

# Protecting Web Contents against Persistent Crawlers

Shengye Wan

Nanchang City, Jiangxi Province, China

Bachelor of Engineering, Huazhong University of Science and Technology, 2014

A Thesis presented to the Graduate Faculty  
of the College of William and Mary in Candidacy for the Degree of  
Master of Science

Department of Computer Science

The College of William and Mary  
May, 2016



## APPROVAL PAGE

This Thesis is submitted in partial fulfillment of  
the requirements for the degree of

Master of Science



---

Shengye Wan

Approved by the Committee, May 2016



---

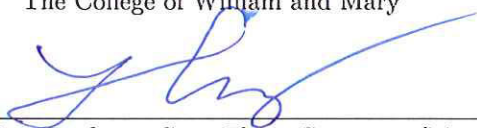
Committee Chair

Assistant Professor Kun Sun, Computer Science  
The College of William and Mary



---

Professor Qun Li, Computer Science  
The College of William and Mary



---

Associate Professor Gang Zhou, Computer Science  
The College of William and Mary

## ABSTRACT

Web crawlers have been developed for several malicious purposes like downloading server data without permission from website administrator. Armored stealthy crawlers are evolving against new anti-crawler mechanisms in the arms race between the crawler developers and crawler defenders.

In this thesis, we develop a new anti-crawler mechanism called PathMarker to detect and constrain crawlers that crawl content of servers stealthily and persistently. The basic idea is to add a marker to each web page URL and then encrypt the URL and marker. By using the URL path and user information contained in the marker as the new features of our detection modules, we could accurately detect stealthy crawlers even most distributed crawlers at the earliest stage. Besides effectively detecting crawlers, PathMarker can also dramatically suppress the efficiency of crawlers before they are detected by misleading the crawlers visiting same page's URL with different markers. We deploy our approach on a forum website to collect normal users' data. The evaluation results show that PathMarker can quickly capture all 12 open-source and in-house crawlers, plus two external crawlers (i.e., Googlebots and Yahoo Slurp).

# TABLE OF CONTENTS

Acknowledgments	iii
1 Introduction	2
2 Background	6
3 Threat Model and Our Goals	8
4 System Design	10
4.1 Website Modification . . . . .	10
4.1.1 Adding URL Marker . . . . .	10
4.1.2 Expanding Access Log . . . . .	11
4.2 Crawler Detection . . . . .	15
4.2.1 Heuristic Detection . . . . .	16
4.2.2 Machine Learning Detection . . . . .	17
5 Security Analysis	21
6 Implementation and Evaluation	24
6.1 PathMarker Prototype . . . . .	24
6.2 Normal User Study . . . . .	24
6.3 Performance Evaluation . . . . .	29
6.3.1 Detection Capabilities . . . . .	29
6.3.2 Suppressing Distributed Crawlers . . . . .	31
6.3.3 System Overhead . . . . .	34

6.3.4	Googlebots – A Case Study . . . . .	35
7	Limitation and Discussion	39
7.1	Usability of URL Encryption . . . . .	39
7.2	System’s Configuration . . . . .	40
7.3	Deployability of PathMarker . . . . .	40
7.4	Baiting Link . . . . .	41
8	Related Work	42
9	Conclusions	45

## ACKNOWLEDGMENTS

I would like to first thank God for being my strength to conduct the research. Without Him, I would not have had the wisdom or the physical ability to do so.

I express my gratitude to my advisor Dr. Kun Sun. Without his encouragement and support, I would not start this research and finish the thesis. Dr. Sun is always patient with me and gave me precious guidance whenever I ran into a trouble spot or had a question about my research or writing.

I would also like to acknowledge my dear friend Yue Li as the second reader of this thesis and I am appreciate his very valuable comments and suggestion on this thesis.

I would also like to thank all the faculty and staffs at the Computer Science Department especially my committee members Dr. Qun Li and Dr. Gang Zhou for their great support.

Finally, I must thank my parents and my grandmother for always being supportive of my education. My family members always support my academic decision and making me feel loved during the entire work of the thesis.

Protecting Web Contents against Persistent and Stealthy Crawlers



# Chapter 1

## Introduction

With the prosperity of Internet data scale, the demand of crawlers has dramatically increasing. The 2014 bot traffic report from Incapsula [20] points out that bots account for 56% of all website traffic, and malicious bots contributes almost one third of the web traffic. Besides the increasing of the total amount of crawlers, more and more algorithms [5, 11, 7, 22] are available for malicious crawlers too. As the result, it becomes more tough for the website administrator to detect the attacking crawlers now.

To detect and constrain malicious crawlers, researchers have developed a number of anti-crawler mechanisms [35, 38, 32, 13, 40, 26, 29, 16, 17, 21]. Among these works, heuristic detection mechanisms have been widely adopted to identify and defeat malicious crawlers through analyzing the User-Agent, referrer, and even cookie fields in the HTTP request headers as well as monitoring the visiting rate of each individual visitor. However, they cannot guarantee to detect crawlers that can manipulate those observed features. Alternatively, machine learning detection mechanisms can detect malicious crawlers based on the different visiting patterns of normal users and malicious crawlers. Many recent works use these two techniques together to defend the attackers.

Different mechanisms could protect websites from different crawlers because they use different features. Among those major targets of malicious crawlers, one type is the website that contains a limited number of confidential document or treasured content. The basic shelter of these websites is requesting users to login before accessing its most valued data. However, even only authorized or paid users are allowed to access those website pages, an insider attacker still has the chance to crawl the entire website and use them with malevolent purposes like selling them or phishing other normal users. For instance, one

malicious doctor may implement a crawler with his or her user account then the crawler could pass the authentication of the hospital’s server and download all patients’ document from the hospital’s internal network. After that, the doctor could sell it to medicine dealers.

Most insider attackers of the websites above crawl the precious contents stealthily and persistently. The crawlers are persistent means they can reasonably sacrifice the efficiency. Furthermore, The crawlers are stealthy so run extra work to better mimic the access behaviors of real users. Thus, it becomes a challenge to detect such a crawler accurately at its earliest stage. Moreover, several insiders may coordinate and use a divide-and-conquer strategy to increase the crawling speed. For most anti-crawler systems, when a malicious crawler operates by itself, the systems can easily block single crawler’s access to the web server using a network firewall. However, when a number of robots coordinate from different identities, it becomes more difficult to automatically block all of them. Because of these properties, the heuristic features or machine learning features of known anti-crawler mechanisms might not work well for detecting the crawlers that target on valuable content of websites. For instance, when a crawler sets a random time gap between two sequential requests, it can escape most anti-crawler mechanisms that work based on the time-related features.

In this paper, we develop a new anti-crawler system called PathMarker that aims to detect and constrain stealthy persistent crawlers targeting at websites that require user to login with valid user accounts for accessing a limited number of valuable content. The main idea is to add an marker to each web page URL and record the web page path and user information in the markers to help identify and confine crawlers. Given one website, PathMarker can automatically generate and append a marker to each web page’s URL. We call the marker as URL marker, which records the information about the page that contains this link and the user account who collects this link. With the aid of URL marker, we can filter many basic crawlers by checking the information in marker. Furthermore, we can build accurate URL visiting path for every user based on the URL marker and

then calculate the visiting depth and width of each single user. Next, we can leverage machine learning techniques to detect crawlers based on the different patterns of URL visiting paths and different URL visiting timings between human beings and malicious crawlers. We adopt Support Vector Machine (SVM) to model the normal users and crawlers using the features related to the URL marker. Finally, we use CAPTCHAs to verify any suspicious user in case our machine learning reports a false positive case.

To avoid the armored attackers forge or reuse the URL markers, we encrypt the URL marker along with the original URL except the domain name to further protect the website structure against crawlers. The crawlers will possibly collect different web links for the same web page because the URL markers of the same page might be different. Thus, we can suppress attacker’s crawling efficiency especially for the distributed crawlers because every user ID of the distributed crawlers would get different encrypted links for the same page. Moreover, when distributed crawlers share collected links in a pool, we can detect them through a user ID mismatch, since the user who collects the page may not be the same as the one who visits the URLs contained in this page.

We develop a PathMarker prototype on an online open source forum website. We set up the forum website and integrate our anti-crawler mechanism. We first train a SVM model based on the log data collected from more than 100 normal users and 6 in-house crawlers, and then test the model using 6 open-source crawlers and another set of normal user data. The experimental results show that our anti-crawler technique can effectively detect all mentioned crawlers. Moreover, two external crawlers, Googlebot [30] and Yahoo Slurp, are also detected. We unveil the distributed nature of Googlebot and study how PathMarker is able to suppress the efficiency of such crawlers.

In summary, we make the following contributions.

- We develop an anti-crawler system named PathMarker to detect persistent and stealthy web crawlers. It can differentiate normal users from malicious crawlers by using the URL visiting path and URL visiting timing features derived from the URL

marker. We can reduce the download speed of individual persistent crawler that mimics human being’s download behavior to the level of human beings.

- PathMarker is able to instantly detect distributed crawlers that share links with each other. If the distributed crawlers do not share links in a pool, our encrypted URL technique can effectively suppress their efficiency.
- We implement a PathMarker prototype on an online forum website and show that it is simple to deploy our mechanism on existing websites. The experimental results show that PathMarker is capable of detecting a number of state-of-the-art crawlers.

The remaining of the paper is organized as follows. Section 2 introduces background information. Section 3 presents our threat model. We present the architecture of PathMarker in Section 4 and detail the security analysis in Section 5. The system implementation and evaluation are shown in Section 6. We discuss interesting observations and potential extensions in Section 7. We discuss related works in Section 8. Finally, we conclude the paper in Section 9.

## Chapter 2

# Background

The web crawler starts by visiting a seed page, which is usually the homepage of the target website. By parsing the seed page the crawler collects URLs embedded in the page. Based on the crawling algorithm, the crawler picks next page to visit from the collected URLs [28]. We show a simple website structure example in Figure 2.1. The maximum depth of this web structure is 3 and the maximum width is 2. We show two sample paths both travelling through three pages but in different orders, where the visiting path A is *homepage*, *page1* and *page3* and the visiting path B is *homepage*, *page1*, and *page2*. Path A has depth of 3 and width of 1 and path B has depth of 2 and width of 2.

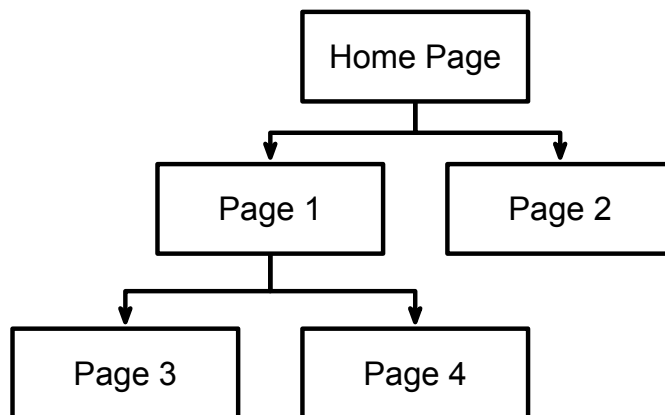


Figure 2.1: A Website Structure Example

The different crawling algorithms will have different depth and width of the visiting web paths for the same website. First, starting from the seed page, depth-first crawlers greedily visit the URLs as far as possible in a branch of a website tree graph before backtracking. In Figure 2.1, the depth-first crawler's visiting path is *homepage*, *page1*,

*page3*, *page4*, and *page2*. Alternatively, breadth-first crawlers try to visit all links of a page before visiting links from another page. In Figure 2.1, the breadth-first crawler's visiting path is *homepage*, *page1*, *page2*, *page3*, and *page4*. The backlink number-first crawlers aim to collect most valuable content of a server. The backlink number of a page represents the number of other pages that are linking to the page. Therefore, the backlink number reflects the popularity of a page. However, the algorithm require some knowledge about the target pages to compute the backlink number before running the crawler which means this algortihm does not work for crawling unknown website pages. Similar to the backlink number-first algorithm, PageRank-first crawlers also try to crawl more important web pages first, but they estimate the importance of a web page based on not only the number links linking to this page, but also the quality of these links. For the crawlers have no preference about depth or width like backlink number-first crawlers or PageRank-first crawlers, we would classify them as "Random-like" in our paper.

## Chapter 3

# Threat Model and Our Goals

We focus on the insiders that have valid user accounts of the targeted websites and attempt to download the entire web contents for profit. One persistent insider may crawl the entire website in a long time period and also the insider could be stealthy by mimicing the access behaviors of human beings. Moreover, a small number of insiders with valid user accounts may coordinate to acquire the whole content of the website.

Our work targets at detecting crawlers who have access to all or part of protected website contents. We assume each user may register one or multiple user accounts on the same website, but the number of total accounts controlled by one user is limited due to economic concerns (e.g., paying the premium), labor concern (repeating registration process), or identity constraints (real name or ID number required). The websites that allow anonymous web page requests are out the scope of this paper because most of the valuable content or confidential document will not be reviewed by everyone without restriction. Also, we do not consider the case that one attacker could control a large number of user accounts by cracking the website’s password database because in this case the attacker could use each user ID only few times and we have small chance to realize one user ID’s visiting behavior within few requests.

Our anti-crawler mechanism targets at achieving four important goals.

1. High Accuracy. PathMarker should provide an accurate detection that is able to detect and constrain most crawling activities with a low false positive rate. Moreover, it should also be able to detect distributed crawlers that may coordinate with each other to bypass the detection on individual crawlers.

2. Fast Detection. The system should ensure a rapid response, which means it can not only detect malicious crawlers, but also detect a suspicious crawler at its earliest stage in order to minimize the loss.
3. Low User Experience Degradation. A good defense mechanism should not affect user experience. Therefore, we need to guarantee the normal usage of websites under our threat model will not be affected apparently.
4. Delay the crawler those could hide from PathMarker. for the unknown crawler with really delicate algorithms who can impersonate human beings to bypass our detection, we should suppress their crawling efficiency to the level of normal human beings.



## Chapter 4

# System Design

PathMarker requires some modification on the protected website to provide more web page visiting information for crawler detection. With the website extension, we can develop an efficient crawler detection mechanism that is able to achieve all the four goals in Section 3.

### 4.1 Website Modification

PathMarker requires two major changes, namely, adding URL marker and expanding access log, on the target website system to help record more web page visiting information for both heuristic detection and machine learning based detection. The URL marker is used to track the visiting path of visitors by recording the parent page and detect distributed crawlers by matching the current visitor with the page obtainer in URL marker. As a definition in our work, if the user gets page 2s link from page then page 1 is the parent page of page 2. Basically the parent page of a link is the page that contains this link. Since the marker part is useful for PathMarker and the server receive every marker at the same time as handling the request, we modify the log function of the server for recording those markers correctly. We will discuss the details about the modifications in the next subsections.

#### 4.1.1 Adding URL Marker

We add a URL marker at the end of each hyperlink URL inside a page. Each URL marker records the page that the URL is retrieved from and the user who obtained the URL. By analyzing these information we are able to verify if a user is visiting a URL that is obtained

by other users through comparing their access logs. We can learn the causal relationship between two links from the URL markers and accurately determine the width and depth of the visiting path. If we do not use the URL marker, then we only could estimate the depth and width based on semantic analysis of every URL, this method takes much time and could not calculate the depth or width of a session accurately. Moreover, the URL marker can be used to effectively detect distributed crawlers that share a URL pool.

The URL marker is appended directly in the URL of each page so it is viewable to all visitors. To protect the information from leaking to the attacker or reusing by the attacker, we encrypt the path and URL marker on the server side. Here we use one key for all users and because we do not worry that different users might reuse others' ciphertext because our URL marker could help us detect this case. We encrypt the path as well since it can help constrain distributed crawlers by forcing them to repeatedly visit the same page that may have many different URL markers.

For example, a typical URL of the domain A is: A.com/B/C.html.

After we add the URL marker to it, it would be: A.com/B/C.html/mk:B/root.html;User1.

The appended URL marker is mk:B/root.html;User1 which means this URL is retrieved from the page A.com/B/root.html and "User1" is the user who obtains the URL.

The whole URL after encryption using AES-256-CBC is A.com/en:bf37cf8f8f6cb5f3924825013e3f79c04086d1e569a7891686fd7e3fa3818a8e.

In a static website, we only need to apply a simple server script function to modify all links in a web page. In a dynamic system, there is usually a URL generator function, which can be modified to add the URL marker.

#### 4.1.2 Expanding Access Log

Most websites maintain their access logs, which record the information of all visitors such as IP address, the page URL that is being visited, and timestamp. PathMarker extends the access log to record more information. When a new request is received, the server decrypts the encrypted URL and parses the plaintext to get the URL marker. Then the

access log records user ID, user IP, visiting URL, URL marker information, and timestamp. Table 4.1 shows an example on logs and how we calculate depth and width according to the logs.

Table 4.1: Example Logs for Computing Path Features

Log ID	Log Info					Analyzing Result				
	User ID	User IP	URL	Marker	timestamp	Short Session ID	Deepest page	MAX depth	Widest Page	MAX width
1	1	127.0.0.1	URL1	URL0; 1	0	1	URL1	1	URL0	0
2	1	127.0.0.1	URL2	URL1; 1	3	1	URL2	2	URL1	1
3	1	127.0.0.1	URL3	URL2; 1	8	1	URL3	3	URL1	1
4	1	127.0.0.1	URL2	URL3; 1	10	1	URL2	4	URL1	1
5	1	127.0.0.1	URL4	URL2; 1	15	1	URL4	5	URL2	2
6	1	127.0.0.1	URL5	URL2; 1	17	1	URL4	5	URL2	3
7	1	127.0.0.1	URL6	URL3; 3	20	1	URL4	5	URL2	3
8	1	127.0.0.1	URL7	URL1; 1	32	8	URL4	5	URL2	3

For PathMarker, we need to calculate the depth and width of a group of continuous logs belong to the same user and we call this group as a session. For each given session who contains some logs, we processing the session log by log. For each log, at first we need to check the information in the URL markers is correct. For example, the log 7 in table 4.1 shows user 1 visits a page with URL marker of user 3, this might happen if user 3 share the link to user 1. If user 1 visits links of other users more than the upper limit we define, then we set user 1 as a suspicious user.

If the user ID in the marker could match with the current visitor’s ID, then we check the log’s parent page that saved in the marker for calculating the depth and width. Here depth represents how deep a page is and width means how many subpages of one page are visited by the user. Basically if user retrieves the URL a from the page of URL b and he visits both pages and he visits URL b is ealier than visits URL a, we consider that URL a is deeper than URL b and URL b’s width should add 1 because URL a was visited.

1. If this log’s URLs parent page is not viewed within this session before this log then we set this log’s URL with depth 1 and width 0 such as the log 1 of table 4.1. Then, we skip to the next log.
2. If the URL’s parent page appears in the same session and being viewed before it, then we set this log’s URL’s depth = its parent page’s URL’s depth + 1. Similarly, If the

link's parent page appears in the session before it, we set its parent page's URL's width = its parent page's URL's width + 1. Note if one session contains a URL several times, we would keep adding the depth. For instance, in the 4.1, both log 2 and log 4 are generated for URL2 but the later log still add 1 depth for whole session.

After we check all logs of entire session, we set and the session's depth and width as the maximum depth and depth of all URLs being requested within this session. Finally we get the depth and width for this session. To better utilize the depth and width we get, In the meantime, PathMarker runs an individual monitoring process locally to monitor each newly added log. This process decides if and when to send a user to the machine learning module for further analysis. This monitoring process are working based on our new conceptions which we are discussing below.

The first conception is Short Session. We consider a user is in a short session if the user requests two pages within a couple of seconds. We use 10 seconds as the default interval time for our experiment since our user study shows that normal users generate two requests within 10 seconds among a single visiting period. The parameter can be tuned according to each website's specific user scenario. In Table 4.1, the short session ID means the first log ID for current short session and the log 8 starts a new short session since the interval time between log 7 and log 8 is larger than 10 seconds. As long as a new request is made after the 10 second time period, the following requests will be grouped in another short session. The length of a session is the number of log entries in the session. Therefore, the length of a short session varies depending on the visiting pattern of the user.

The other concepts is Long Session. PathMarker keeps track of each new log. If the number of new log of a visitor reaches the length of a long session, one alert will be raised and this long session will be sent to machine learning module for further analysis. The length of a long session is fixed. If the system set the length of long session is  $X$ , then for every user the first  $X$  logs would be the first long session,  $X + 1$  to  $2X$  logs would be the

second session so on. We define these two concepts because we discover that users have different pattern about depth and width in their short term and long term while crawlers always behave similarly and we would discuss the details of this observation in section 6.2. In this case we want to use the short session to present the pattern of every one's short term behavior and long session could represent the user's long term behavior. Since various website systems may have different average short session length and long session's length should be longer than this average value to present different pattern of the same user, we recommend to set the length of a long session as twice of the average length of all users' short sessions' lengths. For the rest of the paper, when we are talking about the short session, it represents the short session that only belongs to one long session. If a visitor keeps visiting pages within 10 seconds and his short session across from one long session to another, then we separate the logs as two short sessions.

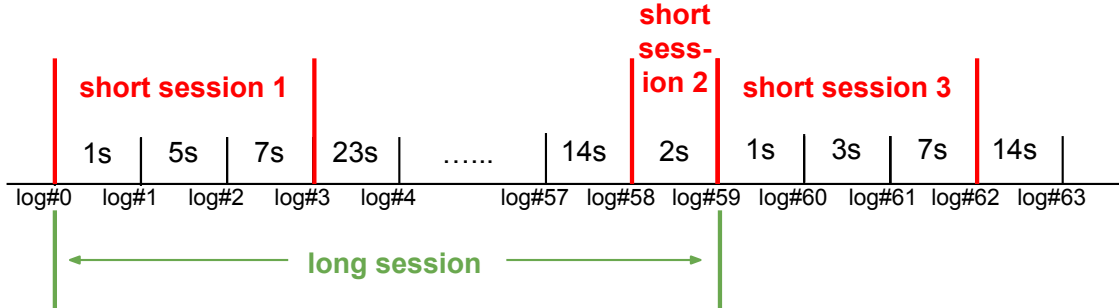


Figure 4.1: New Definitions about Session

To explain our conceptions, let's define the long session's length is 60 (we would discuss why we set this value in the section 6.2). Then we assume there is a new user whose first 64 logs' time gaps show as Figure 4.1. According to the figure, we could see in the figure, this user only finishes one long session which is log#0 to log#59 and he has three short sessions: log#0 to log#3, log#58 to log#59 and log#59 to log#62. Actually the users starts his second short session at log#58 and it could continue to log#62, however the first long session stops at log#59 so we have to separate these logs into two short sessions. Basically, every time when one user's new logs are more than a long session, the system

would use the machine learning module to analyze this user’s new long session. In the section 4.2.2 we would discuss how to utilize these two conceptions to analyse the visiting paths of different users.

## 4.2 Crawler Detection

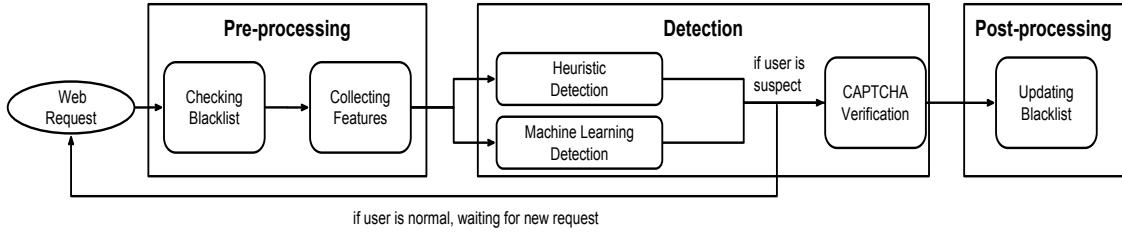


Figure 4.2: PathMarker Architecture

Figure 4.2 is the architecture of our system. According to the figure, PathMarker consists of three modules, namely, pre-processing module, detection module, and post-processing module. The entire defense mechanism works as follows. For incoming HTTP requests, the pre-processing module first recognizes the user IDs of requests. If the user ID is in the blacklist that keeps all the known crawlers’ IDs, then it directly rejects the request. For other users we would extract the URL markers from the original URLs and then save the information.

After the pre-processing stage, PathMarker is able to defend persistent crawlers by utilizing both heuristic detection and learning-based detection with the newly added path features. It first uses heuristic detection to investigate basic web traffic patterns as well as checks URL marker integrity. For instance, it will check if one user generates more than a threshold of web requests. PathMarker also leverages machine learning detection technique to identify suspicious users by learning both the visiting time features and visiting path features from the URL marker and the access log. Since the system keeps logging each user’s web access activities, the machine learning detection component will continuously analyze the visitor’s behaviors and generate an alert when one visitor is suspicious.

For both detection components, after a suspicious crawler is detected, it prompts CAPTCHA to further reduce the false positive rate. After failing to correctly input the CAPTCHA a couple of times, the user will be put into the blacklist. Otherwise, the system releases the alert and continues monitoring the logs of the visitor.

Furthermore, we have a final defense to handle the case that there is unknown expert crawler who could hide from our both layers of detection. We set a upper bound for each user if they visit a certain percentage of total account. Any user visit more than the percentage need to prove to the serve about his or her identity.

#### 4.2.1 Heuristic Detection

Heuristic detection module performs basic analysis on the incoming traffic and aims to discover crawlers based on basic traffic flow features. Since the crawler is usually much faster than a human user, many crawlers can be easily identified through analyzing the page visiting rate. PathMarker investigates the referrer, user agency, and cookies of all incoming traffic. If one or more of these fields in over 10 HTTP requests within an hour are abnormal (e.g. always no referrer, user agency is known bots' agency, or some cookies are missing), the user generating the requests will be labeled as a potential crawler.

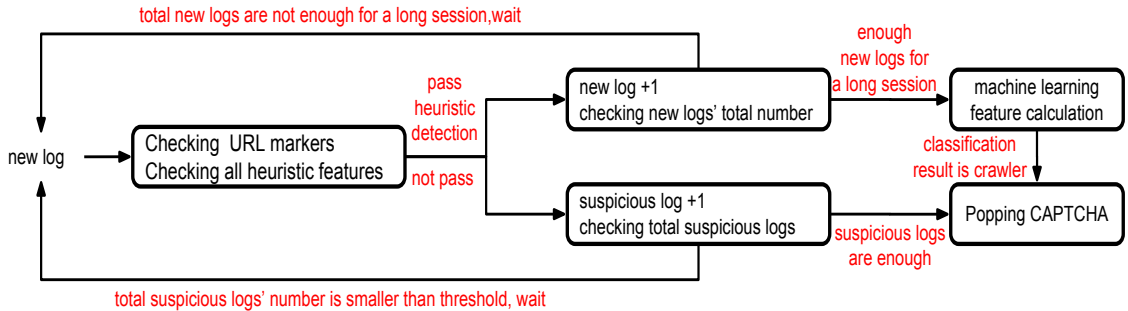


Figure 4.3: Heuristic Detection Working Flow

Besides these general features, this module is also responsible for URL marker integrity checking, which is a new heuristic detection feature we proposed. Specifically, when a new log is generated, the PathMarker first compares the URL of the page and the visitor ID

with the information recorded in the URL marker. If the visitor of this page is not the one recorded in the URL marker (who is the obtainer of the page URL), we flag this log entry and mark this user as a potential crawler that visits shared links obtained by other crawlers. If the user is flagged multiple times within a time period, we mark this user as a suspicious crawler and prompt it with a CAPTCHA. Basically, Figure 4.3 shows how the system handle with each new log the server records.

Though heuristic detection have been deployed on many web systems, it is still a challenge to accurately detect distributed crawlers that share the URLs for crawling. With the integration of URL marker, our heuristic detection module can detect distributed crawlers by examining the causal relationship among HTTP requests.

#### 4.2.2 Machine Learning Detection

We use machine learning technique to determine if an active user who has one or more long sessions is a normal user or a crawler. We adopt the supervised Support Vector Machine (SVM) as the learning model in PathMarker. We identify six features to train the SVM-based detection model, based on the fact that normal users and crawlers have large difference in visiting path pattern and timing.

1. The depth rate of long session  $\frac{\max(D_L)}{L_{Long}}$ .  $\max(D_L)$  represents the maximum visiting path depth in a long session and  $L_{Long}$  represents the fixed length of the long session. This feature describes how visitors keep reaching new pages as deep as possible in a long session. For example, if user 1 just finishes a long session whose length is 60 and we calculate the largest depth of this long session is 13, then this value would be  $\frac{13}{60}$ .
2. The width rate of long session  $\frac{\max(W_L)}{L_{Long}}$ .  $\max(W_L)$  represents the maximum visiting path width of a long session. This attribute is similar to the depth rate of long session, yet in the width dimension.
3. Time interval variation of long session  $\frac{Var(I_L)}{I_L^2}$ . Time interval is the time gap between two consecutive requests, which is represented as  $I$ . This feature is computed as the



variance of time interval in a long session over the square of the average time interval in this long session.

4. The absolute difference between depth rate of long session and depth rate of longest short session in the long session  $\left| \frac{\max(D_L)}{L_{Long}} - \frac{\max(D_S)}{L_{Short}} \right|$ . To compute this feature, we need at first find out the longest short session in one long session. Then we calculate the  $\max(D_S)$  which is the maximum visiting path depth in this longest short session and  $L_{Short}$  which is the longest short session's total length. This feature describes how is the depth pattern of one user's long session different from this user's pattern of the longest short session. One simple example is that when this feature is close to zero, we could see there is no difference while most normal user would have a relatively high depth rate for their short session.
5. The absolute difference between width rate of long session and width rate of longest short session in the long session  $\left| \frac{\max(W_L)}{L_{Long}} - \frac{\max(W_S)}{L_{Short}} \right|$ . This feature describes width pattern difference between the longest short session and a long session of a user.
6. Time interval variation of longest short session  $\frac{Var(I_S)}{I_S^2}$ . This feature is similar to time interval variation of long session; however, the time interval is computed based on the longest short session in the long session.

Feature 1, 2, 4, 5 are our new path-related features that present features in website visiting path. In a short session, human beings usually have more obvious pattern. Specifically, there are two common patterns when a user is viewing websites.

- i. The user may open multiple web pages at one time, so the maximum width of the user's visiting path could be as large as the length of the short session.
- ii. The user prefers to jump to another page after he or she takes a glance at one page, so it will present a large depth of short session.

However, for both cases of normal users, in a long session, the maximum depth and width of a user’s visiting path is likely to be much smaller than the length of a long session, since a long session may contain several short sessions (the length of a long session is roughly double as the length of average short session) and these short sessions are independent to each other in terms of depth and width. Meanwhile, crawlers usually have homogeneous patterns in visiting path. For example, a depth-first crawler would have both large depth rates of long session and short session, while a random-like crawler would have a small rate of depth and width. For all the crawlers we test, their behaviors are consistent when measuring the differences between short sessions and long sessions. Furthermore, even one crawler wants to mimic human being’s visiting pattern based on our features, it does not know the parameters because the default time interval for calculating short session and the length of long session are different for different websites. This is one reason why we suggest administrators set their parameters based on their servers’ logs. As the conclusion, unless an attacker could get the rough parameters or mimic human beings path-pattern well, even he or she sets the visiting with some delicated algorithms, we are still able to find them out( we would discuss more in Section 6).

Besides the path-related features, we also have two timing features 3 and 6. Normal users have a small variance of time interval of their short sessions while large variance about long sessions. For most crawlers, they visit web pages in a more regulated pace so the variance of time interval is very small compared to human visitors. Even for an armored crawler that adds random delay in its visiting pattern, it still can be easily detected since it does not produce different interval variance in a short session and a long session as human beings. Again, by combining timing features with our two new sessions, a crawler has to know the parameters about them for escaping our detecting mechanism. We use variance of time interval divided by the square of the average time interval as the features because it’s better to set all features of a SVM have similar ranges. Otherwise we need one more step to balance these features and that cost time and resource.

Basically, our features well describe such difference so the SVM model is able to

distinguish normal users from crawlers accurately. To get an idea result, the machine learning model should be trained using data from normal users and crawlers. System administrators may use some of or all crawlers available to crawl their own systems. Therefore, it is straightforward to collect data of crawlers. However, collecting normal user data is not easy since we need to guarantee that there is no crawler running when collecting training data. We adopt a screening process from [21] that uses heuristic module to filter out most suspicious users. Besides, we manually check the log of all users and remove users with wrong URL markers.

## Chapter 5

# Security Analysis

PathMarker consists of two layers of detection and one layer of verification. The detection mechanism consists of heuristic detection and machine learning detection, and the verification method is to use CAPTCHA to constrain crawling activities and lower the false positive rate. We show the effectiveness of PathMarker on detecting five types of crawlers. We can also see that when an armored crawler can escape our detection, its crawling efficiency will be suppressed to the manual download level of human beings.

For the types of crawlers, we analyze them with five categories.

1. Basic crawler. This kind of crawlers only use some basic solutions for escaping from the defense system of the server. Heuristic detection module detects most of this type of crawlers because they are not intended designed for stealing data. For instance, some crawlers might camouflage themselves but they still visit the websites with uncontrolled speed. Those crawlers are much faster than human beings and they can be detected by checking the visiting rate. Some crawlers might conceal their activities for some aspects like user agency, but they still expose other features so our heuristic detection module is able to detect this type of crawlers by investigating the user agency, referrer, and cookie fields of HTTP request headers.
2. Timing-aware hidden crawler. When the crawler hides all heuristic features well and it can successfully mimic the timing features of human visitors, only their web page access paths are different from normal users. This kind of crawlers have an increasing ability to mimic human visitors. However, they can be detected due to their incomplete imitation on web page access paths.

3. Path-aware hidden crawler. This kind of crawler not only fakes the HTTP requests, but also control its visiting path. It may be able to download contents from a web system in a much more comprehensive scheme than pure depth-first or breadth-first algorithms. It may even closely simulate visiting path of human users. Note that it is very hard to simulate the visiting path of human users since it usually involves semantic analysis of the content. Similar to timing-aware hidden crawlers, such crawlers incompletely mimic human behaviors – the timing features are largely different from human beings, thus they could be easily detected. Note that any crawler that does not expose a human-like visiting and timing features will be eventually detected through multiple long sessions. In other words, even for some long session a crawler safely escaped, it cannot always bypass the detection.
4. Path-and-timing-aware hidden crawler. This sophisticated crawler is able to simulate a human user in both timing and visiting path. Besides, it also modifies the HTTP requests to make sure the requests are like from normal users. When it exposes almost the same behaviors as human beings, its crawling rate is also downgraded to the level of human beings. It can finally be detected when the download total volume is beyond the threshold set by the website.
5. Distributed crawlers. They may share the URLs they collected or work individually. In the case of sharing URLs, PathMarker can easily detect these crawlers by marker integrity checking. In the other case that the workers work individually, they are downgraded to individual crawlers. However, since the URLs are encrypted, these individual crawlers cannot communicate with each other to avoid visiting repeated web pages. Therefore, the crawling efficiency is suppressed by the visiting rate of a single crawler, in which the visiting rate has to be similar to normal user visiting rate.

Based on the above analysis we could conclude that for all crawlers we could stop them in the early stage unless the crawler belongs the category four which is a rare case for our

mechanism. Moreover, even for the crawlers of type four above, PathMarker could quell their efficiencies a lot.

Then let's analyze the security of our path features. For any crawler, the URLs accessed by them can be generally collected in two ways. The crawler may either collect URLs by expanding and collecting URLs from a seeding page or constructing the URLs based on URL patterns of the target system. However, adding URL marker and encryption part of PathMarker could prevent crawlers from constructing URLs since the attacker could not make a fake URL encrypted properly so our URL cannot be forged. Therefore, the crawler can only collect URLs from the web pages so they have to visit the website with a path and this is why our path features are reliable.

We use CAPTCHA to reduce the false positives of our detection model. Normal users can simply answer a CAPTCHA and stay safe. Since CAPTCHA has been proven to prevent bots, we use CAPTCHA in our system to identify the malicious crawlers.

## Chapter 6

# Implementation and Evaluation

### 6.1 PathMarker Prototype

We implement a PathMarker prototype on an open-sourced web forum. The forum uses a PHP function `site_url()` to generate dynamic web pages. In the function, we add the URL marker at the end of the URL and encrypt the whole URL except the domain name. Upon receiving a request, we decrypt the URL before parsing it. We use the well-known symmetric encryption algorithm AES to encrypt and decrypt the links. We create a table in the database to track user status information. Specifically, the table records the number of logs needed before completing a new long session, the number of marker integrity checking failures during the past day, the number of abnormal HTTP requests during the past day, number of wrong CAPTCHA input, and visiting rate. The number of logs needed before completing a new long session decides whether the visitor would be sent for analysis. Other attributes are used for heuristic detection. We use Support Vector Machine (SVM) in the machine learning module, which is implemented using LIBSVM [9].

### 6.2 Normal User Study

We publicize the forum as a student discussing forum, where students may exchange information and trade second-hand products. We require all users to agree with our data policy to sign up. In our data policy, we state that user data would be used for research purpose in an anonymous manner and the data will not be shared to other entities. Before the forum is publicized, we automatically generated more than 2500 pages about previous

news and short stories to guarantee that there is sufficient content for any crawlers to visit. We also disable all anti-crawling reactions such as popping CAPTCHA and blacklisting in our system to collect crawler data.

We do not build the forum as an insider-only forum for two reasons. First, it does not affect our evaluation on PathMarker since we can still use the persistent and stealth crawlers developed with different algorithms to crawl our website for testing the detection efficiency. Second, we wish to attract external crawlers to crawl our system for presenting our achievement. We need these external crawlers to evaluate the effectiveness of PathMarker to detect unknown crawlers. Therefore, we record user IP addresses in the URL marker if the visitor is not a logged in user.

We collect user data from the forum in an one-month period, among which half of the data is used for training and the other half is used for testing. We ensure the user data is generated by real human users through heuristic detection(See Section 4.2.2) and manual inspection. Besides the data of normal users, we include crawler data in the training set by implementing 6 crawlers to crawl the system. The 6 crawlers are (1) Depth-first, (2) Depth-first with delay, (3) Breadth-first, (4) Breadth-first with delay, (5) Random, and (6) Random with delay. Random crawlers(5 and 6) will randomly choose a link to visit from all links they gathered and put newly gained links in the link pool. As there are many other crawling algorithms, we use random crawlers to represent those crawlers. It is reasonable because in our threat model the attacker want to steal the content which means he or she does not visit the content before so he or she does not know the website’s structure or other information like popularities of different pages. In this case, most advanced crawling algorithms required extra website information such as Backlink-first or PageRank-first tend to be relatively randomly when they are using for downloading unknown website.

Crawlers with delay indicate that they will wait some time between any consecutive requests. In our implementation, the delay follows a Gaussian distribution with mean of 8 and standard deviation of 1 ( $d \sim N(8, 1)$ ). This configuration comes from our history logs



of real users. The testing set also contains both normal user data and crawler data. The crawler data consists of external crawler data and internal crawler data. We build internal crawlers ourselves using all 6 types of crawler provided by Frontera [1], which relies on Scrapy [2] to create web crawlers to crawl the system. Note that our result present a two-step detection process. The first step decides whether the visitor is a crawler. If a crawler is found, the second step classifies the visiting path of the crawler. Therefore, we train two SVM models corresponding to the two steps. For crawler detection, the first step is good enough. The reason we provide the second step is we want to show that there is existing large difference even just among the crawlers while previous work do not exploit these related fields very well. The only difference between two steps is that the SVM model for the second step does not include user data in the training set and it classifies the input visitor into three categories – Depth-first, Breadth-first and Random-like crawlers. Actually according to our result of Figure 6.2 and Figure 6.1 we could see that almost all crawlers’ visiting paths could fit in these three categories.

By analyzing the data of both crawlers and normal users, we first show they behave differently in the forum. Specifically, we have the following observations.

1. No matter which algorithm the crawler is relying on, the features of a crawler’s long session are similar to their features of short session. However, users expose significantly different behaviors in long sessions and short sessions.
2. Normal users may show a similar visiting pattern as crawlers in a short session. Most users have a clear Depth-First pattern. For example, when they finish viewing a page, they choose a link from the current page to visit. In such case, the depth rate of their short sessions is similar to a Depth-first crawler’s depth rate, which is very large. Some other users prefer to open several links in one page at the same time and then view these pages one by one. In such case, the width rate of the short session would be large.
3. The lengths of active users’ longest short session are similar. Most active users’ longest short sessions contain 20-30 log entries. We therefore set the length of a long session as

60 since we recommend the long session to be twice as a user's longest short session.

4. Most time when active user starts a new short session, they would visit the site with a different path, so the depth and width would not keep growing across different short sessions. Therefore, we could see that the depth rate and width rate of long session are usually smaller than short session for normal users.

To conclude, normal users would express different behaviors in short sessions and long sessions. Meanwhile, crawlers perform similar behaviors in long sessions and short sessions. Based on the different visiting paths crawlers and users expose, we carefully select the path features in our SVM models.

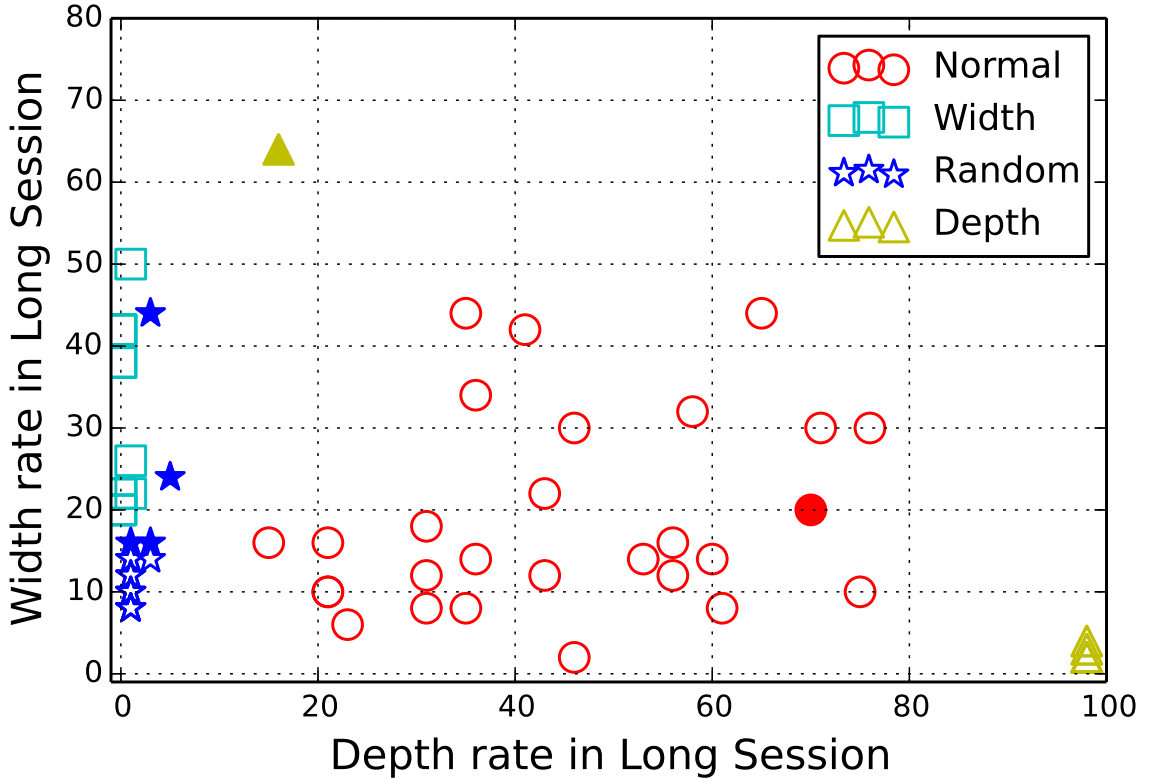


Figure 6.1: Differences Between Crawlers and Users about feature 1 and 2

To illustrate the effectiveness of our models, we show the four path features we use in Figure 6.1 and Figure 6.2, in which each shape is a data point represents a long session. The

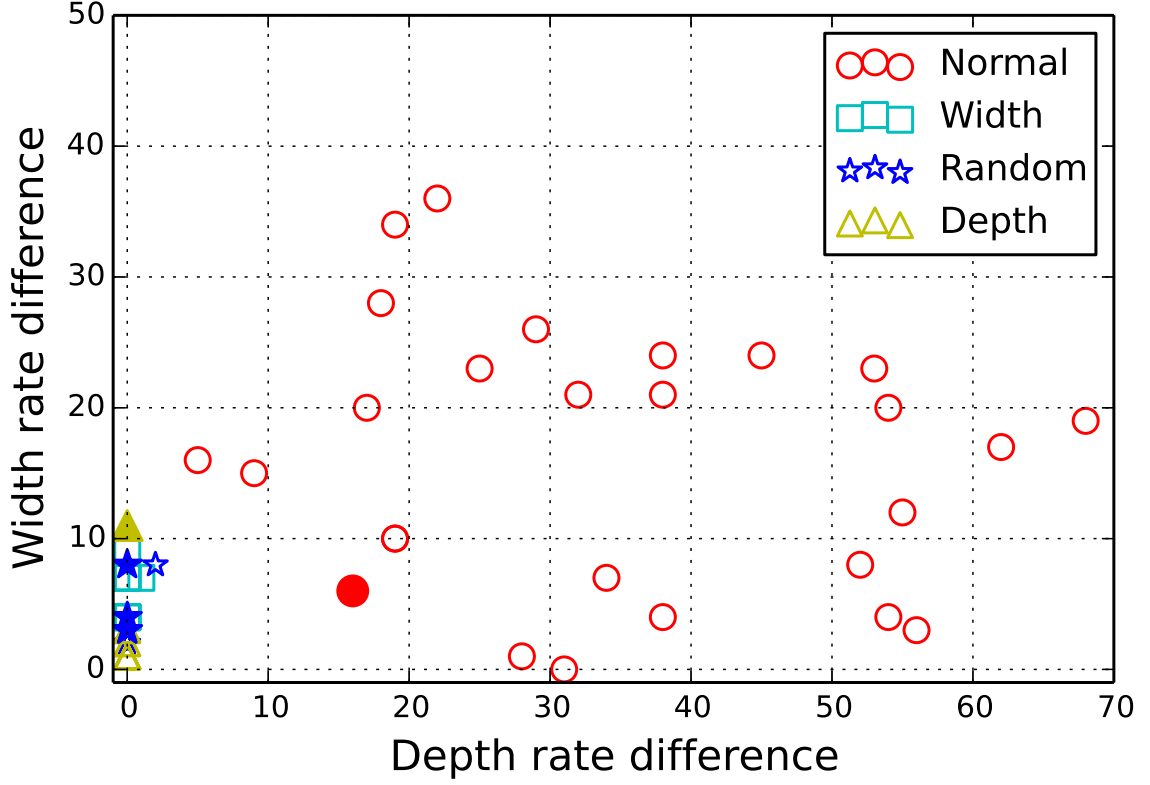


Figure 6.2: Differences Between Crawlers and Users about feature 4 and 5

circles represent the sessions of normal user, the squares represent the sessions of breadth-first crawlers, the triangles represent the sessions of depth-first crawlers and the stars represent the sessions of random-like crawlers. Meanwhile, all the solid shapes represent the case our machine learning module misjudge this session as other types. The depth rate and width rate in long sessions are shown in Figure 6.1 and the depth rate difference and width rate difference are also shown in Figure 6.2. Again, all the hollow points are the sessions we classify correctly. We can clearly see that within a long session, crawlers show extrusive path patterns in terms of width and depth. However, normal users seem having moderate width and depth rate in a long session. We can also see that the behavior difference between the longest short session and the corresponding long session is very small for crawlers while observably for normal users.

## 6.3 Performance Evaluation

We evaluate three aspects of PathMarker. They are (1) the accuracy of crawler detection of PathMarker, (2) the capabilities of PathMarker to reduce the efficiency of distributed crawlers and (3) performance overhead added to the web system. Besides, we also conduct a case study on an external crawler – Googlebots – to show the security features of PathMarker.

### 6.3.1 Detection Capabilities

We show the effectiveness and accuracy of detection. First we specifically study the effectiveness and implication of marker integrity checking. Then we show that our SVM model is able to correctly classify normal users and crawlers with a high accuracy.

**Heuristic Detection.** Our heuristic detection module consists of multiple validation mechanisms such as HTTP header investigation and visiting rate limitation. However, we only discuss the new URL marker integrity checking function since other mechanisms have already been thoroughly studied before and were industrial standards for long time. The number of all log entries of logged-in users is 2,608, among which only 6 logs contain wrong URL marker information, which indicate that a user is visiting a link that is obtained by other users. The percentage of requests with wrong URL marker is 0.23% only, meaning that the users in our system do not usually share links to each other. After manually checking the 6 logs, we believe these logs with wrong URL marker information are not generated by distributed crawlers that share a link pool. Therefore, our system is not under such attacks by any insiders. However, as we will show in our case study on Googlebots in Section 6.3.4, the heuristic detection module is able to detect these link-sharing distributed crawlers almost instantly.

**Machine Learning Detection.** Our test set contains both data from users and crawlers. Besides the 6 crawlers we use to generate test data, we also find two external crawlers. One is Googlebot and the other one is Yahoobot. We believe they are the only two search

engines that try to crawl our system since all public visitors that generate relatively abundant access logs (at least a long session) are recognized by us, among whom Google and Yahoo are the only two search engines. Note that the URLs of our system is encrypted so crawlers are likely to visit the same pages multiple times. Therefore, it is safe to assume that any search engine should generate large amount of requests to crawl our system.

We noticed that these external crawlers are based on different crawling techniques. Both Yahoo and Google use distributed crawlers, which are verified by verifying the user-agency field of HTTP requests and IP address lookup. However, one of Yahoo’s bots is responsible for over 90% of pages collected. This bot has generated over 50 long sessions and all of them are classified as a crawler by our SVM model. Different from Yahoo, Google uses an alternating approach for all distributed workers to crawl our system, which will be discussed in detail in Section 6.3.4.

Table 6.1: Classification Result

Original Type	Classify As 0	Classify As 1	Classify As 2	Classify As 3
0	96.43%	0%	3.57%	0%
1	0%	100%	0%	0%
2	0%	6.25%	93.75%	0%
3	1.51%	1.77%	0%	96.72%

Table 6.1 shows our classification results on the test set. For the accuracy about discovering crawlers from normal users, we successfully identified 96.74% crawlers’ long sessions and 96.43% normal users’ long sessions. Furthermore, For all the 3.26% crawlers’ long sessions that our SVM models misjudged as normal user long session, there is at least one other long session of the same crawler that implies the visitor is not human being. So finally we do not miss any crawler even it’s possible we misjudge its behavior for one or two long session. After we have identified a crawler, we identify the path-patterns of the crawler. As shown in Table 6.1, all three types of crawlers are correctly identified with over 90% accuracy. Type 0 represents normal users. Type 1, 2, 3 represents crawlers that expose extrusive Breadth-first, Depth-first, and random crawling features, respectively.

Basically we could notice that most bots' paths could be fit into the three patterns we define for crawlers. Note that Google and Yahoo crawlers tend to expose Random-like visiting path since they use popularity concerned crawling algorithms.

### 6.3.2 Suppressing Distributed Crawlers

As PathMarker cannot guarantee that all crawlers will be detected, web systems are still vulnerable to smart crawlers that bypass all detection mechanisms. Besides, although the efficiency of a single crawler is limited by simulating time features of normal users, distributed crawlers that leverage several user accounts to cooperate crawling remain threatening. To mitigate the data loss of such cases, PathMarker relies on adding URL marker to the URL and encrypting the whole URL to confuse crawlers. Markers can lead the crawlers to visit repeated pages because markers could be different even for the same page. The basic idea is after encryption the URL becomes different and unrecognizable so the crawlers cannot tell if the page was visited or not. Suppressing crawlers mainly contains two cases. First, a single crawler may repeatedly visit the same page if the page is retrieved from different pages. Second, distributed crawlers may repeatedly visit the same page if the page is collected by different user accounts. We now evaluate how much pressure does PathMarker add to distributed crawlers.

We assume a website contains 10,000 unique pages, each page contains 100 links to other pages. Among the 100 links, 20 of them are fixed, which means that these links reside in each page. For example, the links in the header, footer, and side bars of a website such as homepage and account management button in Facebook. The rest 80 links are drawn from all the 10,000 links. We assume the links satisfy a Gaussian probability distribution with mean 0 and standard deviation 3,333, which is a third of the number of pages. Each page corresponds to a number between 1 and 9,999. The probability of a page being selected is  $P_n = (cdf(n) - cdf(n - 1)) \times 2$ , in which  $n$  is the page number and  $cdf$  is the cumulative density function. Note we double the probability of each page since we ignore negative numbers. Besides, the the sum of probability of all pages is smaller

than 1 since Gaussian distribution does not have a boundary. Therefore, we compensate the probability to the last page.

Now we have developed a website model under PathMarker for a distributed crawler to crawl. The crawlers do not share a link pool to select the next page to visit in order to bypass URL marker integrity checking). Furthermore, the distributed crawlers could share crawled page URLs for avoiding repeatedly visiting same pages (If they do not share this information they would have much worse crawling efficiency). We conduct two sets of experiment. First, we show the web pages crawled in a fixed time period for single and distributed crawlers. We assume the crawling efficiency of each distributed worker is the same. Therefore, theoretically a distributed crawler consisting of 10 workers visit 10 times of pages as a single crawler in a fixed time period.

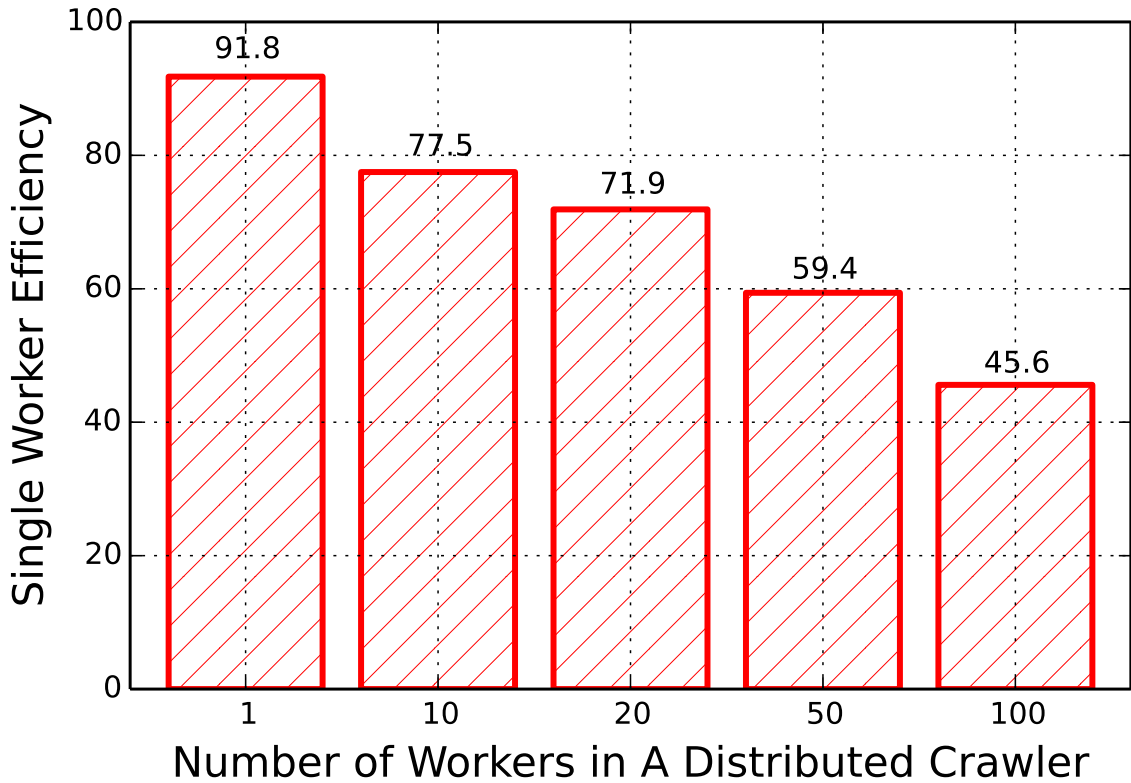


Figure 6.3: Suppressing Distributed Crawlers

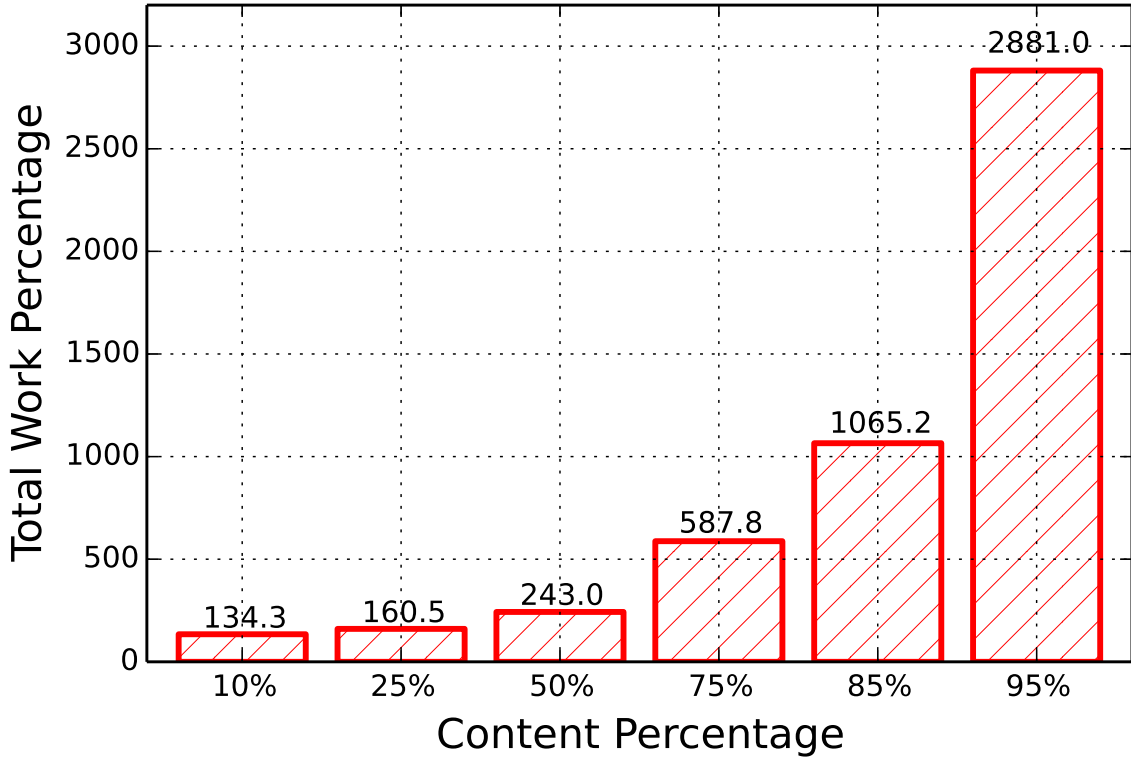


Figure 6.4: Overhead for Distributed Crawlers

Figure 6.3 shows the efficiency about visiting new pages for crawlers with different total number. The number 91.8 of column 1 means when the crawler only has one worker, for the first 100 new URLs got by the worker would cover 91.8 new pages while others would be repeated. The efficiency of each individual worker decreases as the number of worker increases. When using 100 workers, the efficiency of each worker is less than half of a single crawler. More important point is that it's easy to see while the crawlers are keeping crawling, their efficiency would decrease because more and more repeated URLs of same pages they would crawl. In general, crawlers would get the URLs of every page twice or more from different parent pages but it couldn't notice because the URL markers are different and the final encrypted URLs they get are different.

Figure 6.4 illustrates the total work a 10-worker distributed crawler need to do for obtain a certain percentage of the whole website. 100% work means the workload for a crawler who could always visit new page and extra work means this crawler visits



repeated pages. As the figure shows, crawling half of all content requires 143% more queries than crawling an unencrypted website, which means that the crawlers waste 58.9% of its crawling power. Furthermore, over 965% more queries are required to crawl 95% of all content, which indicates a 278.1% crawling power waste. In conclude, we show that even if armored distributed crawlers can escape all detection methods in PathMarker, their efficiency will be largely suppressed by the encrypted path and URL markers and the suppressing effect getting better when the total number of workers or targeted pages is increasing.

### 6.3.3 System Overhead

Our defense system would introduce overhead to the server from two parts: analyzing program and server modification. For analyzing program, the memory consumption is limited. It is written by C and it is just a project whose size is 175KB. Furthermore, it is separated from website so we do not need to worry about that it would affect the running of website. For the server modification, the memory overhead is limited too. We need to add two additional tables in server side but each table only has several columns and for most websites the size of additional tables are much smaller than the size of their original tables that save logs.

To evaluate the runtime overhead introduced by server modification, we conduct the experiment to show how much runtime overhead PathMarker puts on the web system in a visitor’s perspective. We record the time a HTTP request is received and the time the web page is sent out. By computing the time interval we learn the time needed for the server to generate the page. We setup two forum copies that have identical database tables, on one of which we build PathMarker on it. We implement a crawler to automatically query the homepage of the forum, which consists of 116 links, for 1,000 times on both of the two copies. Note that crawlers may not fetch the images in the homepage. However, it does not affect our experiment results since we are only interested in the time overhead introduced by URL markers. The average time needed to generate a page without PathMarker is

32.0ms and the average time needed to generate a page with PathMarker is 41.5ms. This increase is acceptable since for a normal user, they would not feel any extra delay of a 10ms difference and the number of links in every page is large enough for the website contain confidential documents.

#### 6.3.4 Googlebots – A Case Study

After we publicize the online forum, we notice Google search engine is actively visiting it by checking User-Agency and IP address lookup. Since Googlebots is the largest crawler that uses sophisticated and evolving algorithm, we particularly study the behavior of Googlebots under PathMarker to show its security features. Note that for the experiment in this section, we use IP addresses to identify different crawlers so the information in the URL marker is parent URL and the IP address about getting the link.

Googlebot is a typical distributed link sharing crawler. During the one-month data collection, we discovered Googlebots from over 50 IP addresses, which indicate that at least 50 crawling workers are crawling our system. There are 19,844 log entries recording the activities of Googlebots. Although there are many workers visiting our system, we notice that most of them only visit one or two times while several workers are responsible for most of the requests (only 9 workers visited the website for over 25 times). We reckon that Google is trying to probe the network and assign the fastest workers to crawl our system. Now Google can be treated as an attacker who try to download the content of our system using a distributed crawler consisting of several active workers. We ignore the workers that visit our system very few times since it is almost infeasible to prohibited a potential malicious visitor from downloading few pages. Besides, our threat model allows an attacker to possess at most several user accounts.

Now we explain how Googlebots can be mitigated or detected by multiple defense mechanisms. With URL marker appended in the end, the URL of a page varies depending on the identity of the visitor and the page that links to it. Therefore, different visitor collects different URLs for the same page. Even for a single visitor, the URL could be

different because they may visit the page from different pages. Therefore, Google usually collects multiple URLs to a same page and visit the page repeatedly. Among the 19,844 requests, only 1,010 pages (around 40% of total pages) are unique, which means that Google has wasted almost 95% of its crawling power on visiting repeated pages. Note that we encrypt the path and URL marker of a URL so that the attacker cannot parse the collected URLs and find out whether they are linking to repeated or new pages.

Google crawler does not hide itself in HTTP requests by stating its identity as Googlebots. However, it has an efficient rate control mechanism to avoid being banned due to very high visiting speed. We found that in our dataset, each single worker will never visit more than 5 pages in a short session. After manual checking we note that each worker will wait for some time between each consecutive requests from several seconds to several hundred of seconds. Therefore, Google is under kept itself safe from being banned by most of visiting rate controlling system.

Table 6.2: Example Logs for Detecting Distributed Crawlers

Visitor IP	URL	Marker
66.249.67.83	home/node/show/12/	home/topic/show/855/;66.249.67.71
66.249.67.77	home/topic/add/	home/home/getmore/13/;66.249.67.83
66.249.67.86	home/policy/	home/user/profile/13/;66.249.67.80
66.249.67.80	home/node/	home/home/getmore/70/;66.249.67.92
66.249.67.71	home/node/show/12/15/	/index.php/node/show/12/8/;66.249.67.77

While the Googlebot is able to escape visiting rate detection, we can easily discover that it is a distributed crawler by URL marker integrity checking. The total number of wrong URL markers are 12,271, which is 62% of all requests by Googlebots, which means that 62% of the URLs a worker visits is collected by other workers. We show several examples of wrong URL markers in Table 6.2. The table records the IP address of the visitor, the page URL, and the URL marker. From the last field of the URL markers we can see that the collector of this URL marker is from a different IP address than the visitor IP address. Therefore, the URL marker integrity checking mechanism can capture distributed bots in a very clean and instant way.

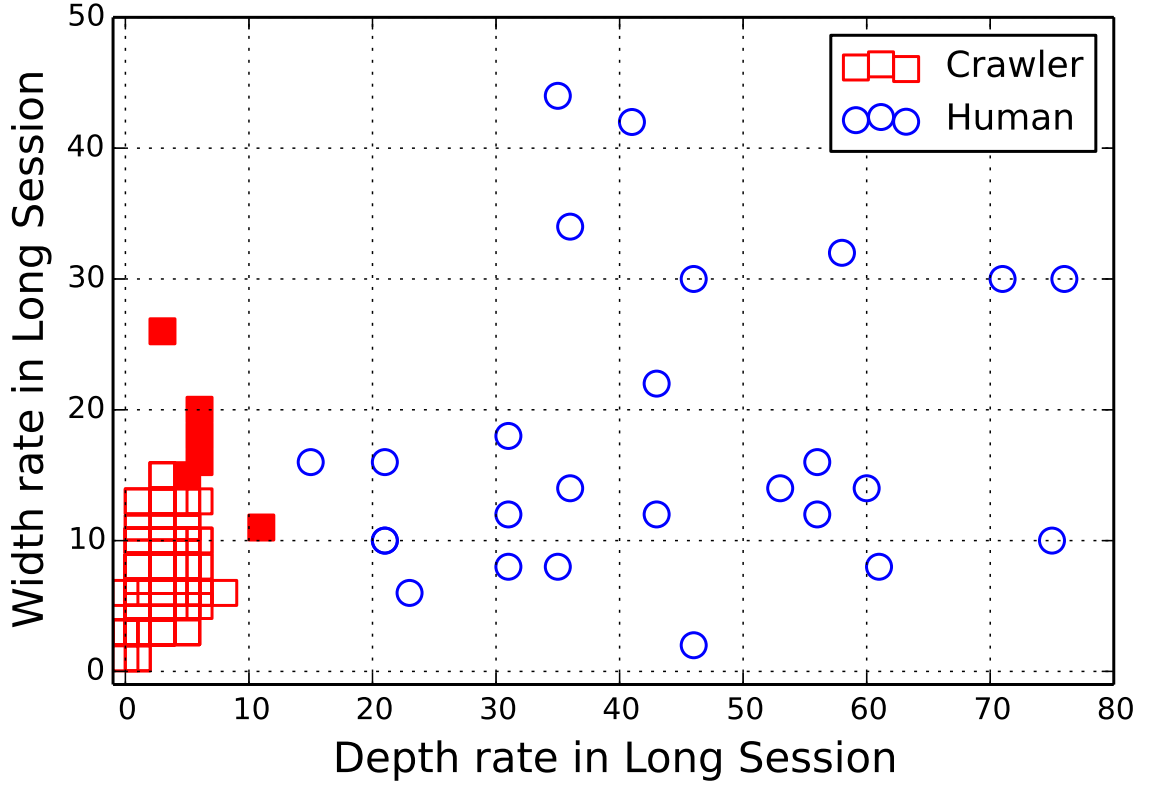


Figure 6.5: Depth and Width rate in long session for Google Bots

Even Googlebots can be easily detected through URL marker integrity checking, we also analyze its path features to illustrate how different they are from normal users. We gathered totally 381 long sessions from Googlebots, which means the SVM module is invoked 381 times to analyze the owner of each long session. Figure 6.5 shows the depth rate and width rate of long session of Googlebots and normal users. We can see an obvious difference between the two groups even we only consider the two features. Among the 381 long session, our SVM model can correctly identify 376 of them as belonging to a crawler, which indicates that the accuracy achieves 98.425%. We also emphasize that although some of the long sessions are misjudged, every single worker has at least one correctly identified long session. Therefore, all Googlebots that triggered our machine learning module (which means having at least one long sessions) will be prompted a CAPTCHA in a full functionality PathMarker system.

In conclusion, we illustrate that Googlebots will be detected by both heuristic detection and SVM detection. We also show how PathMarker can largely suppress the efficiency of distributed crawlers by URL marker integrity checking and URL encryption.

## Chapter 7

# Limitation and Discussion

### 7.1 Usability of URL Encryption

In PathMarker, the path and URL marker information in URLs are encrypted in order to preserve the website structure and mitigate the efficiency of distributed crawlers. This may trigger some usability issues of our system. We admit that users could not know what's the plaintext of URL so the user experience might be affected. The most severe case we could see is that users might only remember the URL of the homepage because other URLs are ciphertext and they are difficult to remember. However, since most users do not remember URLs by themselves and our system works well when user reopens the encrypted URLs in bookmark folder, we think this is not an important issue. Moreover, current web pages have their own titles to identify the content so the plaintext of URL is not necessary for the users. Another issue is users are only allowed to visit others' links under a threshold. For this problem, now we set a relatively high threshold for user to visit URLs from others and for our forum all normal users have not been classified as crawler because of visiting others' links yet. We believe under our threat model, share links will not be a common scenario after all these protected pages are not accessible to everyone. Therefore, we believe that the impact on user experience is acceptable. Furthermore, we still reveal domain name of every URLs so we do not need to worry that users would be vulnerable to phishing attacks.

We believe for protecting valuable contents, it is acceptable to sacrifice some usability. It is also feasible for us to only encrypt the URL marker part and expose the original URLs to users but attackers might forge HTTP requests that can bypass URL marker

integrity checking in this case. Furthermore, they can even infer the defense system and fake visiting path in URL markers to further break down the machine learning accuracy of PathMarker. Therefore, there is a trade-off between usability and security for server maintainers.

## 7.2 System's Configuration

In the paper, we mention that for our online forum, we use the condition that time interval is larger 10 seconds to group a short session and we use 60 as the fixed length of long session. For the other website administrators who want to equip PathMarker, they only need to check their existing logs to decide the interval of short session and then all parameters could be calculated and do not need to change. As long as their websites' structure are not changed a lot or the content's size they want to protect do not grow fast, it is not necessary to re-set those parameters or re-train the SVM.

## 7.3 Deployability of PathMarker

There are many ways to implement the websites' servers and generate web pages. From a high level, there are two kinds of web pages: static web pages and dynamic web pages. For the dynamic web pages, there are different server-side scripting languages like PHP and Python to achieve dynamic website structures. Under this circumstance, we realize that it is very difficult to design a generic tool for all website servers to adapt their URLs with PathMarker. However, to survey our deployability, we look into two famous open-source website framework (Discuz![10] and startBBS which is based on the CodeIgniter[15]) and discuss with several website maintainers.

Based on our survey, we conclude that for static web pages it could deploy PathMarker easily by automatically changing all the URLs in scripts. Besides, even for dynamic web pages like [10] and [15], there are usually one or two most common functions to generate URLs and the total functions would be less than 10. Basically, most dynamic web page

servers are able to integrate PathMarker in their system. We believe it is a reasonable implementation overhead for those who want to protect their valuable web contents.

## 7.4 Baiting Link

In PathMarker, we noticed the crawlers' path patterns could be classified into three algorithm categories, which are Depth-first, Breadth-first, and Random-like. In fact, knowing the behavior of crawlers is able to help predicting the next link that may be visited, and such information is helpful to capture the crawlers too. To show how it works, we introduce a related concept – baiting link as the future work. known baiting link works as a cushion between detection and CAPTCHA. It is a kind of link that hardly any normal users would be interested in. For example, it could be an outdated advertisement in the corner of a webpage. Baiting link is used to enhance user experience while effectively block crawlers. Since users seldom click the baiting link, they can still be safe from CAPTCHA even though the crawler detection mechanism mistakenly consider them as crawlers. When a baiting link is visited, a CAPTCHA pops up and the visitor is required to input the correct CAPTCHA. If the user fails to recognize the CAPTCHA, the system may simply block the user. A key challenge is to ensure that the baiting link will be visited by a crawler within limited requests. For a Depth-first crawler, it is likely to visit the first link of the next page, which can be where the baiting link located. Similarly, for a Breadth-first crawler, it is likely to visit all pages of the next page so the baiting link can be located at any place in the next page. Learning the crawling behavior of crawlers can help websites put the baiting link in a location that can easily trap crawlers while not attract normal users.



## Chapter 8

### Related Work

Both crawlers and anti-crawler mechanisms evolve in their arms race. At the beginning, a naive crawler is the kind of crawler that does not make any effort to conceal its activities at all. It could be a rough crawler created by the attacker. Next, a basic crawler has realized that it can be easily detected, so it forges its requests to make them look like normal requests. Also, a timing-aware hidden crawler also control its timing features, such limiting its visiting rate by adding random delays. Later, an armored crawler may be able to simulate a human user in both visiting timing and visiting path patterns. Moreover, distributed crawlers may assign crawling activities to multiple agents, who are only responsible for downloading certain part of the website content. Each individual crawler can be any of the above five types of crawlers.

Web crawlers have been studied and characterized for a long time[28, 24]. [14] investigates the difference between resources such as images crawler and human request; [12] focuses on analyzes the features and preferences on search engine crawlers. Many works have been published to show observations that are useful to detect crawlers from a large scale network service [40, 26]. Frontier that decides the crawling behavior is a core component of crawlers[28]. Many works optimize the frontier to help crawlers achieve better crawling results. [5, 11, 22, 7]. For instance, [23] proposes a novel solution for mimicing human behaviors according to the human observational proofs so the new crawler could escape the detection of other defense systems.

Nowadays, many web systems deploy basic anti-crawling mechanism and they can detect most naive and basic crawlers. Heuristic detection methods have been widely adopted to identify and defeat malicious crawlers through analyzing the User-Agent, referrer, and

even cookie fields in the HTTP request headers as well as monitoring the visiting rate of each individual visitor. However, they are effective on reducing crawlers' download efficiency but cannot detect all stealthy crawlers. For instance, a persistent attacker can easily surpass those heuristic detection mechanisms by camouflaging and rate limiting. Moreover, attackers can deploy multiple bots to boost the crawling speed, though an individual crawler's efficiency is bounded by the rate limitation set by the websites. One recently work [36] also observed that cyber-criminals might misuse several accounts on stealing sensitive information and they proposed a solution by mapping between an online account and an IP address for capturing these crawlers. However, their final target are detecting those accounts while protecting the content is not considered as the major target in their work.

Researchers have developed numerous anti-crawling artifacts that explore machine learning techniques to suppress the efficiency of crawlers[40, 26] or even completely block them [35, 21, 38, 13], based on the observations that crawlers will behave differently from human beings [38, 21]. One challenge for machine learning based solutions is to select the set of effective features to train the machine learning model. In one of the earliest work [38], Tan and Kumar develop 24 features to train the anti-crawling model.

A number of follow-up works focus on using various features under different scenarios [41, 3, 19, 8, 40, 26, 21]. For example, Jacob et al. [21] use multiple timing features to characterize crawlers, and they are able to differentiate crawlers and busy proxies based on more regular time pattern of crawlers. Numerous features have been proposed and proven to be effective for specific use cases [35, 34]. The usage of request-related features such as the percentage of GET request and POST request, percentage of error responses, and total number of pages requested has been proposed in [21]. There are also other comprehensive features that profile the visiting behavior of crawlers, including traffic timing shape [21], page popularity index, standard deviation in visiting depth [35], clickstream related features[27, 4] and some special features for the Bayesian network to recognize crawlers [32, 33, 37]. Some others are even trying to understand the crawlers to

capture them [39, 31].

Constrained by the crawling algorithms for automatic web content download, it is difficult for crawlers to perfectly mimic human beings' visiting patterns. Therefore, path related features can effectively differentiate crawlers and normal users. Stevanovic et al. [35] uses standard deviation of requested page depth as one feature to describe the visiting path. However, it cannot accurately reflect the difference between crawlers and normal users since the page depth is simply extracted from parsing the URL. Tan and Kumor [38] learn session depth and width from the referrer field of HTTP request headers to more accurately describe the path information. However, it is easy for intelligent crawlers to fake the referrer field of HTTP headers. Similarly, PathMarker also largely rely on path-related features to identify crawlers. Differently, PathMarker relies on the URL marker appended to each URL to learn the referring relationship between two requests. The URL marker and path of a URL are encrypted so the crawler cannot fake visiting path through forging URL markers.

How to deal with crawlers after detecting them is also an essential problem. Setting traps in pages is a common method to catch crawlers [6]. Specifically, websites may integrate invisible links in the web pages that only crawlers can view. As long as the links are visited, the visitors will be directed to an infinite loop or wrong content. Park et al. [29] capture crawlers that do not generate mouse or keystroke events. However, these methods can be easily bypassed by page rendering analysis or imitating mouse and keystroke operations. Among various kinds of crawler blocking mechanisms, using CAPTCHA one of the most reliable one since it is a kind of Turing Test to finally detect machine from users. Recently CAPTCHA techniques may even use video as CAPTCHA [25, 18]. [18] also requires people to recognize more complex content of image such as the orientation.

## Chapter 9

# Conclusions

In this paper, we present an anti-crawler system named PathMarker to help server administrators capture stealthy persistent crawlers who want to download the contents of servers maliciously. PathMarker appends URL markers at the end of all URLs to record information regarding to visiting path and the URL obtainer. PathMarker is able to distinguish crawlers from normal users based on their visiting path and time features. PathMarker can quickly capture distributed crawlers by checking the URL marker integrity. Even for the most advanced crawler that may bypass our detection, their crawling efficiency can dramatically suppressed by our system to the level of human beings. We evaluate PathMarker on an online forum website. Through the data collected in one month period, we are able to detect 12 popular crawlers with a high accuracy. Besides the crawlers we test, we also detect external crawlers such as Yahoo and Google bots. Therefore, we conduct a case study on Google crawlers to show the security features of the two layers of protection mechanism of PathMarker.

# Bibliography

- [1] Frontera 0.3. <http://frontera.readthedocs.org/en/latest/index.html>.
- [2] Scrapy 1.0. <http://scrapy.org/>.
- [3] Alireza Aghamohammadi and Ali Eydgahi. A novel defense mechanism against web crawlers intrusion. In Electronics, Computer and Computation (ICECCO), 2013 International Conference on, pages 269–272. IEEE, 2013.
- [4] Fatemeh Ahmadi-Abkenari and Ali Selamat. An architecture for a focused trend parallel web crawler with the application of clickstream analysis. Information Sciences, 184(1):266–281, 2012.
- [5] Ricardo Baeza-Yates, Carlos Castillo, Mauricio Marin, and Andrea Rodriguez. Crawling a country: better strategies than breadth-first for web page ordering. In Special interest tracks and posters of the 14th international conference on World Wide Web, pages 864–872. ACM, 2005.
- [6] Luciano Barbosa and Juliana Freire. An adaptive crawler for locating hidden-web entry points. In Proceedings of the 16th international conference on World Wide Web, pages 441–450. ACM, 2007.
- [7] Sotiris Batsakis, Euripides GM Petrakis, and Evangelos Milios. Improving the performance of focused web crawlers. Data & Knowledge Engineering, 68(10):1001–1013, 2009.

- [8] Christian Bomhardt, Wolfgang Gaul, and Lars Schmidt-Thieme. Web robot detection-preprocessing web logfiles for robot detection. Springer, 2005.
- [9] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [10] Comsenz Inc. Discuz forum. <http://www.discuz.net/forum.php>.
- [11] Clément De Groc. Babouk: Focused web crawling for corpus compilation and automatic terminology extraction. In *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 01*, pages 497–498. IEEE Computer Society, 2011.
- [12] Marios D Dikaiakos, Athena Stassopoulou, and Loizos Papageorgiou. An investigation of web crawler behavior: characterization and metrics. *Computer Communications*, 28(8):880–897, 2005.
- [13] Derek Doran and Swapna S Gokhale. Web robot detection techniques: overview and limitations. *Data Mining and Knowledge Discovery*, 22(1-2):183–210, 2011.
- [14] Derek Doran, Kevin Morillo, and Swapna S Gokhale. A comparison of web robot and human requests. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 1374–1380. ACM, 2013.
- [15] EllisLab. Codeigniter. <https://codeigniter.com/>.
- [16] Steven Gianvecchio, Zhenyu Wu, Mengjun Xie, and Haining Wang. Battle of botcraft: Fighting bots in online games with human observational proofs. In *Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS '09*, pages 256–268, 2009.

- [17] Steven Gianvecchio, Mengjun Xie, Zhenyu Wu, and Haining Wang. Measurement and classification of humans and bots in internet chat. In Proceedings of the 17th USENIX Conference on Security Symposium, pages 155–169, 2008.
- [18] Rich Gossweiler, Maryam Kamvar, and Shumeet Baluja. What’s up captcha?: a captcha based on image orientation. In Proceedings of the 18th international conference on World wide web, pages 841–850. ACM, 2009.
- [19] Weigang Guo, Shiguang Ju, and Yi Gu. Web robot detection techniques based on statistics of their requested url resources. In Computer Supported Cooperative Work in Design, 2005. Proceedings of the Ninth International Conference on, volume 1, pages 302–306. IEEE, 2005.
- [20] Incapsula Inc. Bot traffic report 2014. <https://www.incapsula.com/blog/bot-traffic-report-2014.html>.
- [21] Gregoire Jacob, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Pubcrawl: Protecting users and businesses from crawlers. In USENIX Security Symposium, pages 507–522, 2012.
- [22] Jing Jin, Jeff Offutt, Nan Zheng, Feng Mao, Aaron Koehl, and Haining Wang. Evasive bots masquerading as human beings on the web. In Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on, pages 1–12. IEEE, 2013.
- [23] Jing Jin, Jeff Offutt, Nan Zheng, Feng Mao, Aaron Koehl, and Haining Wang. Evasive bots masquerading as human beings on the web. In Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on, pages 1–12. IEEE, 2013.
- [24] Md Abu Kausar, VS Dhaka, and Sanjeev Kumar Singh. Web crawler: a review. International Journal of Computer Applications, 63(2), 2013.

- [25] Kurt Alfred Kluever and Richard Zanibbi. Video captchas: usability vs. security. IEEE Western New York Image Processing Workshop, 2008.
- [26] Junsup Lee, Sungdeok Cha, Dongkun Lee, and Hyungkyu Lee. Classification of web robots: An empirical study based on over one billion requests. *computers & security*, 28(8):795–802, 2009.
- [27] Anlia Loureno and Orlando Belo. Applying clickstream data mining to real-time web crawler detection and containment using clicktips platform. In *Advances in Data Analysis*, Reinhold Decker and Hans-J. Lenz, editors, *Studies in Classification, Data Analysis, and Knowledge Organization*, pages 351–358, 2007.
- [28] Christopher Olston and Marc Najork. Web crawling. *Foundations and Trends in Information Retrieval*, 4(3):175–246, 2010.
- [29] KyoungSoo Park, Vivek S Pai, Kang-Won Lee, and Seraphin B Calo. Securing web service by automatic robot detection. In *USENIX Annual Technical Conference, General Track*, pages 255–260, 2006.
- [30] Patrick Sexton. The googlebot guide. <https://varvy.com/googlebot.html>.
- [31] Benjamin I. P. Rubinstein, Blaine Nelson, Ling Huang, and Anthony D. Joseph. Antidote: Understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, 2009.
- [32] Athena Stassopoulou and Marios D Dikaiakos. Crawler detection: A bayesian approach. In *Internet Surveillance and Protection, 2006. ICISP’06. International Conference on*, pages 16–16. IEEE, 2006.
- [33] Athena Stassopoulou and Marios D Dikaiakos. Web robot detection: A probabilistic reasoning approach. *Computer Networks*, 53(3):265–278, 2009.



- [34] Dusan Stevanovic, Aijun An, and Natalija Vlajic. Feature evaluation for web crawler detection with data mining techniques. *Expert Systems with Applications*, 39(10):8707–8717, 2012.
- [35] Dusan Stevanovic, Natalija Vlajic, and Aijun An. Detection of malicious and non-malicious website visitors using unsupervised neural network learning. *Applied Soft Computing*, 13(1):698–708, 2013.
- [36] Gianluca Stringhini, Pierre Moulanne, Gregoire Jacob, Manuel Egele, Christopher Kruegel, and Giovanni Vigna. Evilcohort: detecting communities of malicious accounts on online services. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 563–578, 2015.
- [37] Grazyna Suchacka and Mariusz Sobkow. Detection of internet robots using a bayesian approach. In *Cybernetics (CYBCONF), 2015 IEEE 2nd International Conference on*, pages 365–370. IEEE, 2015.
- [38] Pang-Ning Tan and Vipin Kumar. Discovery of web robot sessions based on their navigational patterns. In *Intelligent Technologies for Information Analysis*, pages 193–222. Springer, 2004.
- [39] Guowu Xie, Huy Hang, and Michalis Faloutsos. Scanner hunter: Understanding http scanning traffic. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '14*, pages 27–38, 2014.
- [40] Fang Yu, Yinglian Xie, and Qifa Ke. Sbotminer: large scale search bot detection. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 421–430. ACM, 2010.
- [41] Dexiang Zhang, Difan Zhang, and Xun Liu. A novel malicious web crawler detector: Performance and evaluation. *International Journal of Computer Science Issues (IJCSI)*, 10(1), 2013.