A Link Obfuscation Service to Detect Webbots

Douglas Brewer, Kang Li, Laksmish Ramaswamy
Deparment of Computer Science
University of Georgia
415 Boyd GSRC
Athens, GA 30602
brewer,kangli,laks@cs.uga.edu

Calton Pu
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332
calton@cc.gatech.edu

Abstract

Web bot fraud activity currently accounts for a large number of web accesses. Current resistance methods such as CAPTCHA are not applicable for bot detection at the granularity of each click. In this paper, we propose a service that counters web bots which mimic human clicks by walking random links. We base our defense on systematically applying link obfuscation. The obfuscation is designed as a service that can be applied to websites without changes from web developers and without changing the behavior of human users.

The service for resisting web bots is called Decoy Link Design Adaptation (DLDA) and works by transparently modifying every page of a protected website. The modifications are made such that walking web bots cannot traverse valid paths through the website. Specifically, DLDA modifies each original link on the page surrounding it with a group of invalid links. These obsfucated links are carefully styled to be unnoticed or avoided by human users; however, they require significant effort for programs (bots) to identify.

Experiments show that DLDA has a very high detection rate for web bots and near zero false positives. DLDA can detect 80% of walking bots ending a session after one minute of inactivity (no clicks). The detection rate increases to 100% when the session is ended where multiple visits of the bots can be grouped into a single session.

1. Introduction

Today, a significant number of web accesses are from automated programs (web bots) rather than human beings. Some of these bot accesses are for legitimate and essential purposes (e.g., search engine indexing) and would be welcome by the target web sites. But, a large and growing fraction of bot accesses are for fraudulent activities such as click fraud [12, 18, 10] and email address harvesting by spammers [15].

This paper presents a service for protecting websites from malicous bots based on link obfuscation. Obfuscation, although not considered a general security solution, has been widely used to resist web bot activities. For example, many users post an obfuscated version of their email address (such as spelling out AT and DOT) to prevent spammers' web bots from harvesting email addresses in original formats, and fake links are put up on websites to trap malicious web bots [11]. Although widely used, these obfuscation efforts often require web developers to manually edit web pages and insert the content. Furthermore, these obfuscation techniques sometimes introduce burdens on human users visiting the website.

Our approach is based on the observation that most web bots do not have the same capabilities as web browsers. Modern browsers are commonly equipped with plugins, scripting engines (e.g. a Javascript interpreter), and complex rendering components to support Javascript-Cascading Style Sheet (CSS) interaction [1]. Although, in theory, web bots could also equip themselves with these components, in practice, they are rarely built with all these features and interaction capabilities.

Our work specifically focuses on countering *Random walking bots* – bots that mimic human browsing by retrieving random links from a web page [1]. Our technique for detecting web bot activity, called *Decoy Link Design Adaptation (DLDA)*, builds on previous methods for detection with decoy information using links[6] and email addresses[4]. DLDA adds artificial diversity to a web page by obfuscating real links on a web page surrounding them with invalid links. Each systematically obfuscated link presents a challenge to the web robot. However, the solution to the challenge is obvious to a human user when the web page is

[1]There are other types of web bots, such as reply bots which repeat valid accesses from pre-recorded human accesses. We address these bots in another work [7]

viewed through a web browser.

We have implemented the DLDA service as an Apache Web Server module. The module operates in realtime modifying the HTML of a web page as it is served and checking the requests as they are received. The current implementation takes advantage of the Javascript, CSS, and HTML integration in a web browser making the website modifications transparent to website developers as well as its users.

We experimentally evaluate web bot detection with DLDA focusing on the accuracy in detecing web bots, the false positives for humans, and the overhead introduced when serving a web page. The session based analysis of the modified access logs shows that we are able to detect 80%-100% of walking web robots. The session inactivity timeout plays an important roles in detection rates; increasing the timeout can group many logical visits by the bot into a single session. DLDA also shows increases in bot detection by either increasing the number of decoy links or with bots visting more web pages. We importantly note that no humans were detected as web robots by our implementation.

2. Prior Work on Web Bot Resistance

Past efforts towards resisting web bots have proven partially successful by introducing interactions that are only handled well by humans. CAPTCHA [17] has successfully resisted web bots that automatically register accounts and post messages on websites. However, the level of human interaction required practically limits CAPTCHA's frequency of use. Applying CAPTCHA to every page and every link would cause users extreme frustration. Thus, CAPTCHA is not appropriate for attacks such as click fraud where it is strongly preferred to check for a human presence every time.

Many unwanted web bot activities are defined in terms of fraudulent clicks, and early approaches such as duplicate click detection [12, 18] have attempted to detect fraud at the granularity of individual clicks. Duplicate click detection searches for duplicate entries in click streams or web access logs and discards them as fraud. Detecting fraud based on a single line in the log is often challenging due to the limited information: typically, only IP address, URL, and some user agent information are available.

Bot behavior is often more easily identified when looking at a group of requests (clicks) called a session. Sessions allow for the discovery of more information about a user's behavior on a website. Interesting details like navigation patterns and durations between requests can be found in a session. Therefore, we build our detection methods to work at granularity of sessions rather than individual clicks.

3. The DLDA Obfuscation Service

Our strategy for providing an obfuscation service to detect web bots is to embed special *decoy links* in web pages. Although these links exist in the web page, they are easily identified and avoided by a human viewing the web page in a web browser. In the best case, the decoy links can be rendered invisible in a browser. Thus, a request to such a decoy link indicates that the requestor is not interacting with the website through a web browser which in turn strongly suggests that the requestor is a bot. This strategy is referred to as *Decoy Link Design Adaptation(DLDA, for short)*

A website owner adopting DLDA will rightly have concerns about its effects on the operation of the website as well as its interaction with outside agents (including users and third-party services). Here we address how the DLDA service will address the following requirements:

- **Human User Transparency** - The use of DLDA should not affect user navigation or behavior.
- **Developer Transparency** - The use of DLDA should not cause a burden to the developer.
- **Bot Detection** - DLDA should detect web robots with a high probability.

4. Human User Transparency

There are many potential link obfuscation principles that can be used to implement DLDA. It is very important to consider how a user will view the obfuscated links on a page, so that a user viewing the page in a browser will avoid invalid links. Just adding links to a web page without any consideration of link placement and style will break user navigation and cause them to click invalid links.

To help human users avoid invalid links, DLDA takes care to make valid links on a page into visually distinct elements. An implementation of DLDA can be based on one of many link design principles that allow for a distinct valid link. We examined four different adaptations of link principles, listed below, to see which one was least likely to confuse human users. They were each put on a *landing page* – a web page consisting of only the links that lead either to the next website or a page saying "You are a bot." We tested this with approximately twenty human users over four days.

- **Multiple Link** — The Multiple Links web design principle positions multiple decoy links off the visible page and valid links remain in their original, visible position. The decoy links contain text mimicing that of the valid links.
- **Tag Cloud** — The Tag Cloud web design principle creates a Tag Cloud of decoy links each with identical text where the largest link is the valid link.

- **Tab Link** — The Tabbed Menu web design principle creates a tabbed menu where decoy links appear as tabs containing identical text and color, and the valid link is a tab with a different color.

- **Shadow Link** — The Shadow Link web design principle creats multiple, invisible decoy links containing the text of the valid link. These decoy links, however, are positioned with the same x, y coordinates as the valid link, but below the valid link itself. Effectively stacking the valid link on top of the invalid links.

Our results showed that the Tag Cloud and the Tab Links techniques cause to much confusion. The invalid links were clicked with a frequency of 24% and 48%, respectively. However, the Multiple Link and Shadow Link techniques each avoided invalid clicks by human users.

4.1. Developer Transparency

The modification of links required by DLDA can be done by the developer creating a website or by a program that modifies the website automatically. Making the developer do the work to enable DLDA on a website can significantly increase development time and costs. Thus, an automatic implementation of DLDA makes the most sense.

Using a program to modify links limits the available options for DLDA discussed in the previous section. We cannot use Tag Clouds or Tab Links to implement DLDA because they would ruin the look and feel of the website. Therefore an implementation based around either Multiple Links or Shadow Links is required and with the correct styling is unnoticeable on the site look and feel.

To achieve the goal of decoying valid links, the DLDA service searches for anchor and head tags. Whenever DLDA finds an anchor tag that is the child of the body element, it is replaced by a group of valid and invalid links. Each output link is an exact replica, excepting the URL, of the input link containing all attributes and their values and exact copies of text between the open and close tags.

The module must also style the output groups of links to comply with either the Multiple Link or Shadow Link technique. This involves outputting CSS before the close of the head section of an HTML document. The Multiple Link technique sets “position:absolute”, “left:-1000px”, and “top:-1000px” to position each invalid link off the screen. Shadow links gives a surrounding span tag a “position: relative” declaration and links get “position: absolute”, “left: 0px”, and “top: 0px” declarations. This causes all the links to overlap within their enclosing span tag. The way a browser renders requires the valid link be last link within the span tag. Figure 1 shows the stacking of links in the HTML and that when style is applied both the protected and unprotected page have the same look and feel.

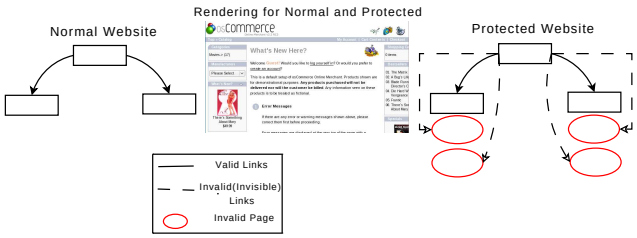


Figure 1. Implemented DLDA modification of a web site. The rendering of the page is the same for both the modified and unmodified web page.

4.2. Bot Detection

The implementation of DLDA with either Multiple Links or Shadow Links leaves exploitable clues for bot writers. Multiple Links only styles the links to be rendered off page, and Shadow links requires the valid link be the last link within the span tag. Avoiding either pitfall, requires outputting the valid link in a random position within the group and the use of Javascript for positioning the links or setting the stacking order for Multiple Links and Shadow Links, respectively.

Making the HTML and Javascript modifications necessary for Multiple Links and Shadow Links to detect bots makes them very similar. However, there is a difference in the complexity of the Javascript necessary to perform the modifications. So, for our prototype, we decided the simpler Javascript implementation for Shadow Links was easier to use and obfuscate.

Obfuscating the links starts by choosing a random position for the valid link within the group of links. This is accomplished using a random number generator limiting its output to between 1 and N where N is the maximum number of links in a group, including the valid link. With the location of the valid links decided, Shadow Links will output a slightly modified form of obfuscated Javascript. The Javascript changes the stacking order of the link group placing the valid links on top of all the invalid ones using the “z-index” attribute. The z-index affects which HTML element is drawn on top of the others when more than one element occupies the same position on the page.

The invalid and valid links are now hard to tell apart, and this leaves the bot likely to click an invalid link. When an invalid link is clicked, DLDA will register this as a potential bot.

5. Implementation

We had three choices for the programmatic implementation of DLDA an offline program, a proxy, or a web server module. The offline program was never an option due to the many different programming languages a website could use for implementation. This left us a choice between the proxy and the web server module.

Our implementation of DLDA allowed a configurable set of invalid decoy links. The set of invalid decoy links should be made to resemble the valid links of the website being protected. This makes it harder for bot writers to distinguish the valid and invalid links within the HTML. DLDA checks if a requested URL is in the configured list of invalid decoy links making a note in the access log if it is.

Finally, the access log is analyzed offline with DLDA creating sessions of clicks. The detection of an invalid link within a session marks the session as being created by a bot. Creating sessions is done by going through the access log one line at time and grouping lines based on IP. A session was terminated or ended based on an period of inactivity called the *session inactivity timeout*. If since the last access from an IP the period of time specified by the session inactivity timeout had elapsed before another access from the same IP, the current session for that IP was ended.

We used sessions for two reasons in DLDA: infected computers and real-time analysis. It is possible that a user is browsing the website on an infected computer that has a bot visiting the same site. It is important to avoid an overlap of the two accesses in this case and sessions provide a measure of safety. Analyzing sessions also shows DLDA's potential for use in real-time analysis; we can see DLDA's accuracy with limited information.

The actual implemenation of DLDA was done as an Apache Web Server module. This provided an easier implementation route than writing a full proxy. It also allowed the use of the "Directory" and "Location" configuration options. These allow us to choose which parts of a website incur overhead from DLDA unlike a proxy implementation where every request made incurs at least some overhead.

6. Experiments

This section presents our evaluation of the web bot detection through experiments. For the purpose of this evaluation, we built a test environment that includes a protected web site. In our setup, we used a default installation of osCommerce Online Merchant as the protected site [5]. osCommerce is one of the most popular open source implementations of an e-commerce web site.

To study the effectiveness of bot detection, we built three different web bots based on bots observed on the Internet. These three are walking type bots with different configurations (Random bot, Link Keyword bot, and CSSJS bot). The Random Walk bot chose links at random from all links on the page, the Link Keyword Word bot chose links based on keywords contained within the anchor text, and the CSSJS bot [2] chose links based on their ability to be seen by the user. The Keyword and CSSJS bot both chose links that fit their criteria at random. We believe these represent typical available strategies available to bot authors. Further, the Random Walk bot uses the same link choosing mechanism as ClickBot.A[9] which chooses a random link from a search results page.

To help mimic realistic traffic to a web site, we also mixed the bot traffic with accesses from human volunteers. We had ten human users visit our website over the course of two days. They were given the instructions to visit the website and click links and report any unusual behavior.

With this setup, we evaluate the effectiveness of the bot detection methods with the following aspects. First, we compare the overall web bot detection accuracy with a well known bot detection method used in industry. We also look at the overhead introduced by the DLDA Apache module, both in terms of the increased size of HTML and delay introduce to users. At the end, we study the impact of the number of decoy links and session inactivity timeout value, on the detection accuracy.

In all the experiments, our setup was an Apache Web Server with the DLDA module modifying only the content from the Online Merchant store. We varied the number of links output by DLDA from between two and fifty output links to gauge the overhead and effect on detection.

6.1. Cross Comparison of Bot Detection Accuracy

We compared DLDA to a technique for detecting web robots based on statistical, passive analysis of sessions by Bomhardt et al.[6]. The statistical, passive analysis looked for common web robot user agents, a near zero average request time, and referrer header. Both techniques were run over the same web server logs of all robot sessions, and we compared how well they detected web robots while varying session times.

We can see from the comparison in Figure 2 that the active approach taken by DLDA is more likely to detect a web robot. The method presented by Bomhardt et al. while useful for detecting well behaved web bots seems to fall short when detecting adversarial varieties. Everything checked for in this passive analysis can be faked by a web robot. Therefore, in the end, the web robot will look like a human.

[2] The CSSJS bot while having both CSS and Javascript capabilities did not provide for interaction between the two.

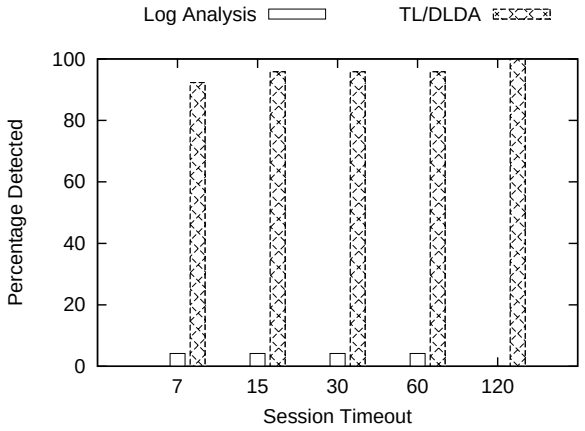


Figure 2. Comparison of DLDA and Transient Links to Log Analysis of Bomhardt et al.

6.2. Module Overhead

We studied the overhead of bot detection methods by measuring the average web page sizes and delay experienced by users. We used Apache JMeter to test the protected website [3]. JMeter was originally developed to test Apache's heavily used Tomcat Application Server and is well suited to testing dynamically generated content. JMeter was setup to request three pages 50 times each. It made 3 parallel requests to the web server and recorded the latency for each request. The three requested pages were those that had the most dynamic content and read from the backend database on each request: the main index page and two different product

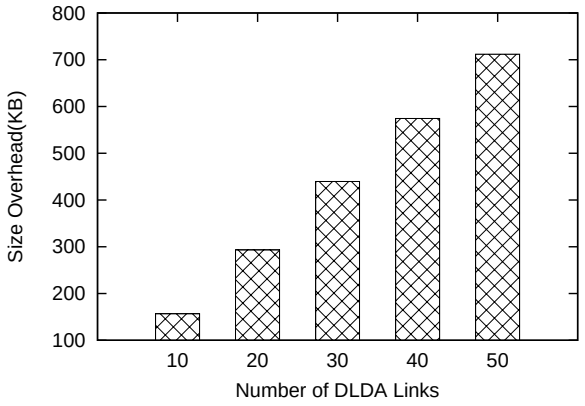


Figure 3. Overhead in bytes added by a different numbers of DLDA links.

DLDA adds data to the page when it modifies the links making the page larger. Figure 3 shows how many kilobytes are added to a page when a differing number of DLDA links is output. We see the overhead is quite large adding 130KB from the beginning. This seemingly obscene overhead comes from the large number of original links on the web page. For example, the Online Merchant outputs 37 links accounting for 15% of the HTML size while another merchant site, Amazon[2], outputs 182 links accounting for 20% of it's HTML size. This means the links output by our test site contain a relatively large amount of data. We expect normal websites with less data in their links to show significantly less overhead. This does, however, show that there is tradeoff for DLDA, and one should use the least number of links possible to detect web bots.

We measured the delay introduced by the DLDA module with various network connections, including accesses from our campus network, and wide-area accesses emulated through proxies. Had we chosen to use an offline program there would be no delay introduced, but DLDA can modify a website produced with any type of programming language when modifying the HTML online. We obtained round trip times(RTT) below 20ms from computers located within the campus network. Round trip latencies above 20ms were obtained through various transparent proxies; using any other type of proxy would interfere with testing DLDA alone.

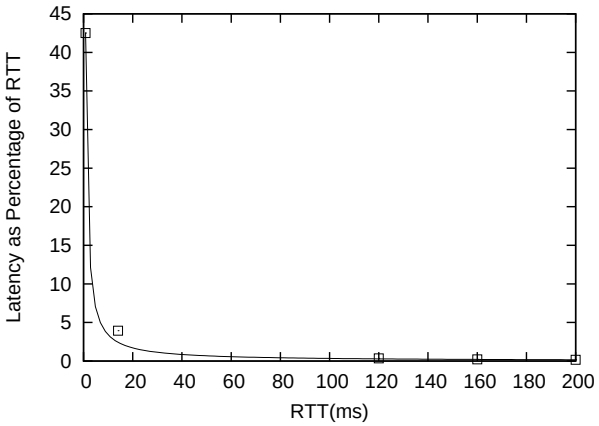


Figure 4. Overhead of modules as percentage of round trip time.

The latency overhead was fairly constant causing the latency to be become less noticeable as the RTT increased. Figure 4 shows that the added latency had an inverse correlation to the RTT time. It also shows that users that are close by (thus with small RTTs) to the a protected server are more likely see a slow down, but with the low RTTs, the user will be less likely to notice it.

6.3. Effect of System Parameters

DLDA has several parameters that potentially affect the accuracy of bot detection. We study the effect of two key parameters: session inactivity timeout values and the number of DLDA links. In addition, bot behavior such as number of pages visited per session also affect the detection accuracy. This subsection presents our study on the impact of these parameters.

Decoy Link Design Adaptation detects walking web robots as they navigate a website by outputting invalid decoy links on each page. The walking robots pick invalid links as they they walk a website. This helps DLDA mark the access as belonging to a potential web robot. Sessions created from logs with DLDA marks allow a human/bot determination based on the number of invalid links per session. For each number of links, the walking bots were run 50 times each over the Online Merchant website making 3 requests and sleeping one minute between each run. This gave us a total of 150 runs per each output number of links.

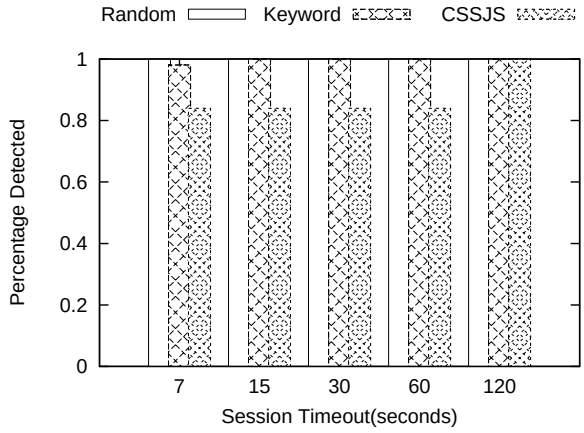


Figure 5. Detection rate of DLDA as the maximum time allowed between requests to included in the same session increases.

Figure 5 shows the detection rate of DLDA as the session inactivity timeout varied. The session inactivity timeout tells DLDA the maximum time allowed between requests such that a new session is created when the timeout is reached or exceeded. We can see from the beginning that DLDA has at worst and 80% detection rate with even a very small session inactivity timeout. The detection rate does increase, but only slightly until the 120 second session inactivity timeout is reached.

At a 120 second session inactivity timeout, DLDA is able to detect 100% of the walking web bots. This disjoint increase in detection rate is due to a "sweet spot" for grouping clicks into sessions. Our bots were set to sleep 60 seconds between each run they made against the protected Online Merchant website. Thus, once the session inactivity timeout increases beyond their sleep time we can group large numbers of runs together in a single session. This means it is better to use as large a session inactivity timeout as possible.

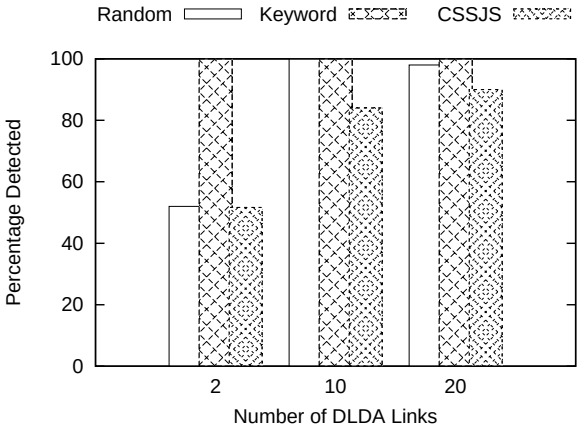
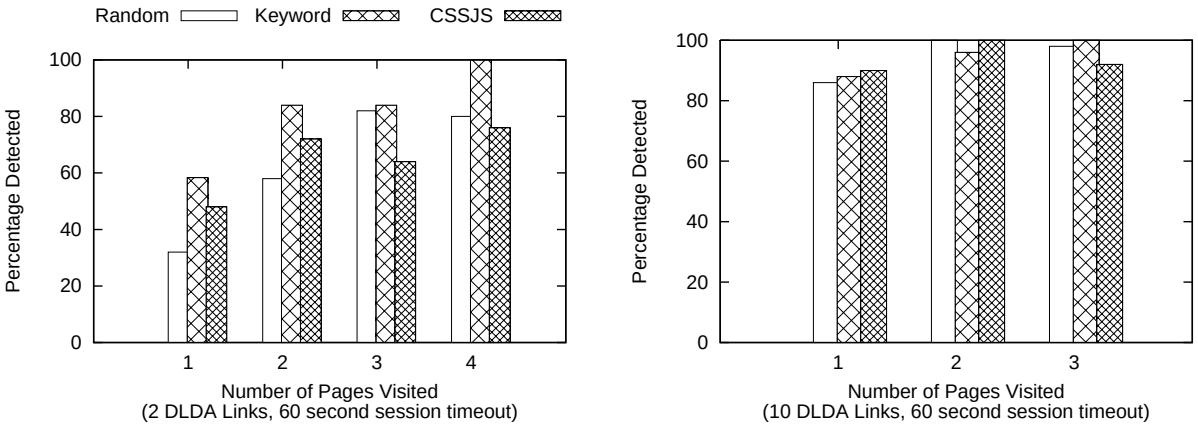


Figure 6. Detection rate of DLDA as the number of links output is increases. 2 is the absolute minimum number of links. More links than 20 did not sigificantly increase detection.

The number of links output by DLDA was varied to see the affect on web bot detection. Figure 6 shows a general trend of increasing detection with an increasing number of links. This is an intuitive result because the more invalid links on a page the more likely a bot will pick one. We see there is a dramatic increase in detection rate from two output links to 10 output links with only a slight increase at 20 links and beyond. The size overhead added by DLDA seen in Figure 3 means that one must make a tradeoff about the number of links to use. Given the size and detection results, 10 links seems to give the best overhead/detection tradeoff.

6.4. Impact of Bot Behaviors

DLDA detection was tested varying the number of protected pages a web robot would visit. We tested detection as the robots navigated different numbers of protected pages with two and 10 DLDA links. We can see in Figure 7 that a bot visiting more pages in a protected website is more likely to be detected. We can also see the immediate difference between detection with a smaller versus larger number of links in Figure 7(a) and Figure 7(b). The difference is very dramatic with an immediate 20-50% difference in detection in clicking a link from one protected page. This



(a) Two Output DLDA Links (one invalid, one valid)

(b) Ten Output DLDA Links (9 invalid, one valid)

Figure 7. Increasing detection of web robots as they navigate an increasing number of protected pages with 2 and 10 output links.

poses the overhead/detection tradeoff. Ten links will easily detect most crawling bots. This is good because it our previous result that it 10 links gives the best overhead/detection tradeoff.

We can see variations in the detection ability of DLDA in Figure 7. These are due to the random nature with which the bots selected links that met their selection criteria. As an example, both valid and invalid links met the CSSJS bot's criteria for links, and it selected a valid or invalid link at random from these. We can also see the various link selection criteria didn't help much, and we didn't expect that they would given that valid and invalid links were the same in every respect except the URL.

Human testing was conducted with our implementation of DLDA. All of the human users were able to navigate a website protected with DLDA without tripping the protection. This means it is likely safe, if you are using the *Shadow Links* implementation of DLDA, to set the invalid links per session threshold to zero or none.

7. Related Work

The traditional way to detect web robots visiting a website has been through analysis of web server logs. Tan and Kumar[16] and Bomhardt et al.[6] recommend this method of find web robots in logs. They create sessions from the traditional logs and look for certain characteristics that indicate a web robot such as requesting robots.txt, having a short avg time between requests, missing Refer fields, and known bot UserAgents. It is also suggested that bots can visit pages that regular humans do not, for example, a crawling bot visiting every link on a page. This can be thought of as a decoy link with an unstructured nature and no attempt to fool a bot. This method however does not detect web robots that adopt a human like request strategy and send browser mimicking header information.

Another way to detect web robots is based on their capabilities. A capability based method using Javascript for detection was introduced by Park et al[13]. They base their detection on the observation that most web robots do not have Javascript enabled. They use Javascript to send back the location of the users mouse in the browser whenever the mouse is moved. It is similar to our approach in that both methods work at the UI level, but bot designers can send back false mouse movement information without having Javascript capability.

It has been observed that bots try to fill out every form field. Websites can exploit this by creating fields in a form that are not visible to the human user in the browser. The human user will not fill out the field because they cannot see it, but the web robot will put some data in the field. Thus, the robot gives itself away. This technique fails when an attacker writes a robot specific to a website. At this point the website is forced to use another technique like CAPTCHA.

CAPTCHA is a common method for blocking web robot use of online services like forums or webmail[17]. CAPTCHA presents a user is with a picture of obfuscated text. The user responds, proving they are human, by typing the text in picture into a field on a form. This is a challenge-response test most humans can pass and current computer programs cannot. In [8] Chow, et al. propose a Clickable CAPTCHA that is more friendly for mobile computing devices like cellphones. In Clickable CAPTCHAs, the user clicks on the pictures of words in set of pictures filled with characters. However, with web robots getting better at breaking character-based CAPTCHAs and a lim-

ited set of valid answers for other types of CAPTCHAs, the CAPTCHA paradigm is losing its effectiveness.

In contrast to the more traditional bot detection strategies, DLDA uses obfuscation to hide valid links. While obfuscation is not generally considered a strong approach for building security, the recent success of Address Space Layout Randomization (ASLR) [14] has shown the effectiveness of obfuscation, especially in appropriate scenarios. ASLR has become default option in both Linux 2.6 and Windows Vista. Although ASLR cannot completely resist all overflow attacks, it has significantly increased the difficulty of launching those attacks.

8. Conclusion

Web bots are increasingly being used for malicious purposes. In this paper we proposed a service called *Decoy Link Design Adaptation(DLDA)* to detect fraud perpetrated by walking web bots. It works by actively modifying the HTML of a website and adding decoy links that cause the web bot to identify itself. Detection of bots is done at the session level looking for sessions with many invalid links.

We implemented the DLDA service as an Apache Web Server module. The module allows a website administrator to easily "plug-in" bot protection for any website and a programming language agnostic protection. The DLDA module added little latency to a generated page. However, it is recommended that no more than 10 links be output by the DLDA module; otherwise, the increase in the size of the page can be significant.

DLDA with 10 links had very good detection rates around 80% with a one minute or less session inactivity timeout. Increasing the session inactivity timeout beyond the bots wait time between visits increased DLDA's detection rate to 100%. DLDA also saw improvements in detection with increasing the number of links output. A drammatic 30% increase in detection was seen going from outputting 2 links by DLDA to outputting 10 links. This increase is observed because more invalid links increases the chance a bot will click one. This can especially be seen when a bot visits only one protected page and clicks a link. There is a difference of 20-50% in detection of web bots when 2 links are output by DLDA versus 10 links output.

We believe this service will be helpful to websites that suspect much of their fraud is conducted by web bots. It will allow them to detect and deal with fraud quickly and with minimal risk of falsely classifying humans as bots.

9. Acknowledgements

This work was partially supported by NSF grants CNS-0716357 and GRA.GP07.I. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Adsense click-bot asap. http://www.scriptlance.com/projects/1236408116.shtml.

[2] Amazon. http://www.amazon.com/.

[3] Jmeter. http://jakarta.apache.org/jmeter/.

[4] List poisoning. http://en.wikipedia.org/wiki/List_poisoning.

[5] Oscommerce online merchant. http://www.oscommerce.com/.

[6] C. Bomhardt, W. Gaul, and L. Schmidt-Thieme. Web robot detection - preprocessing web logfiles for robot detection. *New Developments in Classification and Data Analysis*, pages 113–124, 2005.

[7] D. Brewer, K. Li, L. Ramaswamy, and C. Pu. Transient links. http://www.cs.uga.edu/∼brewer/tl-tech.pdf.

[8] R. Chow, P. Golle, M. Jakobsson, L. Wang, and X. Wang. Making captchas clickable. In *HotMobile '08: Proceedings of the 9th workshop on Mobile computing systems and applications*, pages 91–94, New York, NY, USA, 2008. ACM.

[9] N. Daswani and M. Stoppelman. The anatomy of clickbot.a. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 11–11, Berkeley, CA, USA, 2007. USENIX Association.

[10] J. Goodman. Pay-per-percentage of impressions: An advertising model that is highly robust to fraud. In *ACM E-Commerce Workshop on Sponsered Search Auctions*, 2005.

[11] R. Kloth. Some anti-spam measures to fight spam. http://www.kloth.net/internet/spam/antispam.php.

[12] A. Metwally, D. Agrawal, and A. E. Abbadi. Duplicate detection in click streams. In *In Proceedings of the 14th WWW International World Wide Web Conference*, pages 12–21. ACM Press, 2005.

[13] K. Park, V. Pai, K. Lee, and S. Calo. Securing web service by automatic robot detection. In *USENIX Technical Conference*, 2006.

[14] PaX. Address space layout randomization. http://pax.grsecurity.net/docs/aslr.txt.

[15] M. Prince, L. Holloway, E. Langheinrich, B. Dahl, and A. Keller. Understanding how spammers steal your e-mail address: An analysis of the first six months of data from project honey pot. In *Second Conference on Email and Anti-Spam CEAS 2005*, 2005.

[16] P. Tan and V. Kumar. Discovery of web robot sessions based on their navigational patterns. *Data Mining and Knowledge Discovery*, 6(1):9–35, 2002.

[17] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. Captcha: Using hard ai problems for security. In *In Proceedings of Eurocrypt*, pages 294–311. Springer-Verlag, 2003.

[18] L. Zhang and Y. Guan. Detecting click fraud in pay-per-click streams of online advertising networks. In *Distributed Computing Systems, 2008. ICDCS '08.*, pages 77–84, 2008.