Detecting Web Robots with Passive Behavioral Analysis and Forced Behavior

by

Douglas Brewer

(Under the direction of Kang Li)

Abstract

The use of web robots has exploded on today's World Wide Web (WWW). Web robots are used for various nefarious activities including click fraud, spamming, email scraping, and gaining an unfair advantage. Click fraud and unfair advantage present a way for bot writers to gain a monetary advantage. Click fraud web bots allow their authors to trick advertisement (ad) networks into paying for clicks on ads that do not belong to a human being. This costs ad networks and advertisers money they wanted to spend on advertisements for human consumption. It also affects the reputation for ad networks in the eyes of advertisers.

These problems make combating web robots an important and necessary step on the WWW. Combating web robots is done by various methods that provide the means to make the distinction between bot visits and human visits. These methods involve looking at web server logs or modifying the pages on a website

The enhanced log method for identifying web robots expands on previous work using web server access logs to find attributes and uses AI techniques like decision-trees and bayesian-networks for detection. Enhanced log analysis first breaks down attributes into passive and active methods; it also goes further looking at deeper levels of logging including traffic traces. This allows for previously unexplored attributes like web user agent sniffing, conditional comments, OS sniffing, javascript requests, and specific HTTP responses. Expanding the attribute footprint in log analysis allows for unique web bot identification. Enhanced log analysis is used to generate unique signatures for different web bots. Comparing the signatures allows specific web bots to be identified even when they lie about who they are (e.g. a spam bot pretends to be googlebot).

When log analysis cannot detect sophisticated web robots, web page modification methods like Decoy Link Design Adaptation (DLDA) and Transient Links come into play. These methods dynamically modify web pages such that web robots can not navigate them, but still provide the same user experience to humans in a web browser. The two methods differentiate between the type of a web robot they defend against. DLDA works against a crawling web bot which looks at the web page source, and Transient Links works against replay web bots which make requests with links and data given to them. DLDA detects walking bots by obfuscating real links in with decoy (fake) links. The web bot is forced to randomly choose link and has probability of being detected by picking the decoy link. Transient Links detects replay bots by creating single-use URLs. When a replay bot uses these single-use links, Transient Links is able to tell these links have been used before and are being replayed.

Index words: Web Robots, Web Server, Security, Web Security, Web Bot Detection, Click Fraud

Detecting Web Robots with Passive Behavioral Analysis and Forced Behavior

by

Douglas Brewer

B.S., The University of Georgia, 2005

A Dissertation Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2010

Detecting Web Robots with Passive Behavioral Analysis and Forced Behavior

by

Douglas Brewer

Acknowledgments

I would like to thank everyone who helped me learn anything that I know today. I would like to thank everyone on committee for their help and guidance along the way. A big thank you to Dr. Miller for helping push and expand my programming skills. Thank you Dr. Ramaswamy for feedback and help in make my work better. A special thanks to Dr. Li for letting me do work that I believed in.

I would also like to thank my family for pushing me to do this - My father always said, “You have to at least get a masters these days.” - and providing support.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The World Wide Web (WWW) has become an indispensable medium for sharing information, staying in contact, and commerce. The growth in the usage of the WWW has led to nefarious individuals taking advantage of legitimate services offered on websites and using them to illegitimate ends. While these illegitimate ends can be achieved by human usage alone, it is easier in case of the websites to use what is known as a web robot. The web robot allows the quick and easy repetition of actions, in this case disruptive and/or destructive, within a website.

Web robots can be simply defined as programs that perform a human activity on a website. There are two ways for a web robot to visit a website allowing us to classify web robots as either *crawling bots* or *replay bots*. A crawling bot works by in some way interpreting the HTML of a website to accomplish it's goal; while, the replay bot is given a recording of a human session and typically replays only the HTTP requests. It also possible to have hybrid robot that can use both methods at the same time.

The use of web robots can be both legitimate and illegitimate. The typical use of a legitimate web robot is that of the web crawler. Web crawlers are employed by search engines like Google. They scour the WWW looking for new web pages and updating a search engines views of current pages. The information these robots collect is used to rank web pages against searches performed. There are many illegitimate use cases for web robots, but three of the most popular are financial gain, creating spam, and gaining an unfair advantage. The last two cases typically relate back to financial gain but not always. With these use cases in mind, we can begin to look at the understand the impact that web robots can have on the web today.

The most popular scenario for financial gain with web robots is called click fraud. Click fraud is defined as clicking on an advertisement with no interest in or intention of buying the products or services advertised. This has become a serious problem for companies like Google which make most of their income from Internet advertising. Internet advertising revenue was at $22.7 billion in 2009, but 15.3% of ad clicks were fraudulent for the last quarter of 2009 [30, 13].

The increasing popularity of click fraud can be traced to revenue sharing services like Google's AdSense [3]. With AdSense a website owner can share in Google's advertising revenue by entering into an agreement allowing Google to put its ads on their website. The revenue sharing is based on the number of ads clicked on the website; therefore, it is in the website owners interest to get as many clicks as possible on the advertisements. This can be done by human means, but it is slow and time consuming.

In 2009, 30% of click fraud was conducted by web robots called clickbots. Clickbots allow a website sharing revenue through a service like AdSense to easily and quickly inflate their advertising revenue. However, early clickbots are easily detected because of the clear behavior differences with the quick and repeated clicking of advertisements. However, clickbots have gotten better with Google releasing a study of one of the better clickbots, ClickBot.A [14]. ClickBot.A avoided detection by getting smarter about conducting click fraud. It infected other people's computers and clicked slowly and each infection clicked a maximum of ten AdSense links hosted on the bot author's websites. It is estimated to have cost Google $50,000.

While most people equate spam to email, it has become a problem on the World Wide Web at large with the advent of user generated content. The biggest targets of spam on the web are blogs and forums. When targeting a blog the spammer has the option of either creating a blog on a place like BlogSpot, or putting spam in the comments of another's blog. Spamming a forum typically involves the spammer posting a new topic on the board.

There are two types of spam commonly posted on the web, normal email like spam and link spam. The key difference between the two is their purpose. The normal spam is is trying to sell you something while link spam is trying to get a better search result position for a website included in the spam. To help combat spam, most blogs and forums come with anti-spam modules that try to identify spam messages before they are posted to the public. However, these modules may have little effect on link spam because the message can be relevant. To take care of this scenario, search engines have stepped up and now provide the "rel=nofollow" attribute for links. This tells a search engine that the website displaying this link does not endorse the website it points to. The options to mitigate spam in blogs and bulletin boards are typically not enabled by default and suffer from the same weaknesses as email spam filters.

To help make their spamming easier, spammers have developed bots that can automatically post to blogs and forums. These bots are usually sophisticated having to break a CAPTCHA to sign-up for an account to post their spam. Popular spamming bots that work with most bulletin boards include Forum Poster and XRumer [31, 10]. Both these bots automatically sign-up for a forum account and then post user provided content.

When people need to gain an unfair advantage on the web, they turn to web robots. The reasons for this can vary on a case-by-case basis, but they usually include the need to perform an action again and again over a long period of time in rapid succession. Web robots are very good at this performing the same action at a speed that makes the human rate appear tortoise like. To put this in perspective, it's like doing your taxes by hand versus using a computer program.

Web robots have been used in many cases to gain an unfair advantage over normal people. An interesting case study is that of the Wiseguys and TicketMaster. TicketMaster is a online website where individuals can go to buy concert and special event tickets. Many artists and events have exclusive deals with TicketMaster making them the sole provider of tickets. Recently many of the more popular shows would sell out the best seats more quickly than should happen. A big part of this was the Wiseguys group. They had a large number of computers loaded with a bot that was able to buy tickets from TicketMaster. They would tell their bots to go to the most popular events and concerts and buy tickets for all the good seats. When people would look for good seats at events, they would be forced to buy from the Wiseguys at a hefty markup; they'd figured out a way to mass-scalp tickets online.

Another case of bots with an unfair advantage is Woot Bag-of-Crap bots. The Woot.com website at random times runs specials called the Bag-of-Crap (BoC). The BoC contains a set of three items for three dollars. The items in the bag are randomly selected from the Woot.com warehouse when the BoC is bought, thus a BoC could potentially a very valuable item like a brand new television or camera. Because of this, people will try to buy as many BoCs as they can trying to get a good item at a ridiculously low price. People use bots to help them accomplish this because their is always a limited supply that sells out quickly, so the faster they can buy the better chance, they believe, of getting a high priced item for cheap.

The use of web robots in these unsavory acts presents a need to stop them from committing these acts. The first step to stopping robots is being able to detect their use. There exist general solutions to this based on log analysis and AI hard problems, CAPTCHA. These general approaches have limitations that can be exploited in some cases to avoid detection by the bot writers. In some of these cases, specific solutions have been described like Duplicate Detection for click fraud. However, it is desirable to have a general solution that can avoid limitations and be used constantly for detection.

1.1 Contributions

To avoid limitations in the current solutions, we introduce an improved log based approach and two new approaches, *Decoy Link Design Adaptation (DLDA)* and *Transient Links*, based on modifying the website. Our log based approach increases the attribute space for bot detection, and more importantly, it allows the unique identification of web robots (e.g. Determine GoogleBot, MSNBot, etc...). DLDA and Transient Links provide the ability to human like web robots and can be totally unobtrusive to the human user even though they modify the website. Finally, DLDA and Transient Links work at every web request (or click) allowing for quick detection.

The improved log based approach expands the type of logs used for detection and increases the the information gathered from the logs. This is used to create signatures for each visit to a website. These signatures give behavioral information relating to web robots. This behavioral information allows the detection of web robots and to matching web robots with a known set. Thus, web bots can detected, and they cannot lie about being a different web robot (i.e. We can uniquely identify web robots).

This improvement in log detection does not necessarily detect all web robots. An approach that forces the web robot to reveal itself is necessary to guarantee detection. To this end, two methods were created that cause a web robot to have markedly different behavior than a human visiting a website. These methods involve modification of a website to detect the two classes of web robots described earlier.

In order to detect a crawling web robot, we use *Decoy Link Design Adaptation*. DLDA is based around hiding the original, valid links on a web page within groups of invalid links. The invalid links when clicked will identify visitor as a web robot. The reason for surrounding original links with invalid links is making it difficult to impossible for the web robot to distinguish between types of links and give it a probability to click the invalid link.

Since we can now detect crawling robots, we must now detect replay robots visiting the website. We accomplish this with *Transient Links* which modifies the URLs to prevent their use more than one time. With these, one-time URLs, a replay robot can no longer work against the website. If a robot attempts to replay any previous URLs, the website can detect this and make the determination that the visitor is a web robot.

Explaining and expanding the contributions introduced here, Enhanced Log Analysis is discussed first with a particularly emphasis on the ability to uniquely identify web robots. We will then follow with quick explanation of DLDA and Transient Links along with what makes them unqiue and the background necessary to understand their implementation and use. Next Decoy Link Design Adaptation is discussed in detail and followed by a detailed discussion of Transient Links.

1.2 Sessions

Sessions are an important abstraction for our improved log detection and Transient Links. Sessions group requests that came from the visit to a website. This gives us more information for improved log detection and allows us to handle problems with Transient Links.

The typical thought when hearing the word session is logging into a website. This usually involves a the server retaining some information about log-in. This, however, is not the meaning of session needed for detection.

In our case, session refers to a grouping of requests from a log or logs. Requests are grouped by Internet Protocol (IP) address. This address is used to uniquely identify one computer on the Internet to another computer.

A restriction is put on a session that allows the same IP to produce more than one session. A session must not go for more than 30 minutes without a request. In the case were an IP does not have request for longer than 30 minutes, the current session for that IP is considered terminated and another session started.

This is a general description of a session. How log detection and Transient Links each use sessions will be explained later.

1.3 Related Solutions

We have seen how people are motivated to use bots for financial gain. Now we need to stop these web robots from carrying out their activities. There have been many approaches in the past that have tried to stop web robots. Some in a general sense and some against specific web robot uses. Here, we will look at these solutions and try to understand their advantages and limits.

There are two basic kinds of general solutions in current use today log analysis and Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA). They both work to detect web robots with CAPTCHA having an easy transition into stopping web robots. The log analysis solutions analyze the web server logs looking for tell-tale signs of bot activity. CAPTCHA is embedded within a web page and asks a user to solve a problem that is hard for a robot but easy for a human. The solution is then submitted to the web server and verified.

The traditional way to detect web robots visiting a website has been through analysis of traditional web server logs. Tan and Kumar[32] and Bomhardt et al.[9] recommend this method of finding 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. This method however does not detect web robots that adopt a human like request strategy and send browser mimicking header information.

CAPTCHA is a common method for blocking web robot use of online services like forums or webmail [34]. CAPTCHA presents a user 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 [12] 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 CAPTCHA is losing its effectiveness.

Image recognition CAPTCHAs hope to avoid the web robots improved affectiveness by having humans and computers recognize specific items in images. Asirra is an iplemention of an image recognition CAPTCHA that asks users to identify kittens in a set of pictures [15]. However, Asirra was broken not long after its inception not instilling confidence in this type of CAPTCHA [16].

A specific solution against the use of a web robot is duplicate detection for click fraud. It uses all the information sent back with a advertisement click to determine if the click on an advertisement was a duplicate. If a duplicate click is detected, all matching clicks are discarded as fraud.

Detecting duplicates has been proving effective at discovering click fraud[23]. The observation in click fraud has been that to make significant amounts of money advertisements must be clicked repeatedly and within a short period of time. With the advent of contextual online ads, it is likely the the same user clicks the same ad more than once when committing click fraud. Metwally et al. base their solution on Bloom Filters using them to detect duplicate clicks on ads with a less than 1% error rate.

Zhang and Guan improve on the Bloom Filter based approach[36]. They designed timing Bloom filters (TBF), that allow for the optimal incremental discarding of each previous click instead of large portions of clicks with the previous approach. This method of discarding past clicks decreases the error rate of duplicate detection to less than 0.1%. However, duplicate detection does not differentiate between a bot and a human. With sophisticated Clickbots that act like humans, advertising commissioners may very well start discarding valid clicks.

Another way to stop clickfraud is to change the current Pay-Per-Click model used for online advertising. Goodman suggests that online advertisers no longer pay for clicks but for a percentage of impressions [17]. In the online ad world, impressions on displays in a users browsers. This removes the monetary cost to the advertiser for clickfraud because they no longer pay-per-click. Juels et al. suggest a premium click model [21]. This model provides advertisers the assurance through a trusted third party that click came from a person not conducting click fraud. This, however, requires that information is shared between the third parties which is hard with todays browser security restrictions.

Chapter 2

Enhanced Log Analysis

The first form of bot detection was based on web server log analysis. In this type of analysis a human or a program would go through the logs looking for the tell-tale signs of web robot activity. Log analysis has not changed much since it was first used. The primary indicator of bot activity is the useragent string sent by the web site visitor.

This traditional log analysis has worked well for a long time with typical bots, search engines and web archivers, being well behaved and sending a bot indicating useragent string. However, as the web has grown so has the use of bots that are supposed to be viewed as human. This has proved problematic for using the useragent string alone for web robot detection.

The useragent as recorded in a web server log is sent along with the HTTP request made to the website. This gives the person sending the request control over what useragent string is reported in log. This control allows a web robot to send a useragent string saying they are a typical web browser. This removes the possibility of detection through the useragent string negating it as the primary indicator.

Web log analysis has been expanded to look at bot behavior producing more indicators besides the useragent string. Some bot indicating behaviors are requesting the robots.txt file, multiple requests faster than human could make, and not requesting images and/or other embedded media. The problem with these indicating behaviors is the same as for the useragent string; the bot can be programmed to have or avoid the necessary behaviors. In fact, most bots trying to avoid detection as a bot will already avoid requesting the robots.txt file.

Observing bot behavior is still a good idea even knowing the behavior can change. Bots will exhibit the same behavior until reprogrammed. This gives the person detecting bots the ability to potentially pick out and block the bots before their behavior changes. However, this does not remove the pitfalls of traditional log analysis.

More reliable detection through behavior will need to look at more than the web request behavior of a robot. When a robot makes requests for web pages it sends much more information than is recorded in the web server access log. Therefore, logs other than the web request logs are needed. Two of the logs that can be useful are request header logs and traffic traces.

Header logs and traffic traces give us more surface area to fully explore the web robot's behavior. We explore how the robot reports information to the server through the headers it sends by enabling more logging options on the web server. Capturing the Transmission Control Protocol (TCP) communication between the robot and the web server also gives a window into the robots network behavior.

Header and network behavior of a robot can give signals that are harder to change. Web browsers send headers based on headers sent from the web server and previous requests. This means that robots have to mimic these intricacies when they send headers to the web server, and in most cases, the web robots try to get away with sending as few headers as possible. The network behavior of a web robot is very much dependent on the operating system (OS) that the robot is running, configuration of the OS network stack, and request handling. Making changes to this behavior is involved and requires changing how the OS handles network communication.

So far, observation is limited to passive methods looking at data sent by robot to the web server. It is also useful to include data in response to a robot or requester that can cause them to avoid or make extra requests to server. These avoided or extra requests allow us to determine useful information about the program used to make requests. The avoided and extra requests can be implemented by taking advantage of how programs parse response data. This allows the determination of parsing behavior of the program and in cases where a scripting language is included in HTML, scripting engine availability.

The information gathered from log analysis creates signatures. Signatures are the groupings of attributes that are calculated over a session. Signatures can be used to determine if a visitor was a bot or human and in the case of a bot determine the type of bot.

2.1 Extended Logs

Previously log analysis has only looked at web server access logs. This has produced a limited set of attributes to look at for bot detection. While the set of attributes from normal access logs can be expanded, new types of logs provide new sources of information and attributes.

Header logs provide new information about the HTTP requests; they contain all the HTTP headers sent to the web server. This information can be useful because some bots incorrectly handle HTTP request headers. Examples of this are seen when bots forget to send the "Host" header and when they send empty headers like a "Cookie" header with no data. These types of mistakes are rarely made by web browsers because their HTTP code has been thoroughly tested and worked on over many years.

Header logs also provide an interesting ability called UserAgent sniffing. In an HTTP request, the requesting program, a web bot or browser, sends a header called "User-Agent" in which it identifies itself. It is common for web bots to lie about this to help avoid detection. However, many web browsers have unique header orders and header attributes they send in HTTP requests. This allows for the detection of the real User-Agent via the header log. Web bots who fake the "User-Agent" header do not fake the header orders and attributes allowing for the detection of their lie.

Traffic traces go even further than header logs providing network level information about a host making an HTTP request. This can be useful in that bots can fool with network data and traffic; where as, humans are reliant on their web browsers well established network code. Another interesting thing is individual operating systems (OS) have unique finger prints at the network level. Some bots are designed to operate on a single OS. Examples of this are XRumer, EmailScraper, and IRobot which only run versions of Microsoft Windows. These bots are used in our testing and will be discussed more later.

2.2 Attributes

The attributes chosen for enhanced log analysis are a based on three types of logs. The web server access log, client header logs (forensic logs), and traffic logs. This three logs provide the ability to view a web robots behavior from the perspective of visiting the web page all the way down to the network level.

The logs used determine the amount of information gatherable. Using only the web server access log, only information about the bots request behavior can be gathered. Forensic logs give provide the information necessary to see the behavior of a bot when a file of a specific type is requested. Finally, the traffic logs shows the bots traffic as is sent from the OS.

The traffic logs are the best logs to use for analysis. They contain all the information present in the access log and the forensic log. However, the forensic log must be combined with the access log to provide it's enhanced information.

The logs are partitioned into sessions which are then analyzed. The analysis of the sessions creates a set of attributes for each session giving us information about the behavior of the person or robot that created the session. These attributes are then used to decide whether the session was created by a person or robot. The attributes are also used to try to determine that type of robot by matching against a known set of web robots and their attributes.

We originally created 31 attributes for use in detecting and matching web robots which are summarized in Table 2.1. These attributes were based on typically observed behaviors seen in previous log a based methods like inter-request time and user agent. We also expanded attributes by using the HTTP specification and taking a novel view on previous techniques [19].

Table 2.1: Summary of 31 log based attributes for bot detection.

Attribute Name	Passive/Active
robots.txt	Passive
Inter-Request Time	Passive
Inter-Request Time Variance	Passive
Entropy	Passive
Number of 404s	Passive
400 Response	Passive
URL Similarity	Passive
URL Resolution	Passive
Proxy Requests	Passive
Referer Sent	Passive
Similar/Same Referer	Passive
Favicon Requested	Passive
IE Comments Parsed	Passive
HTML 5 Video Request	Passive
IP Scanning	Passive
Search Scanning	Passive
Web Bot UserAgent	Passive
Has UserAgent	Passive
HTTP Version	Passive
UserAgent Matching	Passive
Host Header Sent	Passive
Cookie Header Sent	Passive
Keep-Alive Header Sent	Passive
OS Matching	Passive
Reset Packet Count	Passive
Port Number Reuse	Passive
Packets Per HTTP Request	Passive
Javascript Callback	Active
307 Redirect Correct	Active
NMap OS Matching	Active
503 (Retry) Response	Active
305 Use Proxy Followed	Active
Long Term Cookie Storage	Active

The attributes are broken down into two groups passive and active. The passive attributes are based purely on the data sent by the requesting person or robot. Active attributes are based on us sending a specific response to a request. A simple example of this is using the HTTP 307 response code with a POST request which requires the requesting browser or program to make the exact same request again to the URL supplied in the Location header.

The complete set of 31 attributes is listed below with a description and why each potentially provides useful behavioral observation:

1. Passive

   I Access Log

   (a) robots.txt — The robots.txt file is requested by well be behaved web robot when they visit a website. It is used to keep web crawlers from visiting parts of a website. Human users rarely request this file, because it is unlinked in a website.

   (b) Inter-request Time — This is the time taken between two web page requests. Humans must wait for a web page render and find links before they can make another request. Robots have a speed advantage in find links in the returned web page, and they can also visit links given to them in a file without waiting for the web page to finish rendering.

   (c) Inter-request Time Variance — This is how much the time between requests changes. Robots have the ability to make requests at consistent intervals while a human must again find links on page causing a greater variation in the time between requests than with a robot.

   (d) Entropy — Entropy is a measure of how a visit to a website was directed or random. A random visit to the website will have a higher entropy; many web search crawlers have a higher entropy when they first crawl the site because they try to crawl the whole website. A visit that is directed to a specific end will likely have a lower entropy because of a single path followed on the website.

(e) 404 Response codes — 404 response codes indicate that the user whether a robot or human requested a web page that could temporarily not be found. Since humans request other web pages from links embedded in other web pages, they rarely visit pages that can't be found. However, robots with a list of links will visit the links even it no longer exists.

(f) 400 Response codes — 400 response codes indicate that the requesting program made an error in HTTP request. Most humans use Web Browsers which have been extensively tested to avoid a 400 response. Thus, 400 response codes likely indicate bot activity.

(g) URL Similarity — This is how similar the requested URLs are to one another. Particularly, paying attention to the file part of the requested URL. Robots sometimes look for the same file in different in locations, where as users view the different files presented as links on a web page.

(h) URL Resolution — When a URL is sent to the web server, most browsers will fully resolve the URL. That is remove things like '..', '.', and extra '/'. Robots can send URLs to web server that leave these things in the URL.

(i) Proxy Requests — A proxy request is one that tries to use a web server to make an HTTP request on the requesters behalf. This occurs with a user sending an absolute URL to the server in the request, that is one starting with 'http://'. Those using browsers are unlikely to send this to an end point website; however, there are many bots written to try to find open proxies.

(j) Referer Sent — Did the requester send a Referer header with their HTTP request. This is typically sent by all browsers when the user clicks on a link. Few browsers will not send this, however many robots will not send this header because it is not required for the HTTP request to complete successfully.

(k) Similar/Same Referer — This is the similarity of Referer URLs. A robot can send Referer URLs to try avoid detection, but it sometimes sends a/the similar/same URL for every request. Web browsers used by humans send Referer as the page where the current request originated.

(l) Favicon — The Favicon is a special image requested from a website (favicon.ico). This icon allows a browser to put the image seen next to bookmarks and sometimes in or next to the URL bar when a website is visited. Robots typically avoid requesting this when visiting a website because they will not display the image to user.

(m) Internet Explorer Comments — These are special HTML comments that the Internet Explorer web browser will read and render as HTML. This mechanism also allows for having only certain versions of Internet Explorer interpret these comments. Most robots will not parse these comments because HTML parsing libraries other than the one used in Internet Explorer will ignore them.

(n) HTML 5 Video Request — HTML 5 is the newest HTML standard from the W3C [35]. This standard calls for "¡video¿" tag to allow browsers to play video without an extension mechanism. It is unlikely that web robots will download the video in this "¡video¿" tag because it is not essential and they cannot play it.

(o) IP Scanning — A bot is IP scanning if it tries to access an undisclosed web server. This requires multiple web servers to detect, accurate detection requires 3 servers with the undisclosed server placed between two web servers that can be reached from a search engine. (Here between means an IP Address that is numerically one more than the first server reachable from a search engine and numerically one less then the other server reachable from a search engine.)

(p) Search Scanning — A bot is Search Scanning if is not IP Scanning. That is to say, if it visits the two web servers accessible from a search engine but does not visit the undisclosed server between the two search accessible servers.

(q) Bot UserAgent — A well behaved web robot will send a user agent string indicating that it is a web robot. This attribute will immediately mark a bot.

(r) Has UserAgent — It is typical to send a user agent string to the server with every request. All web browsers will send a user agent. Bots will sometimes not send user agent strings because they do not set the UserAgent header.

(s) HTTP Version — There are three versions of HTTP available for use HTTP/0.9, HTTP/1.0, and HTTP/1.1. HTTP/1.1 has been available since 1999 and all current web browsers, except Internet Explorer version 6 via option setting, send HTTP/1.1 requests. With Internet Explorer 6 losing market share and an option required to send HTTP/1.0 requests, it can be assumed all browsers only send only HTTP/1.1 requests.

II Forensic Log

(a) UserAgent Matching — A common header sent by most well behaved web browser is the 'UserAgent' header. This header identifies the browser used to request the page. Some robots will try to send the user agent of a browser to avoid detection. This can be detected by user agent sniffing.

(b) Host Header — HTTP/1.1 requires the use of the 'Host' header with a request. This is to support named virtual hosting. However, some robots will leave it out with their HTTP requests.

(c) Cookie Header — This header is sent to the server which issues a cookie. This header contains all the cookie information that was sent from the server and the web browser stored. Robots will typically not store cookies because it is not necessary to complete their objective.

(d) Keep-Alive Header — This header tells a web server to use the connection for more than request and how long the connection should stay alive (i.e. how long to wait before closing the connection). This is sent by some web browsers and is very rarely sent by web robots because it is easier to program a robot to use the connection for a single request.

III Traffic Log

(a) Operating System Matching — The UserAgent header sent by a browser usually includes the OS on which the browser is running. As stated earlier, robots will try to send the user agent string of a web browser. The traffic log can be given to a program to guess the real OS that was used and compare it with the OS stated in the user agent.

(b) Reset Packets — Operating systems have well tested TCP/IP stacks, the parts responsible for sending information over the network. A special TCP packet is called the reset (RST) packet. This packet tells the receiving computer to terminate the current connection with the sender. Robots can sometimes cause problems by leaving network sockets open or closing them early and some OSes are known to have bugs sending extra RST packets.

(c) Port Number Reuse — When creating a TCP connection to a web server a program typically lets the OS choose a port to use. However, a bot writer can ask that specific port be used every time.

(d) Packets Per HTTP Request — Visiting a website the HTML is usually the smallest part of the site. Bots typically avoid downloading images which are usually much larger than HTML causing fewer packets to be transferred for a single HTTP request.

2. Active

I Access Log

(a) Javascript Callback — Web browsers have the ability to execute Javascript, and Javascript has the ability to make extra HTTP requests. Callbacks use this fact to have another request sent from the browser when a web page is loaded. Robots will not or cannot execute Javascript in most cases while most browsers will make the requests without a problem when the page loads.

(b) 307 Redirect — HTTP/1.1 introduced this redirect because of the issues with implementations of the 302 redirect [19]. The 307 redirect response to a request requires that the request be resent to the new URL included in the response. The difference between a 302 and 307 is that the same method of request should be used with a 307, so a POST request that generates a 307 will send another POST request to the new location. Different HTTP libraries handle this differently and different behaviors can be seen for different bots and browsers.

(c) Operating System Matching — This is very similar to the previous OS matching, except matching is done by actively sending data. The program that does this is Nmap [5]. Nmap will send specially crafted TCP, UDP, and ICMP packets to find out the specific OS on a computer. This OS determination is then checked against the OS sent with the user agent.

(d) 503 Response — A 503 response tells the requester the resource is temporarily unavailable. It is possible to send back a time at which the program making the request should retry the request.

(e) 305 Use Proxy — A 305 response tells the requester that the request can be completed using a proxy provided by the returned location header. It is possible to test whether this is implemented or not by returning a response to a proxy server that is controlled by person or company that controls the original website.

II Forensic Log

(a) Long Term Cookie — Cookies have a options to set the length of time for storage by the browser. Thus, we can set it be hundreds of days or years before the browser should discard the cookie from storage. The browser will send this cookie back to use every time a request comes to our server. Most robots will not keep cookies this long if they implement them at all because of the long term storage implications.

2.2.1 Removed Attributes

While we previously described 31 attributes that are possibly useful for detecting web robots through behavior, some cannot be applied. The ability to use behavior as differentiator relies on there being a difference in behavior. In the cases of some attributes, there is no difference in viewed behavior.

The attributes relating to the 305 and 503 response codes were dropped because of implementation inconsistencies with the HTTP specification. Particularly, many web browsers and HTTP libraries consider the 305 use proxy response is insecure[24]. This is because it asks the implementer to send the same exact data to a different location, and if the web server was compromised, the location could be under the control of an attacker; thereby, sending potentially private information to an untrusted third party. The 503 response may ask the requester to retry after a certain amount of time. However, most browsers and libraries will just display the 503 response data and never repeat the request after the specified timeout.

The IP Scanning and Search scanning attributes were dropped because of data collection requirements. It is not always feasible or possible for an organization to run two extra web servers in the configuration necessary to show this behavior. Preliminary testing also revealed no bots that exhibited a Search scanning attribute. IP Scanning still remained a possible attribute after this, but most IP Scanning were looking for non-existent pages and had errors in their requests that negated the need for the IP scanning attribute.

Long term cookies were dropped because of the a trend in increased cookie deletion. A study found that as many as 39% of users were deleting cookies from the computers on a monthly basis[28]. A Long term cookie attribute with this rate of deletion would miss a great a deal of human users possibly grouping them in the same category as bots. Thus, it is better to use the more near term cookie measurements of whether cookies were accepted and returned within the same website visit.

The HTML 5 video attribute is not used because at this time release versions of Internet Explorer (IE) still do not support it. IE is still the most used web browser at this time with around 50% market share [7]. Thus, using HTML 5 video tag for an attribute leaves 50% of the world wide web users showing no potential differences between them and bots with this attribute. However, IE version 9 will include support for HTML 5 video and audio, so in the future, this attribute will be useful and worth including again [18].

2.3 Signature Generation

Bot detection uses signatures created from the described attributes to detect bots. These signatures are generated over sessions. A session is quite simply a grouping a requests from an access log (with or without a Header Log) or traffic trace. Requests are grouped based on how closely in time they occur to other requests from the IP address. This is explicitly stated as "Group all requests from same IP such that no more than X number of minutes has passed since the last request from that IP." X is any number of minutes but for the evaluation of signatures X will equal 30 minutes.

Once sessions are created from an access log or traffic trace, attribute values are generated for that session. These attribute values are combined into a vector to create the signature for a session. This signature is what is used for bot detection and grouping similar bots.

The attributes in the signatures are given values based on the requests and other information gathered from logs for the session described by the signature. The values associated with attributes are based on if the attribute condition occurred in the session or how many times the condition for attribute occurred in the session. Thus, an attribute value can be as simple as a boolean statement of true or false all the way through a percentage of requests. Table 2.2 shows the values that each attribute may take in the signature.

2.4 Bot Detection

Two potential ways to do bot detection are used with the created signatures: heuristic detection and machine learning. Heuristic detection uses 11 attributes from the signature looking for a specific values to determine if a session was generated by a bot. The machine learning approach takes the signatures of a predetermined set of bots and then tries to match to known bots based on signature values.

Heuristic bot detection is the simplest form of bot detection. If one of the 11 attribute values meets a criteria, the session will be marked as a bot. Heuristic bot detection uses the robots.txt, inter-request time, inter-request time variance, 404 response codes, 400 response codes, UserAgent matching, Similar/Same Referer, URL resolution, HTTP version number, and Proxy requests attributes. Table 2.3 shows the value cutoffs used for heuristic detection.

In the machine learning approach to bot detection, a computer algorithm is trained to recognize a signature as either a bot or a potential human. This is accomplished through training the algorithm with a data set of signatures marked as bot and not bot. After the training, the algorithm with the marked data set classification of signatures is done by feeding a signature into the algorithm.

Two machine algorithms applicable for web bot detection are bayesian classification and decision trees. Bayesian classification works over the entire signature creating a probability that the signature belongs to a web bot. A decision tree works by building a tree that is traversed with the signature. When a leaf node of a tree is reached in the traversal of the decision tree, the classification at that leaf node is given to the signature. Thus, difference between a bayesian and decision tree is that bayesian will give a probability of bot classifica-

Table 2.2: Attribute values for Signatures generated from Sessions.

Attribute	***Value***
robots.txt	True if robots.txt requested or False
Inter-request Time	Real number > 0
Inter-request Time Variance	Real number > 0
Entropy	Real number > 0
404 Response Codes	Percentage of Requests returning 404 (0-100%)
400 Response Codes	True or False if a 400 response code appears
URL Similarity	Percentage of similar URLs (0-100%)
URL Resolution	Percentage of Unresolved URLs (0-100%)
Proxy Requests	
Referer Sent	True if Referer header sent; False otherwise
Similar/Same Referer	Percentage of similar/same Referer URLs
Favicon	True if favicon.ico is requests else False
Internet Explorer Comments	True if a request for commented file appears else False
Bot UserAgent	True if the user agent string belongs to a bot else False
Has UserAgent	True if the user agent string was sent for the majority of requests else False
HTTP Version	The version number of HTTP requests 0.9, 1.0, or 1.1
UserAgent Matching	True if sniffed UA matches sent UA else False
Host Header	True if Host header is sent else False
Cookie Header	True if Cookie header is sent else False
Keep-Alive Header	True if sent with request else False
OS Matching	True if sniffed OS matches UA OS else False
Reset Packets	Percentage of Reset packets (0-100%)
Port Number Reuse	
Packets Per Request	Real number > 0 of number packets divided by number of HTTP requests
Javascript Callback	True if callback request is executed else False
307 Redirect	True if same request method to new URL else False

Table 2.3: Heuristic bot detection cutoff values for marking signature as being created by a bot.

Attribute	Bot Cutoff
robots.txt	Value = True
Inter-request Time	Value $<$ 2
Inter-request Time Variance	Value = 0
400 Response Codes	Value = True
Bot UserAgent	Value = True
Has UserAgent	Value = False
URL Resolution	Value $>$ 10% unresolved
404 Response Codes	Value $>$ 30% 404 Responses
HTTP Version	Value = 0.9
Similar/Same Referer	Value $>$ 20%
UserAgent Matches	Value = True

tion where the decision tree will give a direct yes/no answer about the signature belonging to a bot.

2.5 Bot Recognition

Bot recognition is the ability to say that a particular visit was from a particularly web bot (i.e. a visit was from Googlebot). This is particularly useful against web bots that try to hide by faking being a particular web browser. Many malicious web bots do this, and sometimes malicious bots may even fake being a good bot. If a malicious bot fakes being Googlebot, we would like to know this and stop them.

Bot recognition uses the signature of attributes discussed earlier. The signature captures the behavior of a web bot as it visits a website. The behavior captured in signatures can be compared to other known web bot signatures. When a match with a known signature is found, the current visit can be said to have come from the web bot of the matched signature.

This type of a recognition works because web bot behavior is somewhat static. That is to say, one visit to a website will produce a similar behavior as a previous or subsequent visit. This should hold until the web bot's behavior is changed by re-programming.

Comparing signatures of web bots for recognition can be done by matching against similar signatures. There many ways to match similar signatures, but two important methods are direct similarity comparison and clustering. The methods have different strengths and weaknesses relating to matching signatures.

The stated methods for comparison both rely on a distance/difference function. This function is used to determine how close two signatures are to one another. As is the case with most distance/difference functions, the smaller the number produced by the function the closer to the two signatures.

Direct similarity comparison is simply taking a newly created signature and comparing it to known signatures. A signature is matched by finding the closest known signature within a cutoff. The cutoff exists to prevent a forced matching when the signature being compared has never been seen. This gives the ability easily match known bots while detecting possibly new web bots as they visit the website.

The clustering method for comparison is given an entire set of signatures. The clustering starts with no grouped signatures the clustering algorithm is then tasked with joining the signatures together. At the end all signatures have been grouped together in clusters with each cluster representing signatures that show the same behavior. In this case, the same behavior would link the clustered group of signatures to a specific web bot.

The clustering method has two distinct advantages over the direct comparison method. There is no cutoff needed when comparing signatures, and clustering can work without any known signatures. These make clustering a better method when no known data exists. However, when a cutoff and known data exist, a direct similarity comparison is easier and effective.

2.6 Experiments

Testing the improved log analysis required the use of web robots loose in the wild. XRumer, Emailscraper, and IRobot were used to test. These robots each work in different manners and have different goals.

XRumer is given the the base URL of a phpBB2 installation. It has the ability go through the two steps necessary to make a post to phpBB: sign-up and posting. Sign-up for phpBB is protected by a CAPTCHA which XRumer can crack. Once signed up XRumer will post a message of users choosing to the bulletin board.

Emailscraper is the simplest of the bots. Its function is to find email addresses embedded in the HTML of a web page. Given a starting page, Emailscraper will follow all links within a specified depth. It's final output is a list of emails addresses that were found on the web page.

IRobot is a program that allows anyone to create a bot. It works by recording the clicks and interactions with the web page that a user makes. This recording is saved as a script that IRobot is capable of replaying.

To test the robots, two websites, phpBB and osCommerce, were setup. phpBB is a bulletin board system written and the PHP programming language. Bulletin boards allow users to post messages for each other to read and post replies to others. osCommerce is an open source e-commerce store. It allows for anyone to easily setup a online store with items they have for sale. The phpBB website would have been sufficient to test all the robots, but it was worthwhile to also test IRobot on another websites because of the ”create a bot” aspect. Thus, we tested IRobot with a osCommerce.

The phpBB and osCommerce websites were run on the Apache web server with the PHP module installed with the module defaults. The phpBB and osCommerce stores used their default settings except for things explicitly required during setup like database parameters and server URL. The server itself was unavailable from the outside world during tests with bots to avoid interference from outside bots.

When testing the method with these bots, the ability to detect the bots and match up (group together) the same or similar bots were the primary concerns.

2.6.1 Bot Detection

Enhanced log analysis increases the number of attributes available by adding traffic trace logs for detection. To determine if the traffic trace logs are more effective than normal Apache logs, XRumer, EmailScraper, and IRobot were run over the test website. The access logs and traffic traces for their attacks were recorded to see if the increased attributes in traffic traces could provide better detection.

Figure 2.1 shows that all of the XRumer and EmailScraper web bots were detected when using either access logs or traffic traces. However, an increase in detection is seen with IRobot from 88% to 90%. This is because XRumer and EmailScraper run over a website as fast as possible. This leaves them open to detection using only the access log quite easily. IRobot, on the other hand, will try to hide more from detection by allowing the user to select the speed with which to traverse the website. While not decisive, it does show that more attributes available for detection will produce a better result; this is most true when the web bot will try to hide.

Heuristic detection will not work when a web bots tries to hide as can be seen in the IRobot detection rates of Figure 2.1. One way to improve detection has been to use machine learning. A machine learning approach to detection can provide advantages over a heuristic approach because it does not rely on known cutoff values, but it discovers them through training on known data. For testing, the decision tree and naive bayes machine learning approaches were used in testing.

Machine learning requires algorithms to be trained against a set of known data. This was accomplished by taking five signatures from each of the IRobot, XRumer, and EmailScraper web bots and training against them. Real world conditions in signature generation from traffic logs will show selection errors for bot signatures. This selection error was mimicked

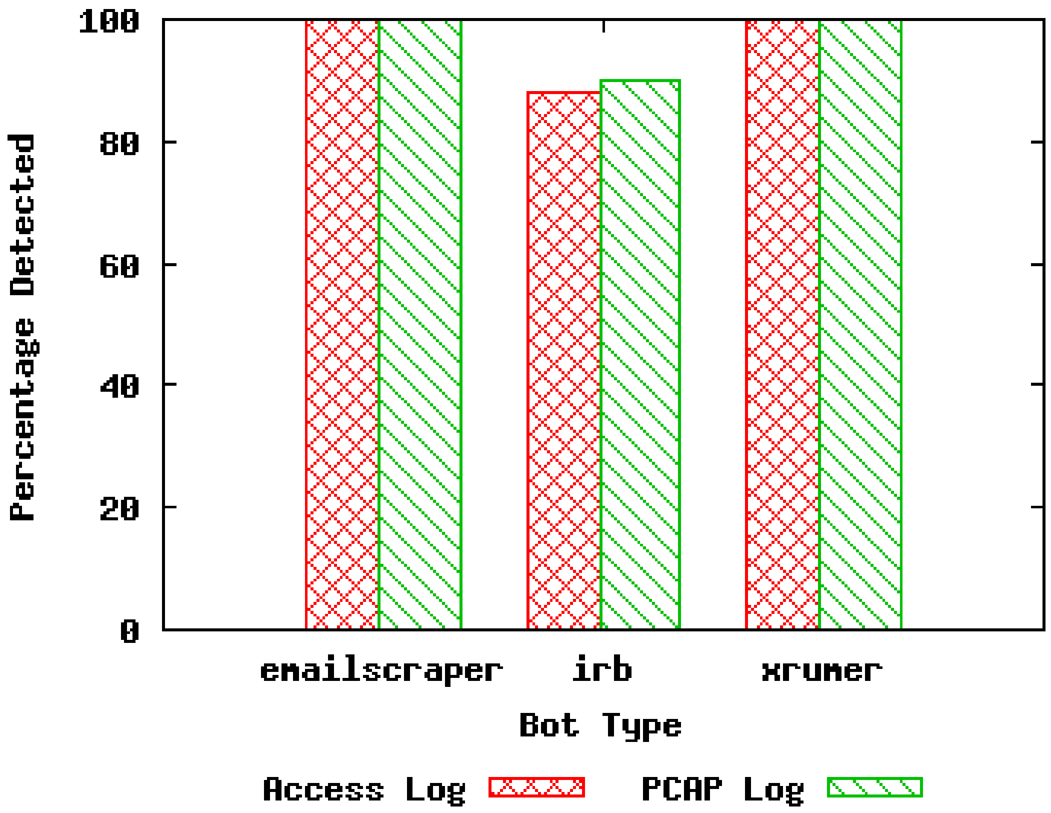


Figure 2.1: Detection of IRobot (labeled irb), XRumer, and EmailScraper by heuristics with Web Server Access Log (Access Log) and Traffic Traces (PCAP Log).

by using noise lines that included possibly mislabeled human and bot sessions. The number of noise lines was varied from 30 to 135 to see the effect.

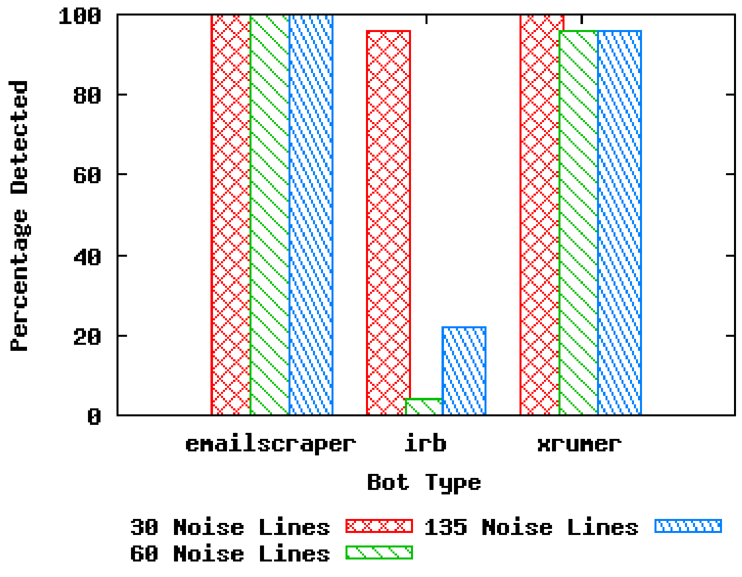
The machine learning approach to detection does better than heuristic approach when using decision trees, Figure 2.2. There is a slight drop in detection for the decision tree of XRumer from 100% to 98% but a large increase from 90% to 100% in the detection of IRobot. The overall increase in detection shows an effectiveness in machine learning that is harder to match with the guess and see approach to heuristic analysis.

2.6.2 Clustering Algorithm

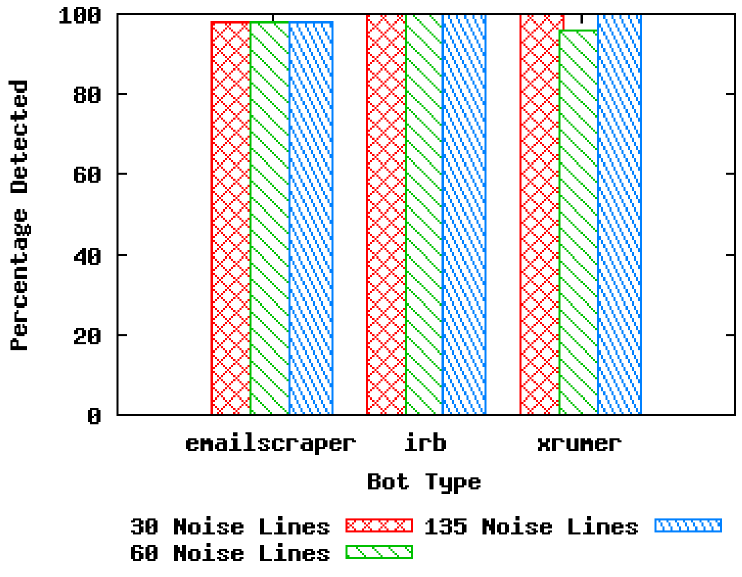
There are many possible clustering algorithms to use for unique identification, but most fall into two main categories hierarchical clustering and partitional clustering. These two types make significantly different starting assumptions for creating clusters. The hierarchical clustering comes in either the top-down or bottom-up flavors. In the top-down approach, the data points are assigned to a single giant cluster and then split into smaller clusters based on distance. The bottom-up approach starts with every data point as a unique cluster, and it combines them into larger clusters based on distance. Partitional clustering starts by assigning points to set of random clusters. The clusters are then refined based on distance of the data points in the cluster to a cluster center.

Hierarchical clustering was tested against the K-Medoids clustering partitional clustering algorithm. They were tested against known search engine crawlers. Signatures were created for examples of Googlebot (Search/Image), MSNBot, YahooBot, and Baiduspider. Each clustering algorithm was asked to create seven clusters; the K-Medoids algorithm was also asked to make 7 passes.

Figure 2.3 shows the ability of each clustering methodology to split the search bots into different groups. Hierarchical clustering, Figure 2.3(b), showed a good ability to group bots, but it could not separate the search bots as well as K-Medoids, Figure 2.3(a). Since the

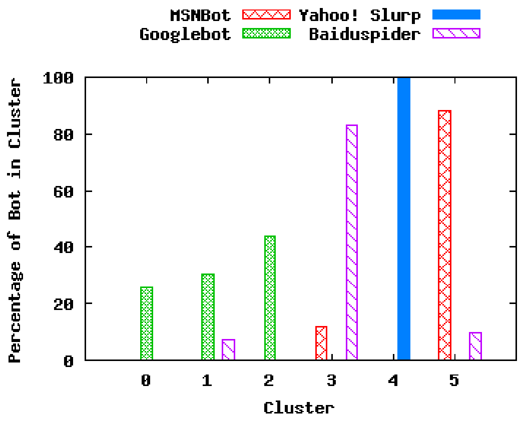


(a) Naive Bayes

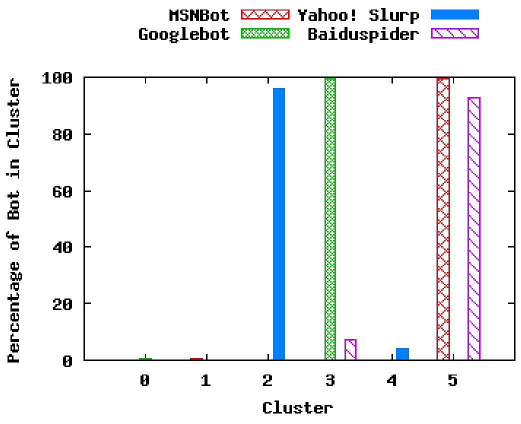


(b) Decision Tree

Figure 2.2: Machine learning detection of IRobot, XRumer and EmailScraper web robots. Each method was trained with the 5 signatures of the known bots and different numbers of noise data.



(a) K-Medoids clustering



(b) Hierarchical clustering

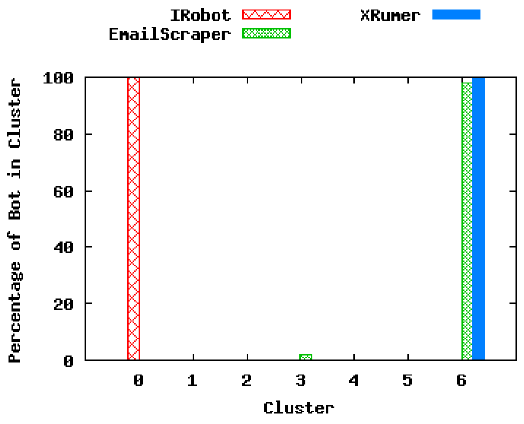
Figure 2.3: Comparison of Hierarchical and Partitional (K-Medoids) clustering algorithms.

point of clustering is to identify unique web bots, the hierarchical method is not as usable as K-Medoids.

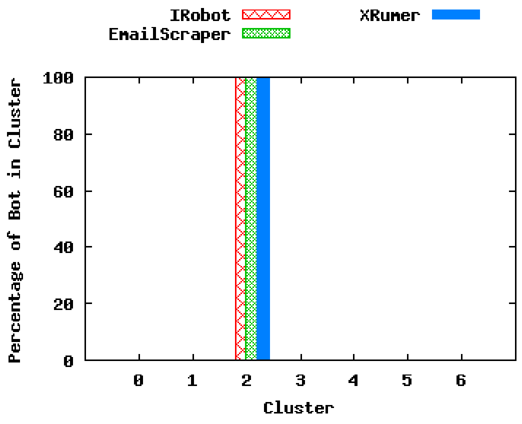
2.6.3 Bot Recognition

We tested recognition of bots using generated signatures by means of the K-Medoids clustering algorithm. Clustering algorithms are reliant on the distance metric for grouping signatures. This required testing different distance metrics for affects on grouping the web bot signatures. If clustering is to be useful for bot recognition, each cluster produced by the clustering algorithm must belong to a single bot or at least be dominated by a single bot. Differing amounts of noise data were added to the clustering of known web bot signatures to see how specific a cluster was to a single web bot.

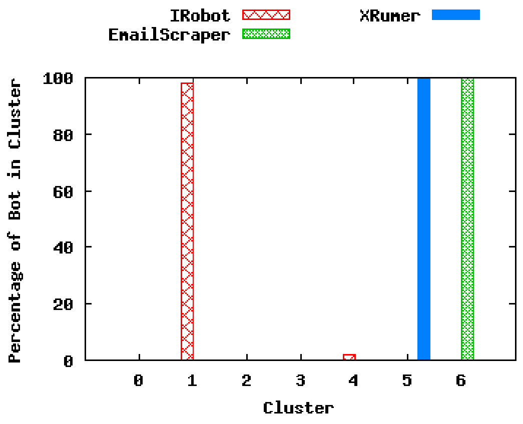
Figure 2.4 shows how the K-Medoids algorithm did at clustering the three web bots based on different distance algorithms. The algorithms tried were those for cosine distance, minkowski distance, and manhatten distance. K-Medoids using the manhatten distance produced the best specific clusters for each of the three web bots.



(a) Clusters using Cosine distance



(b) Clusters using Minkowski distance

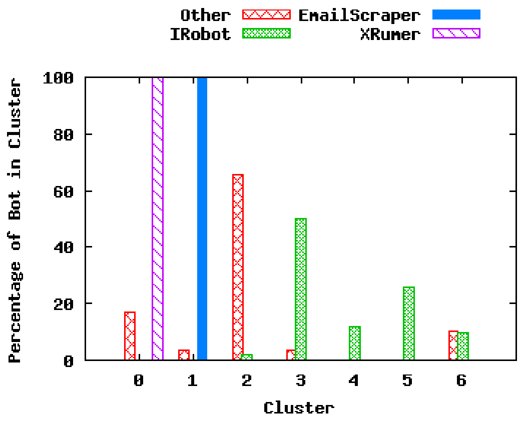


(c) Clusters using Manhatten distance

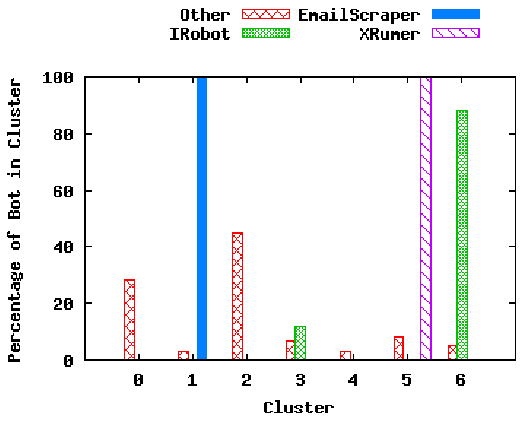
Figure 2.4: K-Medoids clustering output using three different distance measures. The data was split evenly between XRumer, EmailScraper, and IRobot

We can see that the minkowski and cosine distance metrics were better at matching the bot signatures together (e.g. 100% of the IRobot signatures were in the same cluster). They, however, do not separate the bots well. Minkowski clusters all three bots into the same cluster, and the cosine distance only separates IRobot. This grouping of the different web bots together can not happen.

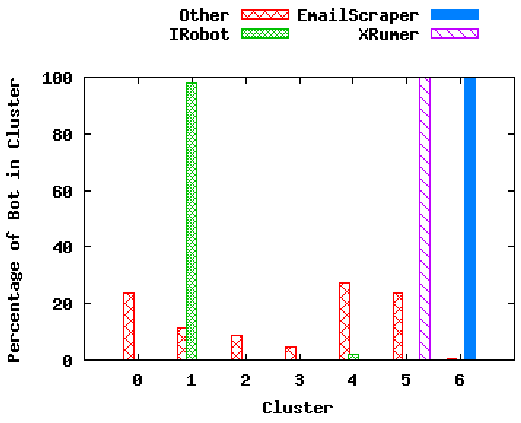
Figure 2.5 shows that noise data had very little affect on clustering the three web bots. XRumer, EmailScraper, and IRobot all maintained separate clusters for themselves. It is also interesting to see that the noise in the data did not cluster closely to any one specific bot.



(a) Clusters with 20% noise data



(b) Clusters with 31% noise data



(c) Clusters with 48% noise data

Figure 2.5: K-Medoids clustering output with increasing amounts of noise data.

The clusters can also be broken down in what percentage is actually the known bot versus the noise. Cluster 1 is 77% IRobot and 23% other, cluster 5 is 61% XRumer and 39% other, and cluster 6 is 98% EmailScraper and 2% other. The fact that clusters 1 and 6 show an overwhelming majority of IRobot and EmailScraper, respectively, provides confidence that clustering can identify unique bots.

```
GET / HTTP/1.0

POST /?p=add_comment HTTP/1.0 name=PillenTake
&email=amomoreofenes@atlantmail.com
&comment=<size>5%5D<b><center>The+Best+Quality+Pills.
+Zovirax+Online+Pharmacy%21+Buy+Discount+ZOVIRAX
+-+Order+ZOVIRAX+-%0D%0A<i>
```

Figure 2.6: Example of Comment Spammer requests.

Real World Website Testing

To get an idea of how bot recognition works with real world data, it was tested against a traffic trace of several websites. These websites belonged to several small businesses all interested in maintaining a web presence. The amount of traffic seen was not enormous but enough to produce 1.5GB of traffic trace logs in two days.

Heuristic detection was run over the signatures output from the traffic trace. It was found that about 80% of the requests were made by bots most sending no user agent and many of the rest being search engine crawlers. However, a few signatures with browser user agents appeared as bots as well. Investigating the possible browsers appearing as bots, their logs showed spamming bot behavior. These bots sent POST requests with spam selling drugs and jewelry to an online comment form. They all sent similar spam and in a similar manner giving the impression that they were the same type of spam bot from here on called the Comment Spammer. Example output of Comment Spammer can be seen in Figure 2.6.

The signatures were also clustered and in the output groupings Comment Spammer signatures were grouped with signatures that were not detected as bots. Looking into this grouping in greater detail, the grouping appeared to have clustered with undetected signatures of Comment Spammer. A total of 145 Comment spammer signatures were grouped together in this cluster. This real world data test of Bot Recognition is important and shows that it is very useful in detecting missed web bots.

Chapter 3

Forced Behavior

So far, I have described how web robots can be detected with *passive* analysis of their behavior. Robots were attributes exemplified by how fast they could make pages requests and whether they implemented javascript based HTTP requests. These methods of detection leave open the possibility that sophisticated web robots will not show up as such.

It is necessary to have a detection method that can do more than split the users into a classes of web robots and unknown. A better classification is to classify user of a website as either a web robot or a human being. This is not entirely possible with only analysis of the passive and active behavior. A differentiating behavior between web robots and humans must be created and exploited for classification.

The second type of behavior analysis to increase classification specificity is called *forced behavior*. The idea behind forced behavior is elegant in its simplicity. A robot and human are presented with challenge; the challenge is easily solved by the human, but for the robot it is impossible or hard. These methods have traditionally been based on humans being able to recognize patterns with greater accuracy than robots.

The challenge used to force specific behavior for differentiating between web robots and humans is based on adding decoy links. Decoy links are just links on web page that lead to a page that is not supposed to be visited by humans. It is important that the decoy links stand out to humans, but are cannot be picked out by web robots. This necessitates that the links look similar to both humans and web robots, but are easily visible within the web page layout to humans.

The layout of the web page is used to create cues to humans about which links to click on the web page. This method is called *Decoy Link Design Adaptation(DLDA)* Common cues may be the size of links or the color of links. These cues allow the humans to always select and click on non-decoy link while the web robots are left to guess.

This challenge will force behavior only if a web robot is made to chose from the links on a web page that includes decoy links. A person running a web robot could circumvent DLDA by giving the robot a script of links to visit. This type of a robot is called a replay robot, and must be detected such that the only viable option is an attempt to click links from the page source.

Detection of replay robots is done with one-time use links called *Transient Links*. The one-time use property of the links means they are only valid for a single visit to a website. If the links are used on subsequent visits, the use of the links triggers an alert that the user browsing the website is a web robot.

3.1 Dynamic Web

The forced behavior methods DLDA and Transient Links require graphical display of HTML, modification of HTML on the client side, and dynamically generated HTML from the server. This requires three fundamental technologies of today's web: Dynamic HTML (DHTML), Cascading Style Sheets (CSS) for positioning, and server-side scripting. While these exist today, they have not always existed and still to some extent do not perform perfectly on current web servers and web browsers. However, the support for DHTML, CSS, and server-side scripting has reached a level where we can implement techniques based on dynamic web page modification.

The World Wide Web (WWW) was created between 1980 and 1990 and was a much simpler incarnation of what we see today. If there had been a real need for bot detection in the early days of the WWW, only log based techniques would have worked and not our dynamic page modification. The simple reason being the web browser and server could not support it.

At this point in the World Wide Web, the HTML was considered static and could not be modified on the client-side. The web browsers in this early web were only able to interpret the HTML to show the text of the web page. The limitations of this method of presentation meant that specific HTML elements could not be positioned. In order to bring positioning into a web page, the browser needed to go through an upgrade to the *graphical web browser*. The first graphical browser was Mosaic produced by the National Center for Super-Computing Applications (NCSA) [25]. It the was first browser to display web pages graphically and was built around the X-Windows system.

3.1.1 Dynamic HTML

NCSA Mosaic was the first graphical web browser and was introduced in 1993. The big innovation with Mosaic was desktop GUI integration. People no longer had to look at pure text, but could view a web page with embedded media content, particularly images within HTML documents. This was the fore-runner to more modern GUI driven websites that could take advantage of the new rendering with DHTML.

The first steps to supporting DHTML occurred when both Netscape Navigator and Internet Explorer introduced a client-side programming language and access to page elements within that language. Netscape released the first client-side programming language, Javascript in version 2.0. Internet Explorer version 3.0 followed with Microsoft's own implementation of Javascript called JScript. These first implementations had limited support for accessing web page elements, but not full support for DHTML.

The first browsers to support DHTML were released in 1997. They were Netscape Navigator 4.0 and Internet Explorer 4.0. Navigator's DHTML was limited in that the page could not be re-rendered after it was loaded; this limitation was not shared by Internet Explorer.

Not surprisingly, the support for DHTML in both browsers was incompatible. This lead to a choice for web developers of supporting one browser or the other.

The big breakthrough for DHTML came later in 1997 when the Document Object Model (DOM) and the ECMAScript (Javascript) standards were introduced. The DOM and ECMAScript specifications allowed for DHTML to operate similarly across the different browsers. Thus, developers could be reasonable assured that their website would operate correctly.

3.1.2 Cascading Style Sheets

DHTML provides the means to manipulate the HTML on the client side, but it does not allow to specify how to place the HTML elements on the screen. This describing of the placement is called the layout, and it is accomplished with Cascading Style Sheets (CSS). CSS has become more and more necessary as web pages have turned into web applications. It allows for a very specific layout of all page elements allowing the website designer to create nearly any look on the page.

The most current specification of CSS is version 2.1 and it allows for a great deal of flexibility in positioning elements. It allows us to perform the positioning of HTML elements necessary for DLDA to work correctly. Specifically, the CSS position directive with the values relative and absolute and the CSS z-index directive provide the ability to overlap HTML elements when they are displayed graphically. This allows for the creation of decoy links that are hidden in a graphical environment.

We've seen that in principle CSS allows us to position elements correctly on a page. However, there are still problem with using CSS to position to elements on a web page, and they all stem from browser support for the CSS specification. Different browsers have differing levels of support and differing incorrect implementations for some parts of the specification. These inconsistencies have made CSS more of a hit or miss proposition.

The inconsistencies in the implementations of the CSS specification means that a web developer is required to possibly have many different CSS files for a website to enable similar display within different browsers. The biggest offender until recently has been Internet Explorer (IE). IE was not able to handle much of CSS version 2 correctly. This led to writing a website twice once for IE and once for other browsers.

3.1.3 Server-Side Scripting

Server-side scripting allows for web servers to generate different HTML each time the same resource is requested. This is important for DLDA and Transient Links. They both need to serve different modified HTML each time a web page is visited.

The start of server-side scripting began with the Common Gateway Interface (CGI) [29]. CGI allows a web server to execute a separate program to handle the HTTP request. The program is given all information pertaining to the request and is tasked with sending back a response. The original implementations of CGI were slow because every request required the web server to start the program to handle the request in a new process.

To alleviate the problems with CGI overloading the web server hardware because of the one process per request model, scripting methods that were run in the web server process itself were created. This meant that the web server had to evolve to load in separate code from other vendors. This was accomplished by web servers introducing extension points that allowed third parties to hook into their request/response processing and data handling.

The arguable most famous in-server scripting language is PHP: Hypertext Processor [33]. The PHP scripting language allows you to setup a web server extension and then write web pages in PHP. When a request is made to a PHP page the server passes the page to the PHP extension which then parses and executes the code. The important feature for PHP has been the direct inclusion of HTML with the PHP itself.

The server extensions introduced in response to in-server scripting languages are useful for DLDA and Transient Links. It allows us to rewrite any HTML the web server sends back. This gives us the ability to introduce our forced behavior without specific developer intervention.

3.2 Navigation Behavior

The idea of forcing bot behavior is unique. Previous work focuses on finding bot behavior after the fact or trying to separate humans and bots based on capabilities. Forcing bot behavior requires that humans and bots uses the differing capabilities of bots and humans, but makes decision about a website visitor being a bot or human based on a navigation behavior.

There is an important difference between human and web bot navigation behavior. A human views the web page through a browser where they visually select links and click them with a pointer. Web robots can have two different navigation behaviors. They may either navigate via a script (a Replay Bot), or they view the HTML and select links via some programmed method (a Walking Bot). The visual selection by humans of links gives an important, exploitable difference.

This realization leads to *Decoy Link Design Adapation (DLDA)* and *Transient Links*. DLDA obfuscates valid links in HTML by surrounding them with invalid links. The invalid links when viewed in the web browser give the human a clue that should not be clicked but the bot is left without this information thus exploiting the humans visual navigation. Transient Links modifies the links of a web page to create single-use links that when re-used indicate a web robot. The observation that a human clicks links the browser visually means that they do not replay links.

The next two chapters will show how DLDA and Transient Links are actually implemented using this navigational behavioral difference. DLDA is explored first followed by Transient Links.

Chapter 4

Decoy Link Design Adaptation

Decoy Link Design Adaptation(DLDA) is based on using decoy information in web pages to confuse bots and hopefully identify them. There have been many different approaches to this list poisoning, hidden form fields, hidden links each for stopping a different kind of web bot. List poisoning works against email address harvesters by polluting their lists of harvested addresses with invalid emails, hidden links are meant to detect bots that crawl website like web scrapers, and hidden form fields are meant to detects bots like spam bots that send information to a website.

The contribution of DLDA to this long history of decoy information is the structure of its use. The DLDA technique is based around including hidden decoy links within a web page to obscure an original link on a page. Using this obfuscation makes it hard to for web bots to select the correct link to follow when they are looking at the source of a web page. The decoy links are then setup in such a way that they indicate a web bot has visited when they are clicked.

There are many ways to create a challenge for a bot using DLDA. Since DLDA is based around obfuscating links, implementations of DLDA should be based around the myriad of different strategies for grouping links. Some of the more popular strategies include:

- Tag Clouds – A group of links where each is given a size based on some criterion with the most relevant having the largest size. *add image*

- Link Bars – Typically seen at the top of web pages containing links like “Home” and “About Us”. *add image*

- Tabs – Used mainly in web applications, they separate information by grouping it logically within different pages. The links in tabs are the tab headers. *add image*
- Menus – Pop-up menus have become popular in menus on web pages. They group links together based on a common heading usually based on what tasks can be accomplished. *add image*

4.1 Human User Transparency

Given the use of graphical web browsers, we need help human user distinguish DLDA links with visual cues. The most common case means giving the valid links in DLDA a distinct, attention drawing look in the web browser while making invalid links look similar to each other. 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.

Not all of the strategies for grouping links presented so far are applicable or usable for DLDA. For DLDA, it is very important to consider how a user will view the obfuscated links on a page. The main goal of DLDA is distinguish human users of a website from bots visiting the website. The ability to distinguish requires the humans to avoid the DLDA links while still capturing the bots.

Link Bars and *Menus* can be used for grouping links, but are not a good choice for DLDA. People have been trained to view all links within these as clickable and valid. We must use grouping methods that show a preference for people clicking the valid link when it is shown to them. We therefore will drop Link Bars and Menus from consideration and add two more to our list for potential DLDA usage.

- Multiple Links - Multiple Links is a link grouping technique where multiple invalid links are placed on within a web page. The links are then rendered off the page by the web browser with CSS fixed positioning. With this technique the user is unable to click the invalid links on the web page because they are rendered off-screen.

Table 4.1: Landing Page description for each of the four link grouping techniques.

Tag Clouds	The landing page for Tag Clouds had the valid link as the largest link in the tag cloud.
Tabs	The landing page for Tabs had the invalid links all the same color and the valid link with a different color.
Multiple Links	Invalid links were positioned off the page with CSS while the valid link had not styling information.
Shadow Links	The valid and invalid links were stacked on each other with "position:absolute".

- Shadow Links - Shadow Links groups invalid links and a valid link together within a logical structure. When the links are displayed on the page the invalid links and valid link are positioned to overlap with the valid link placed on top. The human user is unable to click the invalid link because they are always hidden beneath the valid link.

With four possible techniques for testing, *Tag Clouds, Tabs, Multiple Links, and Shadow Links*, we need to determine which is most likely to allow human users to avoid the invalid links. We did this by setting up an experiment where humans and robots were tasked with navigating each technique exactly once. This allowed us to get an idea of which technique(s) were the bets to pursue for further study and implementation.

We setup four web pages, called landing pages. These pages each contained one link set consisting of a valid link and many invalid links. Each page was then programmed to use of the four listed possible techniques. Table 4.1 describes how each was setup with the chosen technique. They were then put online and linked a fake e-commerce website we ran. The linking to the commerce website was to give the user bot or human a task on the e-commerce site and see how each landing did or did not hinder their progress.

Figure 4.1 shows how well each technique fared against humans and web robots. We can see that most techniques did well against the robots with over 50% detection, but when it comes to humans the Tag Clouds and Tabs falsely identified the humans 30% and 50% of

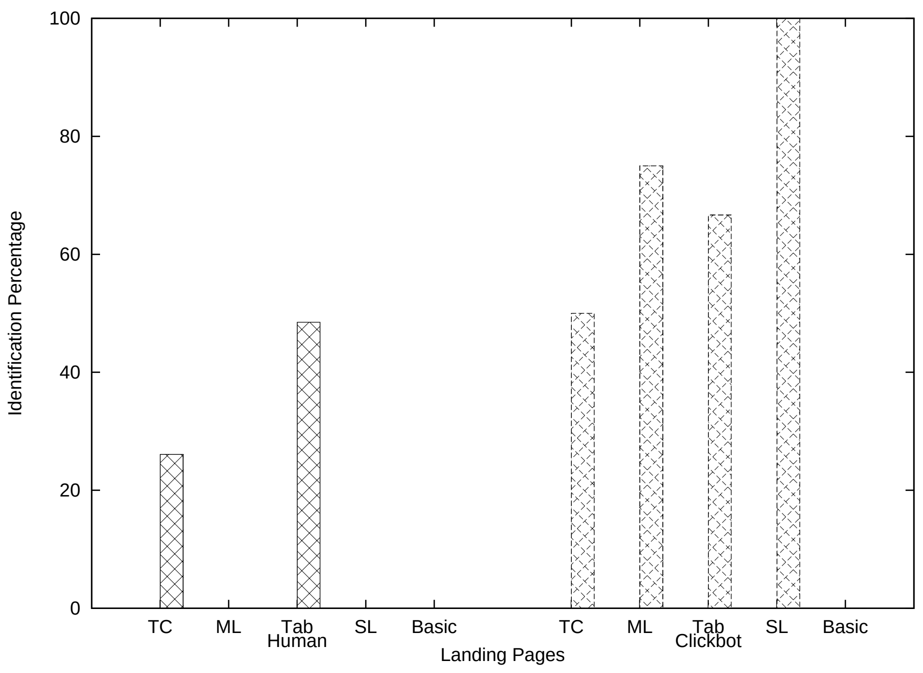


Figure 4.1: Results of preliminary human and robot study. Shows how often humans and robots were detected robots based on the technique used for the landing page.

the time, respectively. This means the visual differences of the valid links in the Tag Clouds and Tabs implementations were not enough to be a guaranteed guide to human users.

The Multiple Links technique had no false identifications while the Shadow Links technique had one false identification. We were able to determine that the false identification on the Shadow Links technique was because of human error and not the technique. Thus, we can say that the Shadow Link or Multiple Link technique are the best to use on a real website.

4.2 Comparison with CAPTCHA

CAPTCHA and DLDA are similar in the sense that the human user and web robot are presented with a challenge that is easy for a human to solve but hard for the robot. There are many different types of CAPTCHA available; however, the most common types involve the users typing letters appearing in an image, selecting an image based on a cue (e.g. a "select the cat"), or solving some kind of logic problem (e.g. 2 + 2 equals ?). DLDA provides similar challenge asking the user to select the valid link among many invalid as rendered in the browser.*Graphic for Each type of CAPTCHA*

The most similar CAPTCHA variety when compared to DLDA is cue based CAPTCHAs. Both DLDA and the cue CAPTCHA create a set of choices, DLDA with links and cue CAPTCHA with images. Each then gives the user a cue so that they will select the correct choice. When user makes choice, it is evaluated at the server with a correct choice allowing an action for CAPTCHA and not marking the link click as a bot for DLDA.

There are, however, very important differences between DLDA and CAPTCHA; where, when, and how often they can be used. These differences are owed to how the two methods are implemented, CAPTCHA with images and DLDA with links. This difference alone allows us a more unintrusive implementation.

CAPTCHA's original design implementation was to stop web robots from adding data to websites that accept user generated content. The interaction model for users who added information to a website was to visit a page with a form and fill in fields with information and then send it back to the server. CAPTCHA took advantage of this model by having web page authors add another field to their forms that asked the users to enter the alpha-numeric characters they saw in a picture. This worked because at the time bots were unable to read pictures and easily identify the characters contained within.

The interaction model having to enter things into a form and send it back to the server causes a problem for CAPTCHA. It makes the normal workflow for browsing a web page cumbersome. If website asked a user to solve a CAPTCHA every time they clicked a link, they would likely be able to detect every web robot that came to their site, but they would alienate every one of their legitimate human users. This prevents CAPTCHA from being used for things like *click fraud*. It necessary to come up with a CAPTCHA like mechanism that does not break the normal browsing workflow; this mechanism we have called *Decoy Link Design Adaptation(DLDA).*

The reason DLDA does not cause the same aforementioned problems as CAPTCHA has to do with design. DLDA was designed with the regular website workflow in mind. The most typical thing a user will do a website is to click links; as such, DLDA should be made to work on links within a website. It is this design choice that allows us to put DLDA on every page of a website and not alienate the human user population.

4.3 Implementation

We have described the basic principles behind DLDA, but now we need put those principles into practice through an implementation. There are two ways DLDA can put into practice: modification of a website by a developer or automatic modification by a program. Each method has advantages and disadvantages.

Modification of a website by a developer allows a wider range of linking grouping methodologies to be used when DLDA is applied to a website. The developer in concert with a usability expert can discern correct placement of invalid links such that they will look like a normal part of the website. However, the placement of the invalid links, guided by the usability expert, will have human website visitors clicking the valid links on the website.

The downfall with modification by a developer is the extra development time and cost. When a developer is modifying a website he or she must decide how to make the invalid links appear on the page most likely with the help of a usability expert. This will increase the time to develop the website and likely the cost.

Automatic modification by a program has the advantage in that it can applied to an existing website without modifying the website itself. This means development time and cost will be the same as if the website was not going to be protected.

The disadvantages of the automatic modification are a limited set of applicable link grouping techniques and added time for modification. Automatic modification requires that the applied link grouping techniques does not change the site look for feel and cause undue navigation burden to a human user. This restrictions for automatic modification are satisfied by both Shadow Links and Multiple Links. Added time for modification is only a problem if the web pages are modified as they served.

The overall advantage lies with the automatic implementation for its ease of deployment and cost savings. Because of this, our implementation automatically modifies web pages with a link grouping strategy. This makes testing easier on a variety of real world websites without a significant time investment.

4.3.1 Apache Server Module

Having chosen to use an automatic application of a link grouping strategy, we need to decide how the automatic application should be done. There are three ways to implement the automatic application: a compiler like program, a proxy, or a web server module. Each has specific advantages and disadvantages to how it affects the web page coding and the web page as it is served.

The advantage of a compiler like program is the time to server a page. Since the program will modify the code of the page to output the Shadow Links or Multiple Links technique as it is building the page, the web page has no extra overhead when generated and served to the user.

A compiler like program for automatically applying Shadow Links or Multiple Links has a great many difficulties for application. The biggest problem is the shear number of programming languages a website can be written in. The compiler like program would have to parse these languages and locate the HTML that is being output. Another problem with modifying a web pages generated from a programming language is that the links can be built across multiple code statements. This makes it very hard to rewrite the code to implement the link grouping technique.

An implementation of automatic application via a proxy is a more palatable option. The proxy has the advantage of being able to handle web pages written in any language because the page is modified as it is being served to the user. It can also work with any web server that is currently available or will be available as long as they both use the same HTTP protocol.

A proxy implementation has a key disadvantage when compared to a compiler like program. The proxy will need to inspect every incoming HTTP request and every outgoing HTTP response to see if the contents require modification. This is a problem because it means inspecting requests and responses that aren't even being considered for modification leading to an increased overhead on all requests and responses. This problem can be alleviated by putting the files that will get onto a separate server from those that won't, but this then increases the infrastructure purchase and maintenance costs.

The best way automatic implementation is through a web server module. A web server module has the same main strength as a proxy based implementation in that it can modify a web page produced in any sever-side language. It also removes the proxy disadvantage. A web server module can be made to operate on only certain served files from the server. This means it will never introduce overhead except to the files that are meant to be modified.

The disadvantage of a web server module is that it typically is tied to the implementation of a single web server. This means that each server to be supported by the implementation needs a separate module. Not surprisingly, however, most of the code for modification of web page is not server dependent. Thus, only the glue code between for registering the module and server callbacks need to re-written.

With the disadvantages of the compiler like implementation and the proxy, we decided it was best to implement DLDA as an web server module. Particularly, we implemented the module for the Apache web server. The Apache web server serves web pages at over 66% of the top one million trafficked websites. Thus, an Apache module implementation is applicable to a great majority of websites.

4.3.2 Shadow Links

We chose to implement the Shadow Links technique over the Multiple Links technique. Our choice was motivated by similarity of implementation. In order for Multiple Links to be effective, it will have to adopt the invalid link placement strategy of Shadow Links. This is because it easy for a bot writer to determine which links are invalid when they are all grouped to the same location and don't appear on the page.

Choosing to use the Shadow Links technique is not necessarily enough, we must make sure that the valid and invalid links cannot be distinguished when viewed in the HTML source. If invalid links do not appear valid when viewed in the source, there is a potentially easy way for a bot writer to avoid the invalid links. The valid link can also not appear in the same location among the valid links consistently. If this occurs, the a bot writer will know where the valid link appears and can easily choose the valid link when they visit the website.

There are two necessary requirements so that valid and invalid links cannot be distinguished when viewed in the HTML source: 1) similar URLs and 2) same URL text. To elaborate, the similar URL requirement means that when the URLs are viewed both the valid and invalid URLs are similar enough that one cannot distinguish between the invalid and valid URL; while, the same URL text means that a group of links with invalid links and one valid link share the same text between the anchor tag.

The last necessity to foil the bot writer is to randomly output the valid link within the invalid links. This means when the links are grouped together the bot writer must guess where the link is within the group. This prevents them from programming the valid link position into their bot and defeats bots based on upon direct selection criteria like XPATH.

With these constraints, we created a Shadow Links module for the Apache Web server. The module when loaded into the Apache web server will modify web pages to incorporate the Shadow Links technique. The module will also check all links coming into the web server to see if they are in fact invalid links that were output by the module.

Shadow Links Module

The Shadow Links module must work within the constraints of the Apache Module system [8]. Particularly, Shadow Links must use and Filter and Translate Name functions. The Filter function allows the Shadow Links module to re-write the HTML before it is output by the server, and the Log function allows the module to view and check the URL used in the request.

The Shadow Links Filter function modifies the page by parsing the resultant HTML and adding a variable number of invalid links to every valid link within the HTML. The valid link is placed randomly within it's group of decoy links. Thus, the module must somehow move the link to the top of the invalid links when the page is displayed in a browser.

To move the valid link to the top, the module outputs obfuscated Javascript. The obfuscation in the script prevents bots from determining which links on the page are valid or invalid, and provides the unobtrusive user experience that was seen when first testing Shadow Links. The current implementation of the module outputs a templated Javascript, but the module could be modified to output rotating Javascript.

The invalid links output by the module are randomly chosen by a file supplied by the user of the module. This allows the module user to configure invalid links that cannot be easily distinguished from valid links. An upgrade to this module could use real links from the website that are not reachable from the page being modified.

The Log function is called by the server whenever a request is made. The request information is passed to the Shadow Links log function which checks the requested URL. If the URL is one of the invalid links. A log message is written stating that an invalid URL was clicked and the request is allowed to complete.

4.4 Detection Success

This section presents our evaluation of DLDA's 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 [6]. 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 [1] 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 [14] 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 instructions to visit the website, 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.

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

4.4.1 Cross Comparison of Bot Detection Accuracy

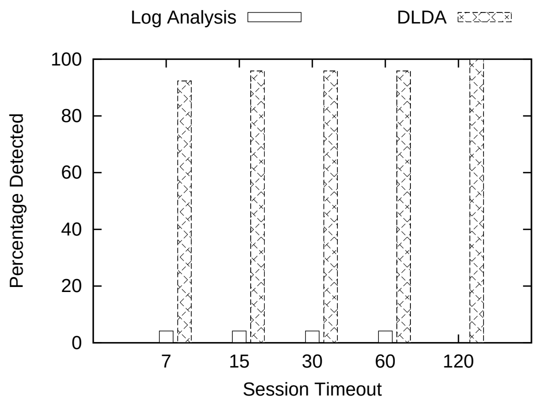


Figure 4.2: Comparison of DLDA to Log Analysis of Bomhardt et al.

We compared DLDA to a technique for detecting web robots based on statistical, passive analysis of sessions by Bomhardt et al. [9]. 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 4.2 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.

4.4.2 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 [4]. 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 back end database on each request: the main index page and two different product

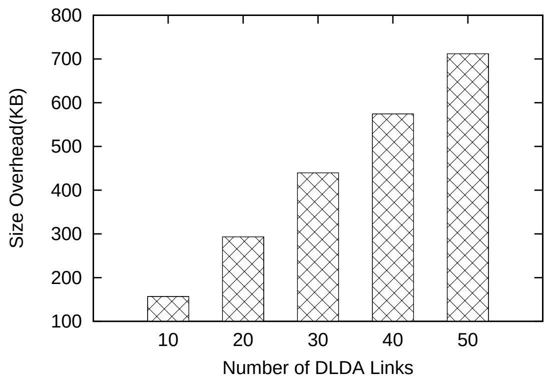


Figure 4.3: Overhead in bytes added by a different numbers of DLDA links.

DLDA adds data to the page when it modifies the links making the page larger. Figure 4.3 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 trade off 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.

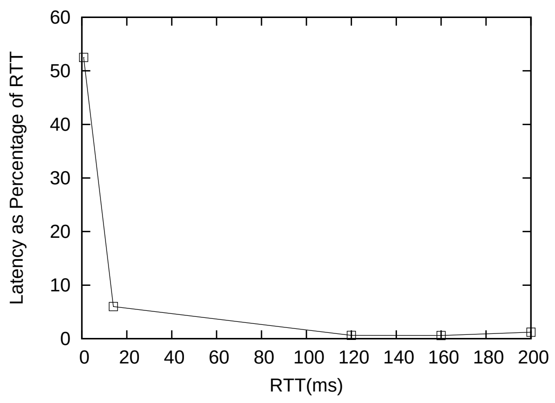


Figure 4.4: Overhead of modules as percentage of round trip time.

The latency overhead was fairly constant causing the latency to be become less noticeable as the RTT increased. Figure 4.4 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.

4.4.3 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.

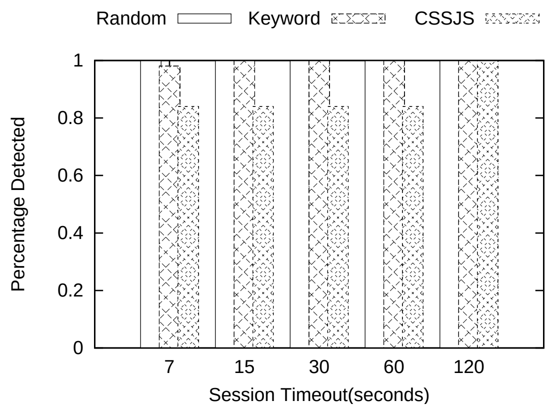


Figure 4.5: Detection rate of DLDA as the maximum time allowed between requests to included in the same session increases.

Figure 4.5 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.

The number of links output by DLDA was varied to see the affect on web bot detection. Figure 4.6 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

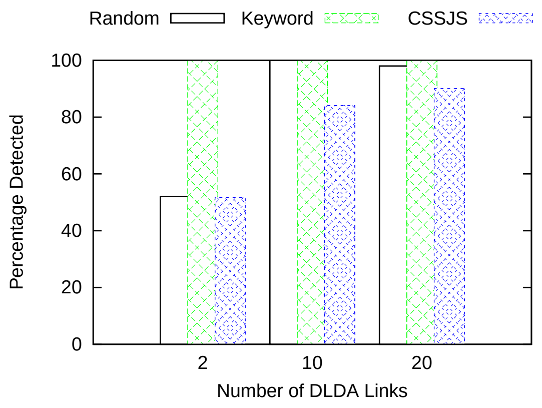
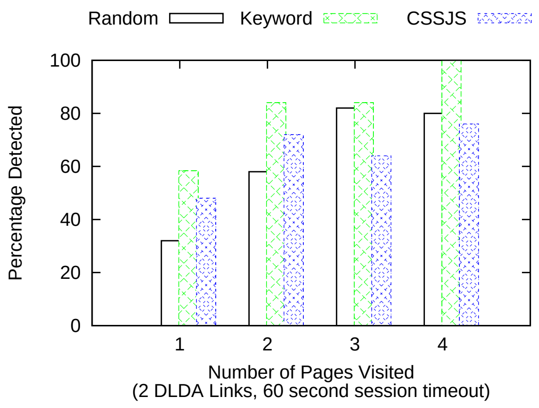


Figure 4.6: Detection rate of DLDA as the number of links output is increases. 2 is the absolute minimum number of links. More links than 20 did not significantly increase detection.

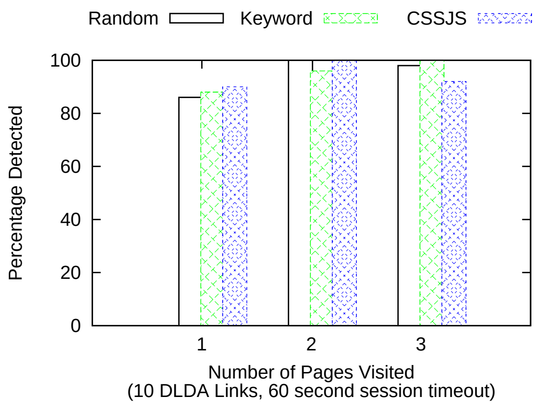
output links with only a slight increase at 20 links and beyond. The size overhead added by DLDA seen in Figure 4.3 means that one must make a trade off about the number of links to use. Given the size and detection results, 10 links seems to give the best overhead/detection trade off.

4.4.4 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 4.7 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 4.7(a) and Figure 4.7(b). The difference is very dramatic with an immediate 20-50% difference in detection in clicking a link from one protected page. This poses the overhead/detection trade off. 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 trade off.



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



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

Figure 4.7: Increasing detection of web robots as they navigate an increasing number of protected pages with 2 and 10 output links.

We can see variations in the detection ability of DLDA in Figure 4.7. 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.

Chapter 5

Transient Links

Transient Links is the counter part to DLDA designed to catch the replay bots that DLDA will not. Replay bots are arguably the easiest type of web robot to write because they need not parse the HTML they receive. They only need to send HTTP requests to a web server with the data they want and check the response codes.

Central to the idea of Transient Links and detecting replays is the concept of a single-use URL. This is a URL that can only be used one time. If the URL is used more than once, it would be considered invalid and result in the user receiving an error saying the resource pointed to by the URL does not exist.

The replay of a single-use URL can be detected because of the nature of the URL. It is the detection of the replayed single-use URLs that provides for the detection of replay bots. Detecting replay bots in this fashion is unlikely to cause a false detection with a human. Humans when visiting a website will start a dedicated URL and subsequently follow links provided on the page. This means that they don't replay any URLs because the starting URL must not be a single-use URL.

5.1 The Transient Links Technique

Transient Links is roughly based on the idea of generating random links for each page on a website. This gives a large number of URLs to each single page on a website. Thus, URLs may change over time allowing one to prevent the re-use of URLs associated with replay bots.

Transient Links improves on Random URLs that change over time by tying a random URL to a visit-session creating single-use URLs. A visit-session is the time period in which a user is actively navigating a website. With URLs tied to a visit-session, the URL is changed or made invalid for the user when they abandon their current session.

The abandonment of a visit-session is based on a timeout. The visit-session is abandoned when the user has not visited a web page within the timeout. At this point, the visit-session is destroyed, and when the user returns, a new visit-session is created and new URLs output based on the new visit-session.

The association of a URL with a visit-session allows Transient Links to detect when a replay of the URL occurs. The URL is associated with information specific to the user that created the visit-session. Information of interest includes IP address, starting URL, and a visit-session identifier. This information can be compared to determine if the URL was used outside the visit-session that created it.

Transient Links detects web robots when they replay a single-use URL from another user's visit-session. Transient Links compares the visit-session identifier assigned to the bot and that assigned to the single-use URL and sees they are not the same. This difference allows Transient Links to determine that a replay has occurred. Thus, a replay robot can be detected and stopped. However, to not detect a human, entry to the website must be limited to special entry points.

To prevent the web server from needing a large table mapping random URLs to real URLs and visit-sessions, information is stored in the URL. The single-use URL contains the visit-session identifier and the original page URL. Users and bots are prevented from reading the information stored by encrypting it within the URL.

5.2 Challenges

Transient Links modifies individual links of a website changing the URLs to single-use, random versions in the process. This could potentially impact important web applications and legitimate human actions on the Web. The central challenge in designing a concrete implementation to realize the technique is to ensure that the impact on other applications and use cases is minimal. We briefly discuss these issues below.

5.2.1 Human Link Reuse

People commonly reuse previously visited links. Browser bookmarks and sending links via emails (which would later be clicked on by other human users) are common examples of humans reusing links. This reuse requires that same links be allowed to be used more than once and beyond one visit-session contradicting the single-use principle of Transient Links.

5.2.2 Search Engines

With the tremendous growth of the World Wide Web, the importance of search engines as a mechanism of locating relevant information can hardly be overemphasized. In many of the popular search engines like Google, the hyperlink structure of the Web forms an important factor in ranking query search results. As Transient Links alters the links on a website, website administrators are rightfully concerned whether they would adversely impact the ranking of their websites. In fact, Transient Links can affect the two major functions of search engines namely, indexing website through crawling and ranking the pages of the website. The problems arise essentially because search engines use URI equality [19] to determine identicalness of web documents. This demonstrates that Transient Links must allow URI equality metrics to continue to work for search engines.

5.2.3 Analytics

Many website administrators employ analytics software for understanding the behavior of their users, judging the success of a particular feature or web page design, and for deciding content/advertisement placement. Most analytics software uses information such as the IP Address, URL, Referrer URL, and UserAgent string. Analytics engines can be based on either log analysis or page tagging. The log analysis engines get their data from the log files of web servers like Apache, and the page tagging engines send back information to a web server through javascript. Either type of analysis engine relies on URL equality metrics for their statistics. We must carefully implement single-use URLs such that URL equality is maintained so analytics software can provide correct statistics.

5.2.4 Caching

Caching is very important to the scalability of the World Wide Web. Companies like Akamai [1] provide sophisticated caching services to accelerate their client's web sites. They rely on keeping content closer to their users allowing for faster downloads.

Employing Transient Links implies that the links of a web page are made distinct for each session in which the page is served. Thus, the pages referred to by Transient Links become uncacheable. However, most heavy weight objects like pictures, flash, and movies tend to be embedded in normal HTML pages and will not suffer from caching problems. In cases where they are links on a page, Transient Links should leave them if configured to do so.

5.2.5 Visit Session Mapping

Transient Links maps each URL to a visit-session created when a human or web bot visits the website. The naive way to implement such a mapping is using a table that allows the look up of the visit-session by the URL. This is prohibitive in a web environment with a large number of people visiting the website causing the table to become to large causing it and the web server to be swapped to and from disk.

5.3 Overcoming Challenges

All the challenges mentioned occur because Transient Links creates single-use links that are only valid for a single session. This is a necessity for detecting web robots, but breaks the common use cases seen on the web today. In order to make Transient Links workable, we must find ways to have single-use links, but still allow the links to work in much the same way as today.

5.3.1 Allowing Human Link Reuse

We relax the single-use property of Transient Links to allow a new visit-session to begin with a URL from another visit-session. Transient Link URLs now start a new visit-session when they are used outside their creating visit-session. This allows the common workflow of copy a URL from the address bar and paste into an email, a web page, or an instant messenger, as well as, enabling bookmarking of Transient Links.

This relaxation means that Transient Links can no longer detect web robot replays on the first URL. Detection now occurs when the web robot replays multiple URLs from a previous visit-session.

5.3.2 Successful Search Engine Crawling

Search engines when they visit a website are doing two separate things: crawling and ranking. A search engine bot crawling requires URL equality to know when it has a visited a page before; this can be accomplished by identifying the crawler and disabling Transient Links. Ranking a web page requires that a search engine know a unique URL, and Transient Links lets a search engine know the unique URL with a canonical link.

Allowing search engines to crawl a website successfully requires that we turn off the randomness feature of Transient Links or turning Transient Links off altogether. This means that URL equality metrics will hold for the crawler. This allows the crawler navigate the website. This can be accomplished through useragent sniffing which has problems, IP address matching which is much more reliable, and in the case of Google, reverse DNS.

Allowing ranking requires the search engine to know that all the different Transient Link URLs point to the same page. This is accomplished through the use of the *<link rel="canonical" href="http://www.example.com">*. This tells the search engine that the

URL it is visiting now is the same as the URL in the link tag. Thus, the search engine can know which page to attribute page rank and give the best position in it's index [22]. The fact that a "rel canonical" link is embedded in a page does not allow a replay robot to break Transient Links. The URLs displayed in the "rel canonical" are only seen after a visit to the page and can be disabled for use through a mechanism in Transient Links.

5.3.3 Analytics

Analytics programs either work over the web server logs, called log analysis, or by sending information back to a server via javascript on the web page, called page tagging. Transient Links must work with either variant in a simple manner. However, the disparate manner in which they work means that one solution will not work with both types of analytics programs.

In order for Transient Links to work with log analysis, the real URL must be recorded in the log. This means the Transient Link URL must be translated into the real URL either before it is logged or by post-processing the log. It would be preferred if the real URL was recorded to the log in the first place, thus an implementation that changes the logged URL in the web server is preferred.

The best solution for a page tagging implementation is to replace the use of "document.location" with the "rel=canonical" link tag href attribute. The analytics javascript will search for the link tag and extract the href using the Document Object Model (DOM) of the web browser. Sending back this href attribute it now has the real URL for the page.

5.3.4 Enabling Caching

Caching of web pages is major issue with Transient Links. Modifying the links on a page every time it is served will cause failure in serving cached content. This is not necessarily a desirable outcome especially for pages that take a considerable amount of time to download. Transient Links must step aside for caching at the appropriate times.

Transient Links does not re-write the URL of images, flash objects, sound, or videos embedded within a web page. Those that are linked through the page with an anchor tag can be re-written but typically are not. This design decision means that the largest components of a web page will be cached as they always have been. However, this does nothing for dynamic pages that may have a large creation time.

Dynamic pages that require a lot of running time are usually cached on the server side. Transient Links will not interfere with this type of cache, as it is only concerned with the links in the final output, and this type of caching typically only caches the data needed to render the dynamic page. Should the final HTML output need to cached at the server, Transient Links must operate on the cached final HTML.

Finally, should a web page's HTML be large enough that caching it close to the user is desired, Transient Links can be used in a selective fashion. The selective application of Transient Links means that some links are protected but others are not. When applied in a selective fashion, it is important that all paths through the website include URLs protected by Transient Links. Without this requirement, a replay bot could visit the website without ever having to use a Transient Link URL thereby avoiding detection.

5.3.5 Visit Sessions in URLs

Storing visit-session ids in the URLs allows a quick mapping to the correct visit-session without need the a look up table. Storing the visit-session id in the URL presents its own challenge of how to prevent the user from changing the id. The answer to this is encrypting the the visit-session id and the real URL together. This prevents the URL from working if it is tampered with.

5.4 Implementation

Transient Links must use an automatic implementation because Transient Links must be randomized each time a web page is requested. This precludes the manual inclusion of Tran-

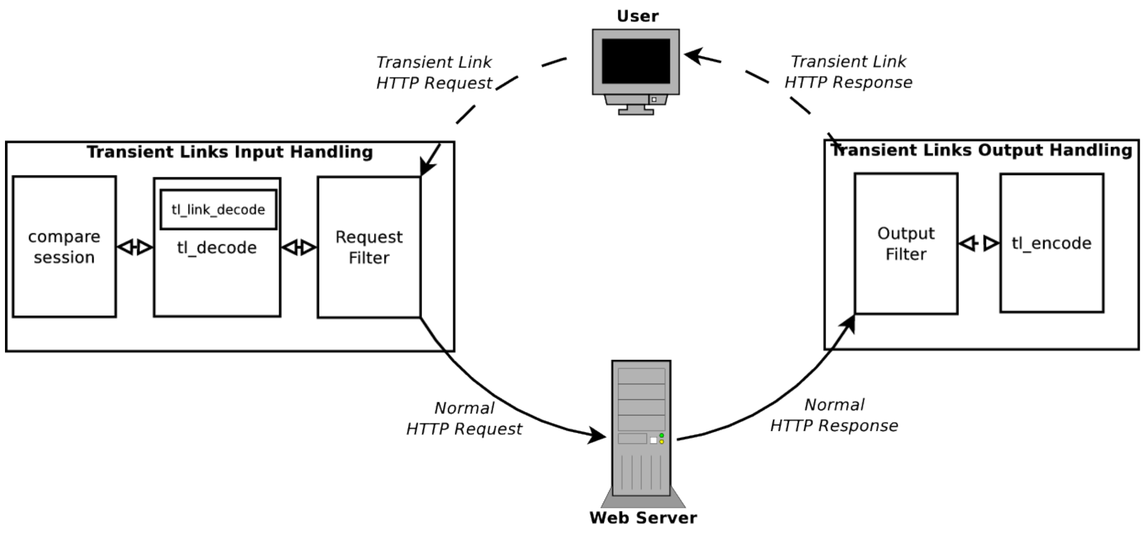


Figure 5.1: Transient Links algorithm. HTTP Requests are decoded for the server and responses are encoded for the user.

sient Links in a web page by a developer. A compiler-like automatic implementation also will not work because of the similarities between it and manual inclusion (e.g. The inability of a compiler-like implementation to randomize a page for each request).

The automatic implementation must work in real-time as requests are made and responses sent. This leaves two possible implementations the proxy or a web server module. Which is better is debatable, but we chose a web server for ease of implementation and for re-use of code with DLDA.

Any implementation of Transient Links must include at least two stages the URL encoding and the URL decoding of the single-use URL, as seen in Figure 5.1. The encoding of the URL replaces the every original URL in an HTML web page with a single-use URL. The URL decoding will read the single-use URL used to make a request and decode it back to original URL.

As described in the challenges section, Transient Links cannot use a truly single-use URL. In our case, single-use URLs are tied to the notion of a visit-session. A single-use URL belongs to a visit-session which is started every time a user visits a website with Transient Links. This session information is used keep all the necessary abilities discussed in the Challenges section while still detecting the replay of a URL.

The use of a visit-session requires a third stage outside the automatic implementation of Transient Links: a log analysis step. In this step another automatic program works to analyze the logs produced from a Transient Links protected website. It is this a log analysis that enables the detection of a web bot that is replaying URLs.

5.4.1 URL Encoding

```
input  : A request object request and the final HTML output output
output: The modified HTML
begin
    if $request.response\_code == HTTP\_REDIRECT$ then
        $location \leftarrow request.headers["Location"]$;
        $location.path \leftarrow tl\_encode(location.path, request.session)$;
        $request.headers["Location"] \leftarrow location$;
    end
    foreach $anchor$ in $output$ do
        $anchor.href \leftarrow tl\_encode(anchor.href, request.session)$;
    end
    foreach $form$ in $output$ do
        $form.action \leftarrow tl\_encode(form.action, request.session)$;
    end
end
```

Algorithm 1: OutputFilter

When the web server generates the final HTML output to be sent to the requesting client, our module is invoked using the OutputFilter, shown in Algorithm 1. This module does two things: 1) it will create a single-use URL for the Location header link if the response is a HTTP redirect, and 2) it will modify all the "href" attributes in the anchor tags and the "action" attributes in form tags. All the links are modified using the tl_encode function.

Algorithm 2 shows the tl_encode function which creates transient URLs for the URLs on a web page. This encoding function first combines the real URL, the session ID, and the

```
input  : The actual URL of an href to encode $real\_url$ and the Transient Link session $session$
output : The encoded URL
begin
    $iv \leftarrow byte[16]$;
    for $i = 0; i < 16; i++$ do
        $iv[i] \leftarrow random\_byte()$;
    end
    $decrypted \leftarrow combine\_data(session.id, session.start\_url, real\_url)$;
    $encrypted \leftarrow aes\_cbc\_encrypt(iv, decrypted)$;
    return $url\_base64encode(strcat(iv, encrypted))$
end
```

Algorithm 2: tl_encode

starting URL; the session ID is a unique id that identifies a stored session, and the starting URL is the first URL requested at the protected web site. This information is then AES encrypted with a configurable server key[26]. Finally, it is base64 encoded, correcting for characters not allowed within URLs.

5.4.2 URL Decoding

With Transient Links now output in HTML, the web server needs to have a way to translate back to the original URL. This is accomplished with the RequestFilter, shown in Algorithm 3. The RequestFilter takes the Transient Link URL and replaces it with the real URL that was encoded using the tl_decode function. It is also responsible for creating the Transient Link session, with help form the tl_decode function, that is associated with the current request.

The tl_link_decode function is the inverse of the tl_encode function, Algorithm 4. In the most straightforward case, it will do exactly the opposite of tl_encode. tl_link_decode will base64 decode the URL, decrypt it, and finally extract the session id, starting URL, and the real URL. The session id and starting URL are put into the Transient Link session associated with the request.

```
input  : A request object $request$
output: The modified $request$ with transient link session and real document url
begin
    $uri \leftarrow request.uri$;
    $session \leftarrow new\ tl\_session()$;
    $session.ip \leftarrow request.remote\_ip$;
    $uri.path, session = tl\_decode(uri.path, session)$;
    $request.session \leftarrow session$;
end
```

Algorithm 3: RequestFilter

```
input  : Transient Link URL $url$ and Transient Link Session $session$
output: Actual URL $real\_url$ and filled in session $session$
begin
    $was\_relative \leftarrow was\_relative\_url(url)$;
    if $was\_relative$ then
        $relative\_part, url = split\_url(url)$;
    end
    $decoded \leftarrow url\_base64decode(url)$;
    $iv \leftarrow substr(decoded, 0, 16)$;
    $encrypted \leftarrow substr(decoded, 16)$;
    $decrypted \leftarrow aes\_cbc\_decrypt(iv, encrypted)$;
    $session.id, session.start\_url, real\_url = split\_data(decrypted)$;
    if $was\_relative$ then
        $real\_url \leftarrow url\_resolve(relative\_part, real\_url)$;
    end
    return $real\_url, session$
end
```

Algorithm 4: tl_link_decode

```
input  : Transient Link encoded URL $url$, Transient Link cookie $cookie$, and
         Transient Link Session $session$
output: Document's actual URL $real\_url$ and Possibly modified session $session$
begin
    $cookie\_session = session.clone()\ real\_url, session = tl\_link\_decode(url, session)$;
    $tmp, cookie\_session = tl\_link\_decode(cookie, cookie\_session)$;
    $session \leftarrow compare\_session(session, cookie\_session, real\_url)$;
    return $real\_url, session$
end
```

Algorithm 5: tl_decode

tl_link_decode must also deal with relative URLs that were encoded using tl_encode. It determines if the URL was relative URL using the fact that a forward slash will appear in the middle of the URL. It then will split the URL separating the prepended part and the encoded URL. The encoded URL will always appear after the last forward slash. Finally, after doing the decode steps described previously, it will resolve real URL against the prepended part generating a fully resolved URL.

```
input  : A transient link session $session$ and the current document's real URL
         $real\_url$
output: Possibly a new transient link session
begin
    $stored\_session \leftarrow get\_stored\_session(session.id)$;
    if $stored\_session.ip\ != session.ip$ or $stored\_session.start\_url\ != session.start\_url$
    then
        $session \leftarrow new\ tl\_session()$;
        $session.ip \leftarrow session.ip$;
        $session.start\_url \leftarrow real\_url$;
        $session.id \leftarrow new\_session\_id()$;
        $store\_session(session)$;
    end
    return $session$
end
```

Algorithm 6: compare_session

The most important component of the entire decoding process is the compare_session function, shown in Algorithm 6. This function decides whether a link was used outside its session. If it was, a completely new visit-session is made for the request. This new visit-session is then used through out request and to encode all the URLs in the OutputFilter. It is also at this point the session.id will be added to the web server log.

5.4.3 Transient Link Log Analysis

Log analysis for Transient Links allows for the detection of replay robots. The problems with a naive implementation of Transient Links discussed in the challenges section require that single-use URLs be allowed to be used more than once, specifically when the user comes back to visit a website. This means the reuse of a URL cannot automatically mark a web robot.

The log analysis analyzes information output by the Transient Links web server module. Each time a single-use URL is used in a request, the session identifier is matched possibly creating a new one if the session is invalid. The session identifier returned after matching either the same session it was still valid or the new session identifier is put in the Apache log.

A robot replaying URLs will send URL that have expired visit-session ids, and as discussed in URL decoding, the log lines for the web robots requests will each have unique session ids. This is the key that allows log analysis to detect that a robot has replayed URLs.

The log analysis program, Algorithm 7, works by first sessionizing the web server log. This is not the same as the visit-session that was discussed; it is the creation of log sessions. Log sessions are created by grouping requests from IP addresses based on a time between the requests. The log sessions themselves are then analyzed for the replay activity.

The log analysis program creates log-sessions. Log-sessions are based on the sessions discussed in the Introduction section on Sessions. Requests are still grouped by IP, but a sub-grouping on the visit-session identifier is also used. Thus, a log-session can consist of multiple visit-sessions as deliminated by visit-session identifier.

The log analysis program then looks for log-sessions that have multiple instances of single-request visit-sessions based on the visit-session identifier. Any log session with this characteristic would caused by a replaying URLs. The reason for using multiple instances of single-request visit-sessions is because a human could come to a website and visit just a single page. This final analysis is then output into another log; it is at this point someone can make the determination of a replay bot or not.

```
input  : Apache log file
output: Detected Bot Log-Sessions
begin
    $fp \leftarrow open(log\_file, 'r')$;
    $logsessions \leftarrow map()$;
    $times \leftarrow map()$;
    foreach $line$ in $fp$ do
        $parsed \leftarrow parse\_log\_line(line)$;
        $ip = parsed[IP]$;
        $id = parsed[SESSION\_ID]$;
        $current\_time \leftarrow parsed[TIME]$;
        $time\_diff \leftarrow current\_time - times[ip]$;
        if $time\_diff >=$ 30min then
            $single\_visit \leftarrow 1$;
            foreach $session$ in $logsession[ip]$ do
                $single\_visit += 1$;
            end
            if $single\_visit >= CUTOFF$ then
                $print('BOT')$;
            end
            else
                $print('HUMAN')$;
            end
            $logsessions[ip] \leftarrow map()$;
        end
        else
            $logsessions[ip][id] += 1$;
            $times[ip] = current\_time$;
        end
    end
end
```

Algorithm 7: analyze_log

5.5 Experimental Methodology

Transient Links is aimed at stopping malicious web robot actions on websites. Testing Transient Links requires that a website be attacked by replay bots. Transient Links can then be evaluated on how well it can detect the attacking web robot.

5.5.1 Emulated Website

In order to test Transient Links, we need to have a website that is similar to what one encounters in daily browsing of the web. We would also like a testing website whose counter part real websites tend to see fraud in many different forms. An e-commerce website is commonly used by everyone and their proprietors can experience everything from click fraud on ads to unfair product buying with web robots.

A well known and commonly used e-commerce store is osCommerce [6]. It has many corporate sponsors including Paypal and Amazon Payments [27]. Besides having well known corporate backing, osCommerce is also used by big names like Google for Google-Store and Youtube-Store, and it is also used for many small web commerce shops selling a myriad of different items.

5.5.2 Replay Robots

With Transient Links goal to detect replay bots, we needed to test against different approaches to a replay bot. A replay bot is defined as recording the URLs requested and information sent to the server and resending that information. A replay bot can either resend this information perfectly mimicking the website visit or with variations in time and information.

We were unable to find a replay bot that could vary the website visit on it's own, so we wrote one with the ability. We called it the Modifying Bot. This bot takes a list of URLs and replays them in a similar manner to the human website visit. It is free to change the inter-request timing within a fixed random delay/speedup and can use different information for a single recorded replay.

Transient Links also needed to be tested against an available replay robot. We chose to test CARENA because of its web browser integration and free availability. CARENA hooks into the Firefox web browser and records all requests made. It is able to distinguish the requests made by the human driving the browser and those by the browser itself. The nice thing about browser integration with CARENA is that it will request all embedded items on the page, but it will replay the website visit with the exact timing as before.

5.5.3 Metrics

Transient Links is meant to detect web robots, but it does so by modifying a website in real time. Thus, the effectiveness as well as the overhead introduced must be measured.

Effectiveness

Transient Links effectiveness can easily be tested when it is applied to a whole website. When applied to a whole website only an "entry URL" is available in unmodified form. Thus, every other link clicked on the website will be a Transient Link. However, as mentioned earlier in the challenges section, this may not work for all websites and a selective application may be desired.

Testing Transient Links effectiveness with a partially modified website tells us how likely a replay bot is to be detected with different interleaving of Transient and non-Transient Links. Transient Links is most effective when applied to every page on a website. Modification affects the ability of intermediate servers and web browsers to cache web pages. We need to see how well Transient Links can perform with a smaller portion of a website modified.

Overhead

Our Transient Links implementation is based upon real time page modification. Modifying the HTML of a web page could take a significant amount of time for large web pages or pages with a significant number of links. It is important that our implementation be fast in all cases.

In measuring the overhead of Transient Links, it is important that we only measure the overhead added by web page modification. This first requires that we measure the speed of the pages without any modification to get a baseline. The osCommerce store is a dynamic website and incurs overhead in producing each page. Using this base line measurement we can then take the difference of this and the time for the web site with Transient Links to get only the page modification time.

Caching

Caching was the only challenge that did not have a clear answer to alleviate problems caused by Transient Links. Transient Links by construction only modifies the anchor tags in HTML making HTML uncacheable. We looked at the Alexa Top 5 to see how much this affected caching. We tested how much of a web page download was HTML versus CSS, JS, and images, as the smaller a portion of the download HTML the less effect Transient Links has on caching.

5.6 Experiments

This section presents our evaluation of replay bot detection. 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 [6]. osCommerce is one of the most popular open source implementation of an e-commerce web site. To study the effectiveness of bot detection, we built a replay robot. The replay robot was fairly simple and sent requests based on links it was given in a script. We believe this represents a typical replay robot.

To mimic realistic traffic to a web site, we also mixed the bot traffic with accesses from human volunteers. The evaluation spanned over 2 days with participation from 10 human users.

We specifically studied the accuracy of detecting replay bots with the Transit Link method. We also look at the overhead introduced by Transient Links, in terms of the increased size of delay introduced to users.

5.6.1 Detection Accuracy

Transient Links is meant to differentiate replay web robots and human website visits. It detects replay bots as they replay URLs causing multiple one URL visit-sessions to be created. We tested replay detection running different replay robots over the osCommerce Online Merchant website.

We set the log analysis of Transient Links to allow a single one-request visit-session. Typically, when analyzing a log, a one-request visit-session would have been caused by a web crawler like Googlebot or a human visiting a single page. The latter is an important case because Transient Links must never detect a human as bot. Allowing the a single one-request visit-session adds a third detection status to Transient Links of Unknown, as a bot or human could have caused this.

For detection, we looked at the percentage of URLs protected by Transient Links, varied from 25% to 100%, and the number of pages visited within the replayed session. This allows us to show the likelihood for detection based on the selective application of Transient Links.

Creating the sessions for testing the selective application of Transient Links required a slight modification of our module to randomly modify a given percentage of the website. Since we could not do this exactly, we emulated this by giving each link a possibility for being a Transient Link. For a site with only 25% Transient Links, we gave each link a a 25% chance of modification when it was encountered.

To select the actual sessions for testing, we randomly selected URLs from the pages of the partially protected osCommerce website. We did this selection five times for each percentage of the site modified. The Modifying Replay Bot and CARENA then ran these sessions over the protected website.

Full Website Modification

We first tested both CARENA and the Modifying Replay Bot against the osCommerce test site with all links modified to Transient Links. We did not need to create as many sessions as for the selective application because each session would contain only Transient URLs. To that end, we created two sessions and played them through CARENA and the Modifying Replay Bot. Each bot ran each of the two recorded sessions ten times creating a total of twenty sessions for each.

With Transient Links enabled across the entire site, we were able to detect 100% of the replayed web visits. CARENA and the Modifying Replay Bot did not escape detection because they caused Transient Links to create new visit-sessions for every URL they replayed. This showed up in the log analysis as multiple single-visit visit-sessions. Thus, we can say Transient Links is very effective when applied fully. However, we need to test Transient Links when it is selectively applied.

Selective Website Modification

We can see in Figure 5.2 that the Modifying Replay Bot was detected with between 40% and 100% of it's replay visits to the website with 25% and 75% modification, respectively. We can see that the percentage of links modified on a website has an important bearing on detection. As one would expect, the more modified links on the website more likely a

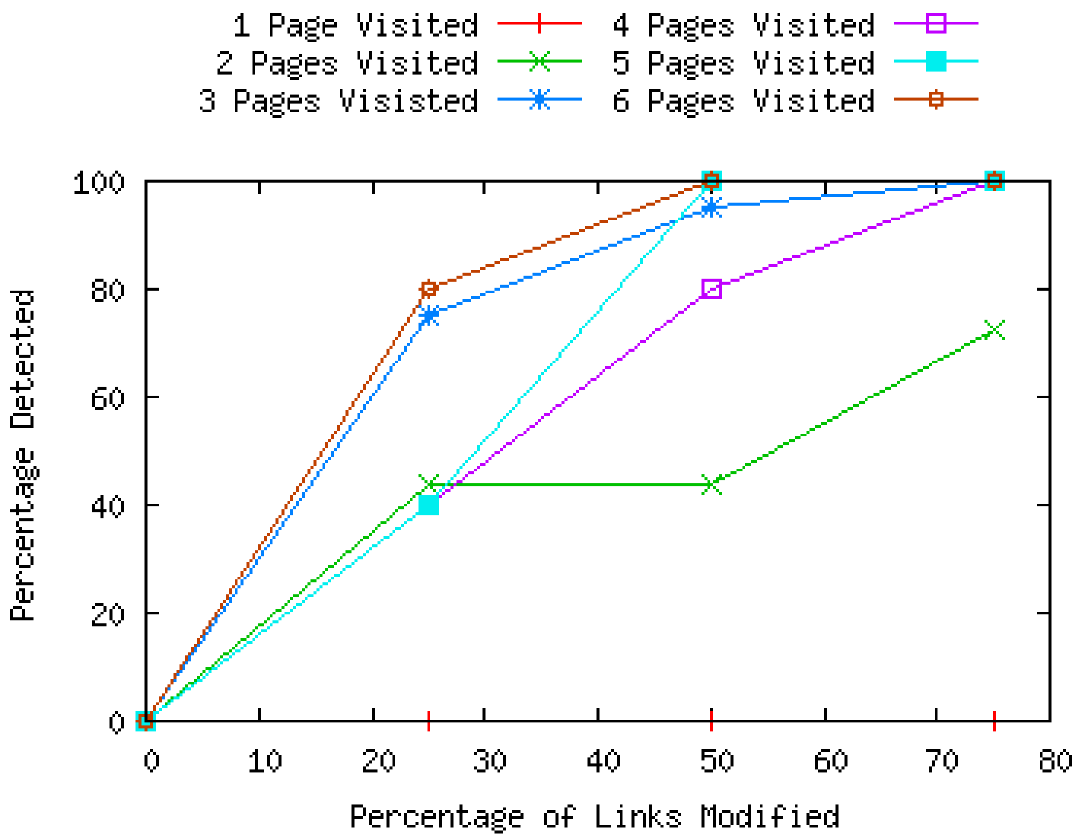


Figure 5.2: Detection of the Modifying replay bot with 0% to 75% of the URLs as Transient Links

detection is to occur. The intuition behind this is that more single-use links are likely to appear in a session with more links to visit.

One would expect that the greater the number of links the greater detection rate. This holds for all but 3 pages and for 5 pages at 25%. This has to do with where a Transient Link appears in a replayed web visit and the number of Transient Links. For three pages, when the link appears in between the two other links, the bot will be detected, but when it appears at the beginning or the end, the bot is missed because we allow a single one-visit visit-session. Allowing this single one-visit visit-session also explains the detection at 5 pages.

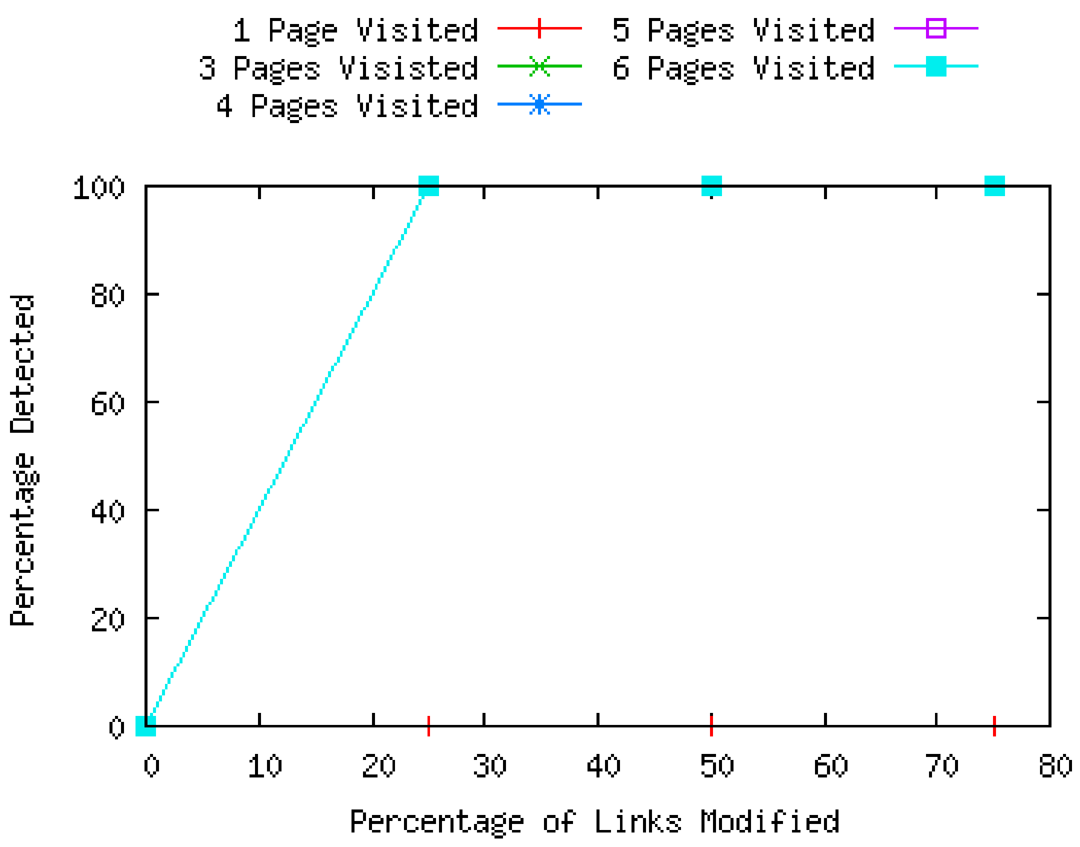


Figure 5.3: Detection of CARENA with 0% to 75% of URLs as Transient Links.

At 25% modification much of the web visits only saw a single Transient Link. This shows that two Transient Links must occur in sequence.

Figure 5.3 shows that CARENA had much greater detection rate than the Modifying Replay Bot. It is detected 100% of the time across all the selective modifications. The difference in detection is due to CARENA replaying exactly all the headers that were sent the first time. Transient Links will set a cookie that can be used to track the state when a visit-session contains normal links. CARENA was unable to handle this; the Modifying Replay Bot was and shows the lower detection rate because of it.

Human Testing

To test for human false positives in Transient Links, we contacted 34 users and asked them to visit our protected osCommerce store. We gave them no specific instruction save that they should perform an action on the website rather than merely visiting the page. We also did not say the website was protected by a our Transient Links module.

Investigating the human created sessions on the protected osCommerce website showed no false positives. This is not unexpected because the users did not replay links when they visited.

5.6.2 Module Overhead

We study the overhead of Transient Links by measuring the delay experienced by users. We varied the AES keys size parameter with Transient Links testing 128, 192, and 256 bits for its effect on Transient Links' overhead.

We used Apache JMeter to test the protected website [4]. JMeter was originally developed to test Apache's heavily used Tomcat Application Server and is well suited to testing dynamically created content. JMeter was setup to request three pages 400 times each. Three pages were requested that had dynamic content and read from the back end database on each request: the main index page and two different product category pages.

Figure 5.4 shows the overhead of Transient Links is consistently a 3ms. The different key sizes had a constant output time of 64ms to the output time of 61ms for no modification. Thus, varying the key size of the AES encryption did not change the overall time for a web page to be modified.

5.6.3 Caching

To test percentage of a website that remains cacheable, we selected the Top 5 sites from Alexa in the month of October 2010. We had a crawler view the main pages and go one

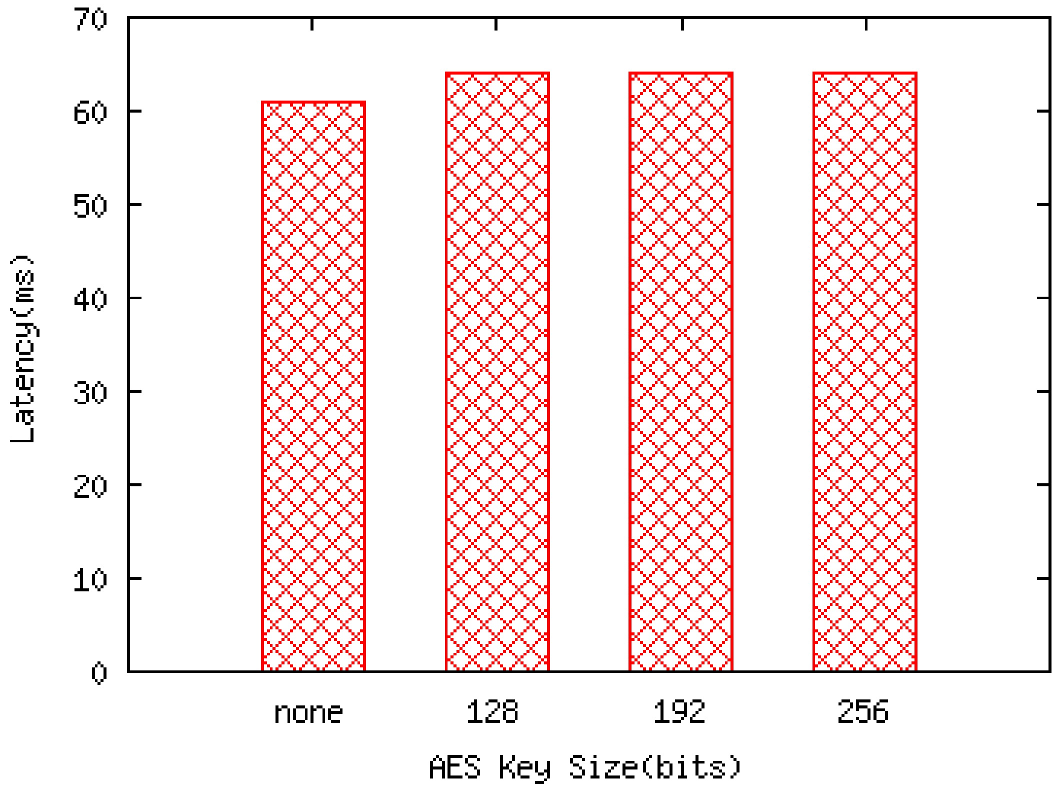


Figure 5.4: Latency introduced by Transient Links varying the AES Key size. None is where the Transient Links module was not present.

level deep. That is follow any links from the main page. The crawler extracted size data as it went, and aggregated that data to create the percent of a website that remains cacheable.

Table 5.1 shows that most websites still have 68% to 94% of their size cacheable. This is significant. It shows Transient Links is not a major problem when caching web data and can be applied with only minor consideration. As always, if certain pages of HTML must be cached, Transient Links can be disabled on those pages.

Table 5.1: Percentage of Alexa Top 5 Cacheable (size in bytes)

URL	HTML Size	Javascript Size	CSS Size	Image Size	Cacheable
facebook	29345	202529	67330	189804	93.99
google	46229	6285	6159	89970	68.89
login.live.com	11338	185184	7333	0	94.43
yahoo	67656	183933	94662	815789	94.17
youtube	130549	265921	230902	1066216	92.29

5.7 Discussion

While Transient Links works well at detecting replay bots, 100% of replay bots tested; it is not infallible. It has limitations on it's ability to detect replay bots, and it is possible that a hybrid replay bot may defeat Transient Links. Here, we discuss the limitations and why the hybrid replay bots are not a threat to Transient Links.

5.7.1 Transient Link Limitations

Replays of the single-use URL from the computer that caused its creation may not be registered as a replay even when run through a bot. This is because of the relaxation in the Transient Links definition of single-use. Humans must be able to share the links that they are given by Transient Links without triggering bot detection. This condition only persists as long as the visit-session that is associated with the single-use URL is valid on the server. Thus, an absolute timeout on the length of a Transient Link visit-session can limit this problem.

Another limitation due to the relaxed definition of single-use is the requirement of at least two replayed Transient Link URLs for detection. This is not necessarily a problem with many useful workflows requiring more than one link click. If we ignored the need for sharing Transient Link URLs, it would be possible to do detection with only one URL replayed.

5.7.2 Link Ordering

As discussed in the Selective Modification experiments, the order Transient Link URLs appear in a web site visit can affect detection. Upon investigation this was because of our detection for the allowed one-request visit-session. The answer to this is to only allow the log-session created in log analysis to consist of only the one web page visit for the one-visit visit-session detection. In this way, order of Transient Link appearance no longer affects detection. However, this could lead to an increase in human detection.

5.7.3 Hybrid Replay Bots

In the experiments we saw that Transient Links was able to detect all the traditional replay bots. Traditional replay bots will, given a set of URLs, request the URLs in the order in which they were given. A hybrid replay bot can side-step the detection of Transient Links when used against a protected website. A hybrid replay bot works by replaying a session using something other than a URL like a link keyword or image or an XPath location. An example of this type of replay bot is IRobot [20], where the bot works by recording the XPath location of each URL on the web pages visited. Since this bot will click the newly created single-use links, it will avoid detection as a replay bot.

While these are replay bots in that they replay a previously recorded website visit, they are more closely related to walking web robots. In our paper on detecting walking bots[11], we described a method of defeating walking bots that is applicable here. With IRobot using XPath to select the link in the page, it will be defeated with our walking bot detection which changes the location of the valid link every time the web page is visited. Similarly other types of hybrid replay bots will be detected by our walking bot detection.

Chapter 6

Conclusion

The bot detection methods presented have shown that our contributions to detecting and stopping fraud with web robots is practical. Enhanced Log Analysis provided a method to uniquely identify web robots, and DLDA and Transient Links provide the ability to detect sophisiticated human like web robots. All of the methods introduced are unobtrusive and easily avoided by humans. Finally, DLDA and Transient Links provide a way to detect web robots at every web request.

Enhanced log analysis provides a methodology to passively observe the behavior of web robots that visit a web site. It does this by viewing logs including the traditional web server access log and a traffic trace log. This allows for new behavioral attributes to be added to past work and for unique identification of web robots. The unique identification of web robots is of much interest because it means bots can no longer lie about their identity (e.g. a spam bot claiming to be googlebot).

Enhanced log analysis creates signatures for detection and bot matching. Signatures are vectors of attributes created from sessions (groupings of web requests). The attributes used in signatures expand based on the type of log. Access logs can produce attributes about time between requests and the number of requests resulting in page not found errors. Header logs allows us useragent sniffing providing a way to detect robots that lie in their "User-Agent" header. Finally, traffic traces provide us information about the type of OS and network behavior of a web robot. This is useful in identifying of a bot is lying about its OS in the "User-Agent" header.

Enhanced log analysis was evaluated on its ability to detect and uniquely identify web bots. Detection was done with both a heuristic approach and a machine learning approach. The heuristic approach was able to detect XRumer and EmailScraper 100% of the time but could only detect IRobot 90% of time. The decision tree machine learning algorithm was able to improve on the overall detection. It was able to increase detection of IRobot to 100% of the time, but the detection of XRumer fell to only 98% of the time.

The signatures created from enhanced log analysis allowed for the distinguishing of distinct types of web robots in traffic traces. Using K-Means clustering, we were able to generate distinct groupings for each tested web robot: XRumer, EmailScraper, and IRobot. Upon introducing noise into the clustering process, the K-Means algorithm was still able to generate unique clusters for the we bots. It also did so in a convincing fashion keeping much of the noise outside of the clusters for those bots. The grouping for IRobot contained only 23% noise, XRumer only 39%, EmailScraper only 2% noise.

Enhanced log analysis has proven it's ability to detect bots and even group them according to type. But, it is possible that bots may get better and mimic humans to a degree that even it cannot detect. In this case, another way is needed to detect web bots that does not rely on the behavioral information they send to us. The way to do this is force them to differentiate themselves from humans.

Forced behavior provided by DLDA and Transient Links is an important step forward in differentiating human and bot traffic. CAPTCHA is loosing effectiveness as bots get better at solving them, and they become harder for humans to solve. DLDA and Transient Links in concert provide a new methodology that allows for the effective separation of human and web bot accesses. As an added advantage, they can be applied to the entirety of the website providing a larger surface area for detecting web robots.

DLDA and Transient Links are complimentary approaches to detecting web robots. Web robots can be divided into two classes: walking bots and replay bots. Walking bots download and use the HTML source of a website to navigate; where as, replay bots are given URLs and data to send to the website and do not view the source. DLDA was designed to stop walking bots by obfuscating real links on a web page with decoy links; Transient Links detects replay bots with single-use links that cannot be used outside their initial website visit. When DLDA and Transient Links are used together, the website cannot be visited by a web robot without the robot being detected no matter how the web robot tries to escape detection.

DLDA was tested for its ability to detect walking web robots while avoiding human false positives. The testing methodology involved varying the number of decoy links output and varying the depth to which the web bots crawled. Since DLDA adds extra links to a web page we also tested it for size and time overhead added to a page.

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 dramatic 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.

DLDA's overhead was minimal in time delay adding only 30ms to a page generation when a dynamic osCommerce page was requested. However, the size for a page could increase dramatically as much as 130kilobytes for only 10 DLDA links output. The reason for this in our testing was because of osCommerce which appears to be a degenerate case. osCommerce 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.

Transient Links, the counter part to DLDA's walking bot detection, has a myriad of a different challenges to overcome. These challenges include bookmarking, search engines, web site analytics, and caching and stem from the implementation. Keeping web robots from replaying a previous human website visit requires that links be unique per website visit. This is similar to having a random link for each page each time the website is visited. Bookmarking is an obvious challenge because it is a human replay. The other challenges stem from the need for the "random like" links for each page; search engines and caching require that a URL be unique per page.

Transient Links is able to overcome most of these challenges caching, however, cannot be fully solved in the software. To enable bookmarking, Transient Links allows for a new website visit to be started with a link from another, different website visit. This places a restriction on Transient Links ability to detect web robots with only one request to a website. The problems for search engines is solved with the output of the "rel=canonical" link which lets the search engine know the preferred URL for each page. Page tagging analytics has the same solution as the search engine problems, but the log based kind have no issue as Transient Links must decode the requested URL before the web server is able to log it. Finally, caching problems are not easily solved, but Transient Link is tested for its ability to leave most caching intact.

Transient Links has a differing detection abilities depending on selective application. It was tested for detecting web robots based on how much a website was protected, and it was tested to show that it would not falsely detect humans as web robots. Since Transient Links changes the URLs on a website, it was tested for the time delay added to a web page.

Transient Links proved that it could detect replay robots with high accuracy as much as 100% when applied to the whole website and 40% to 75% when selectively applied. Selective application has an expected increase in detection as more of the website was protected. Transient Links did not falsely detect any of the 34 human testers as a bot. This shows Transient Links is effective at detecting bots while avoiding falsely detecting humans. Transient Links proved extremely usable with the module only adding 3ms overhead to a completely dynamic page. With the implementation relying on dynamic modification it could be used easily on any website with the Apache web server.

Bibliography

[1] Akamai. http://www.akamai.com/.

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

[3] Google adsense. https://www.google.com/adsense/.

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

[5] Nmap. http://nmap.org/.

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

[7] Usage share of web browsers. http://en.wikipedia.org/wiki/Usage_share_of_web_browsers.

[8] Apache Software Foundation. *Filters*. http://httpd.apache.org/docs/2.0/filter.html.

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

[10] Botmaster Labs. Xrumer. http://www.botmasternet.com/.

[11] Brewer, D., Li, K., Ramaswamy, L., and Pu, C. Link obfuscation service to detect webbots. In *SCC 2010* (2010).

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

[13] Click Forensics. Click fraud index. http://www.clickforensics.com/resources/click-fraud-index.html.

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

[15] Elson, J., Douceur, J. R., Howell, J., and Saul, J. Asirra: A captcha that exploits interest-aligned manual image categorization. In *In Proceedings of 14th ACM Conference on Computer and Communications Security (CCS)* (2007), Association for Computing Machinery, Inc.

[16] Golle, P. Machine learning attacks against the asirra captcha. In *In Proceedings of 15th ACM Conference on Computer and Communications Security (CCS)* (2008), Association for Computing Machinery, Inc.

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

[18] Hachamovitch, D. Html5, modernized: Fourth ie9 platform preview available for developers. http://blogs.msdn.com/b/ie/archive/2010/08/04/html5-modernized-fourth-ie9-platform-preview-available-for-developers.aspx.

[19] IETF Network Working Group. Hyper text transfer protocol – http/1.1. http://www.ietf.org/rfc/rfc2616.txt.

[20] IRobotSoft. Irobotsoft visual web scraping & web automation. http://irobotsoft.com/.

[21] Juels, A., Stamm, S., and Jakobsson, M. Combating click fraud via premium clicks. In *In Proceedings of 16th USENIX Security Symposium* (2007), USENIX, pp. 17–26.

[22] Kupke, J., and Ohye, M. Specify your canonical. http://googlewebmastercentral.blogspot.com/2009/02/specify-your-canonical.html, 2009.

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

[24] Mozilla Foundation. Bug 187996 - strange behavior on 305 redirect. https://bugzilla.mozilla.org/show_bug.cgi?id=187996.

[25] National Center for Supercomputing Applications. Ncsa mosaic. http://www.ncsa.illinois.edu/Projects/mosaic.html.

[26] NIST. Specification for the advanced encryption standard (aes). *Federal Information and Processing Standards Publication 197* (2001).

[27] osCommerce. oscommerce: Corporate sponsers. http://www.oscommerce.com/partners/corporate.

[28] Peterson, E. T., Bayriamova, Z., Elliot, N., Fogg, I., Kanze, A., Matiesanu, C., and Peach, A. Measuring unique visitors: Addressing the dramatic decline in accuracy of cookie based measurement. http://www.forrester.com/rb/Research/measuring_unique_visitors/q/id/51116/t/2, 2005.

[29] Robinson, D., and Coar, K. The common gateway interface (cgi) version 1.1. http://tools.ietf.org/html/rfc3875.

[30] Silverman, D. Iab internet advertising revenue report. Tech. rep., Interactive Advertising Bureau, 2009.

[31] SoftPlus. Forum poster. http://fp.icontool.com/.

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

[33] The PHP Group. Php. http://www.php.net/.

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

[35] W3C HTML Working Group. Html5. http://www.w3.org/TR/html5/.

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