. von Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In *Proceedings of Eurocrypt*, pages 294–311, 2003.

[9] L. Wang, K. Park, R. Pang, V. Pai, and L. Peterson. Reliability and security in the CoDeeN content distribution network. In *Proceedings of the USENIX Annual Technical Conference*, 2004.