Cybersecurity

RESEARCH Open Access

PathMarker: protecting web contents against inside crawlers

Shengye Wan[1]*, Yue Li[1] and Kun Sun[2]

Abstract

Web crawlers have been misused for several malicious purposes such as downloading server data without permission from the website administrator. Moreover, armoured crawlers are evolving against new anti-crawler mechanisms in the arm races between crawler developers and crawler defenders. In this paper, based on one observation that normal users and malicious crawlers have different short-term and long-term download behaviours, we develop a new anti-crawler mechanism called PathMarker to detect and constrain persistent distributed crawlers. By adding a marker to each Uniform Resource Locator (URL), we can trace the page that leads to the access of this URL and the user identity who accesses this URL. With this supporting information, we can not only perform more accurate heuristic detection using the path related features, but also develop a Support Vector Machine based machine learning detection model to distinguish malicious crawlers from normal users via inspecting their different patterns of URL visiting paths and URL visiting timings. In addition to effectively detecting crawlers at the earliest stage, PathMarker can dramatically suppress the scraping efficiency of crawlers before they are detected. We deploy our approach on an online forum website, and the evaluation results show that PathMarker can quickly capture all 6 open-source and in-house crawlers, plus two external crawlers (i.e., Googlebots and Yahoo Slurp).

Keywords: Anti-Crawler mechanism, Stealthy distributed inside crawler, Confidential Website content protection

Introduction

With the prosperity of Internet data sources, the demand of crawlers is dramatically increasing. The 2018 bot traffic report from Distil (Network 2018) points out that bots account for 42.2% of all website traffic, and malicious bots contribute almost 21.8% of the web traffic. Meanwhile, an increasing number of new algorithms (Baeza-Yates et al. 2005; De Groc 2011; Batsakis et al. 2009; Jin et al. 2013a) have been adopted by malicious crawlers to increase the download efficiency and decrease the chance of being detected.

To protect confidential documents or sensitive data, most websites require users to authenticate themselves before accessing the valuable web content. Though this layer of authentication can successfully block external malicious crawlers, authorized insiders can still crawl the entire website. For instance, with full access to the NSA's files, Edward Snowden used inexpensive and widely available web crawler tool to scrape at least 1.7 million secret files (Snowden used common web crawler tool to collect NSA files). To prevent from being easily detected, stealthy insiders may customize their crawlers to better mimic the access behaviour of real users. To compensate the intentionally degraded download efficiency, insiders have the patience to launch a persistent download in a long time period. Furthermore, multiple insiders may collude and adopt a divide-and-conquer strategy to speed up the crawling process.

To detect and constrain malicious crawlers, researchers have developed a number of anti-crawler mechanisms (Stevanovic et al. 2013; Tan and Kumar 2004; Stassopoulou and Dikaiakos 2006; 2009; Doran and Gokhale 2011; Yu et al. 2010; Park et al. 2006; Gianvecchio et al. 2009; Gianvecchio et al. 2008; Jacob et al. 2012), which can be generally divided into two categories, namely, *heuristic detection* and *machine learning detection*. Heuristic detection mechanisms rely on analyzing well defined features (e.g., visiting rate of individual visitor) to define abnormal website access behaviour at first, and then define any other behaviour as normal

*Correspondence: swan@email.wm.edu
[1] Department of Computer Science College of William and Mary, Williamsburg 23187-8795, VA, USA
Full list of author information is available at the end of the article

behaviour (Yu et al. 2010; Park et al. 2006). However, they cannot guarantee to detect crawlers that are able to constrain their behaviours and manipulate those observed features. Machine learning detection mechanisms can detect malicious crawlers based on the different visiting patterns between normal users and malicious crawlers (Stevanovic et al. 2013; Stassopoulou and Dikaiakos 2006; 2009). In other words, they first model the normal website access behaviour and then define any other behaviour as abnormal. Most recent anti-crawlers mechanisms combine these two techniques to better defeat malicious crawlers (Tan and Kumar 2004; Jacob et al. 2012). However, it is still a challenge to detect and constrain armoured inside crawlers that can collude in a persistent scraping.

In this paper, we develop a new anti-crawler mechanism called *PathMarker* that aims to detect and constrain persistent distributed inside crawlers, which have valid user accounts to stealthily scrape valuable website content. Moreover, we manage to accurately detect those armoured crawlers at their earliest crawling stage. The basic idea is based on one observation that the normal users and malicious crawlers have different short-term and long-term download behaviours. In other words, crawlers have similar accessing patterns regarding to the path (e.g., depth-first, width-first, or random access) in both the short-term and the long-term; while human beings have obvious accessing patterns only in the short term and have no certain patterns in the long term. This is because crawlers are working based on certain crawling algorithms, and once the algorithm is chosen, the crawling paths would follow certain pattern.

It is well known that existing path based anti-crawler solutions (Tan and Kumar 2004) have the difficulty in accurately calculating the depth and width of one access. Given a group of access logs, we may not know the *parent page*[1] of each log entry's link so we can only guess how deep or wide this link is. For instance, when one page A is linked in two other pages B and C, it is difficult to find its parent page if both pages B and C have been accessed before A. Moreover, when a number of distributed crawlers collude in a download task, each individual crawler may have no obvious path pattern.

We solve this problem in PathMarker by automatically generating and appending a marker to each web page URL. We call the marker as *URL marker*, which records the page that leads to the access of this link and the user ID who accesses this link. To further protect the URL markers from being misused by armoured crawlers, we encrypt the URL markers along with the original URL except the domain name. URL markers can help quickly detect distributed crawlers that share collected links in a pool 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 that page. With the aid of URL marker, we can not only perform more accurate heuristic detection using path related features, but also develop a Support Vector Machine (SVM) based machine learning detection to model the different patterns of URL visiting paths and different URL visiting timings between human beings and malicious crawlers. Moreover, URL markers can decrease the crawler's download efficiency. since the crawlers may download the same web page multiple times via different parent pages by different users.

We develop a PathMarker prototype on an online open source forum website. We train SVM models based on the access 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 crawlers. Moreover, two external crawlers, Googlebot (Sexton) and Yahoo Slurp (Yahoo), are also detected, and PathMarker can successfully suppress these two distributed crawlers.

In summary, we make the following contributions.

- We design an anti-crawler system named PathMarker to detect persistent and distributed web crawlers that have the credentials to download the privately accessible valuable web contents as insiders. PathMarker relies on the URL visiting path and URL visiting timing features derived from the encrypted URL markers added to the URLs.
- PathMarker can instantly detect distributed crawlers that share download links with a high accuracy. If the distributed crawlers do not share links in a pool, our encrypted URL technique can effectively suppress their efficiency to a rate lower than the sum of individual crawlers. For individual persistent crawler that mimics human being's download behaviour, we can reduce its download speed of to the level of human beings.
- We implement a PathMarker prototype on an online forum website and the experimental results show that PathMarker is capable of detecting state-of-the-art crawlers including Googlebot and Yahoo Slurp.

The remaining of the paper is organized as follows. "Background" section introduces background information. "Threat model and assumptions" section describes threat model. We present the architecture of PathMarker in "PathMarker architecture" section and perform the security analysis in "Security analysis" section. The system implementation is presented in "System implementation" section and the evaluation is detailed in "System evaluation" section. "Discussion and limitations" section discusses limitations and potential extensions. We present related works in "Related work" section. Finally, we conclude the paper in "Conclusions" section.

Background

Crawling algorithms

Web crawlers start their downloads 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 that page. Based on the crawling algorithm, the crawler picks the next page from the already collected URLs to visit. Olston and Najork (2010) summarizes 15 techniques that are able to decide the order of page accesses using different features. Here we focus on three common techniques (i.e., depth-first, width-first, and random-like) to show the visiting paths in different crawling algorithms. We use a simple website structure shown in Fig. 1 to present the differences.

First, starting from the seed page (home page), *depth-first* crawler greedily visits the URLs as far as possible in a branch of a website tree graph before backtracking. Thus, its visiting path in Fig. 1 is *homepage*, *page*1, *page*3, *page*4, and *page*2. *Breadth-first* crawlers visit all links of a page before visiting links of another page, so its visiting path is *homepage*, *page*1, *page*2, *page*3, and *page*4.

PageRank-first crawlers aim to collect most valuable content of a server. PageRank is an algorithm used by Google to rank all pages of a website. *PageRank-first* evaluates the importance of a web page based on several features and then keeps visiting the link with the highest rank. However, these features have no directly relationship with depth or width. Therefore, we classify PageRank-first algorithms as “random-like”, since the corresponding crawlers have no preference about depth or width. Besides this one, all the crawlers whose visiting path has no preference about depth or width can be classified as “random-like”.

Anti-crawler mechanisms

Researchers have developed a number of anti-crawler mechanisms.

Hypertext transfer protocol (HTTP) request fields checking. Web server administrators can examine their web servers’ logs and check several fields of the HTTP request such as referrers and cookies to detect the abnormal requests. Some crawlers’ requests miss these fields while some other requests have obviously differences comparing with normal users’ requests in these fields. One typical field of a request is *User-Agent*. Each HTTP request contains the field User-Agent and we can tell which software is acting on behalf of the user according to this field. For instance, one User-Agent of Google robots is “Googlebot/2.1” while normal users’ User-Agents in most case would be the names of browsers. Though it is effective to detect simple crawlers by checking these fields, armoured crawlers are able to modify these fields in their requests to escape this type of detection.

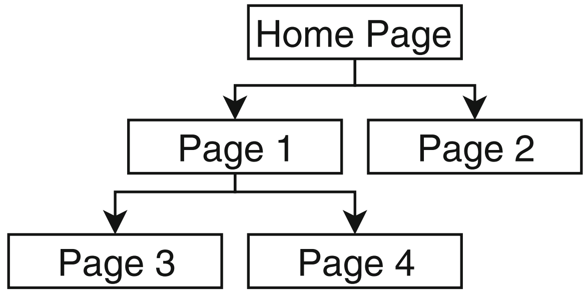

Fig. 1 A Simple Website Structure

URL features recognition. The server can check a session’s URLs of one user to decide if the user is normal. A *session* is a group of continuous access logs that belongs to the same user. For example, some crawlers try to “guess” the URLs for future access, so they will visit non-existent URLs with a high rate within a session. Another example is that when crawlers try to avoid visiting the same page multiple times, the rate of revisiting pages will be low within a session.

Timing-based features recognition. When the server checks the timestamps of a session’s logs, it may derive several features for recognizing the crawler. One example is that some crawlers may access pages more quickly than a human being’s capability. An armoured malicious crawler may discover the upper bound of visiting speed by trail and failure, and then it can set its download rate below this threshold. However, since the distributed crawlers of one attacker have similar timing-based patterns, defenders may detect them by analyzing the similarities of each user’s time series (Jacob et al. 2012).

Crawler trap. A crawler trap can be used to catch crawlers and allure them to make an infinite number of meaningless requests. Crawler traps such as *hidden links* are transparent to normal users but can be seen by crawlers (Barbosa and Freire 2007). An armoured crawler may identify the crawler traps by analyzing the CSS structure of a page.

Threat model and assumptions

This work targets at detecting insiders that have valid user accounts to scrape the privately accessible web contents from the victim websites. The websites that everyone could visit without authentication like *google.com* are out of the scope of this paper. We assume the attacker can be persistent on crawling the entire website in a long time period. Moreover, a number of insiders may coordinate to scrape the website contents. We assume the attacker may control one or multiple user accounts, but the number of compromised accounts is limited due to economic concerns (e.g., paying the premium), labour concern (repeating registration process), or identity constraints (real name or ID number required). The scenario that an attacker controls all user accounts by cracking the website's password database is out of the scope of this paper.

Since we target at detecting inside crawlers, We assume that we can access the source code of the web server and make changes on certain web pages. Furthermore, we assume we have the permission of reading the server's access logs. We assume the web server has one self-owned secret key to encrypt all the URL. Since this key is not shared with any user, users cannot forge valid URL requests to the server.

PathMarker architecture

Figure 2 shows the PathMarker architecture, which consists of two major components, namely, *Website Preprocessing* and *Real-Time Detection*. Website preprocessing contains two major changes, namely, *Adding URL Marker* and *Creating Extended Access Log Table*, on the target website system to help record accurate web page visiting information for real-time detection. We introduce URL marker into website's existing URLs to help accurately track the visiting path of each visitor.

The real-time detection module consists of three components, namely, *Heuristic Detection*, *Path-Based Machine Learning Detection*, and *CAPTCHA*[2] *Verification*. For incoming HTTP requests, PathMarker first extracts the URL markers from the original URLs and collects features for the next step detection. Heuristic detection focuses on investigating the fields of requests as well as checking URL marker integrity.

Path-based machine learning detection focuses on checking both visiting path features from the URL markers and visiting time features from website access logs. After a suspicious user is detected, PathMarker prompts CAPTCHAs to further reduce the false positive rate. If the user fails or activates the CAPTCHA challenges multiple times, it will be marked as a malicious crawler.

Website preprocessing

Given a website to be protected, PathMarker needs to perform two major changes, adding URL markers into all URLs and creating extended access log table for saving visitors' information and URL markers information.

Adding URL marker

PathMarker dynamically adds a URL marker at the end of each hyperlink on each web page. Each URL marker records the page that the URL is retrieved from and the user who requests the URL. By analyzing the information in URL markers, we are able to verify if a user is visiting a URL that is obtained by other users through comparing visitor's user ID with the ID in the marker. We can also learn the causal relationship between two links from the URL markers and accurately determine the width and depth of every access log.

The URL marker should be protected with encryption; otherwise, the malicious user can easily manipulate both the parent page and the requesting user ID in marker to defeat our mechanism. The server can use one self-owned secret key to encrypt the URL with marker before sending it out to the requester. One secret key can be used for all users since distributed crawlers cannot reuse others' ciphertext due to the ID information contained in the URL marker. Moreover, encrypted markers can suppress distributed crawlers by forcing them to repeatedly visit the same page that has different URL markers for different crawlers.

We show one example for URL Marker. One sample URL of the domain A is *http://A.com/B/C.html*. After adding the URL marker into the sample URL, it becomes *http://A.com/B/C.html/mk:B/root.html;User1*, where the appended URL marker is *"mk:B/root.html;User1"*. This marker means this URL is retrieved from the page *"A.com/ B/root.html"* by user *"User1"*. The entire URL after encryption becomes *http://A.com/en:bf37cf8f8f6cb5f3924825 013e3f79c04086d1e569a7891686fd7e3fa3818a8e*.

Creating extended access log table

Most websites maintain access log tables, which are responsible of recording all visitors' accesses information

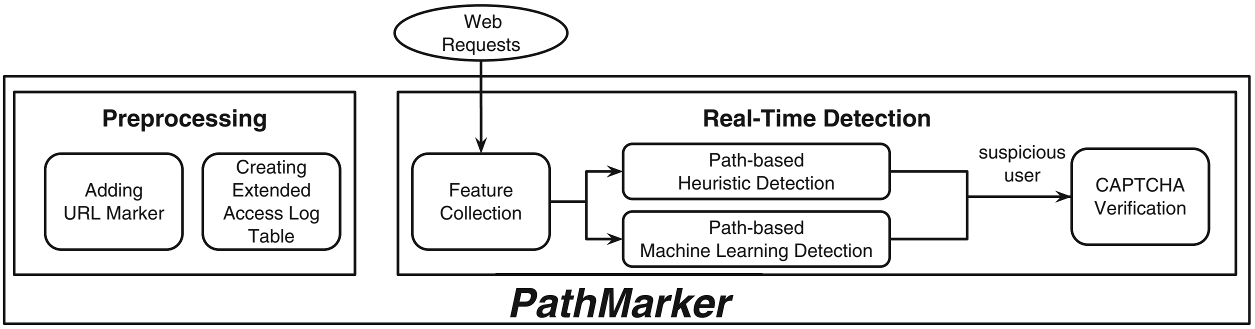

Fig. 2 PathMarker Architecture

such as IP address, the page URL being visited, and timestamp. In order to save more information than the normal access log, PathMarker creates a new table in the database. We call the new table as *Extended Access Log Table*. When a new web request is received, the server decrypts the encrypted URL and parses the plaintext to get the URL marker. Then PathMarker extracts the information in the server's normal access log and insert them into the extended access log table with the visitor's user ID and corresponding marker's information.

Heuristic detection

Heuristic detection module performs basic analysis on the incoming traffic and aims to discover crawlers based on basic traffic flow features such as the referrer, User-Agent, and cookies of all incoming traffic. Besides those general features, this module also performs URL marker integrity checking, a new heuristic detection feature we proposed. Specifically, after the server extracts the marker from the URL, the PathMarker first compares the visitor ID with the information recorded in the URL marker. If the real visitor of this page is not the one recorded in the URL marker, 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 marked multiple times within a time period, we mark this user as a suspicious crawler and prompt it with a CAPTCHA.

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 user IDs in markers.

Machine learning detection

We first define two concepts, namely, *short session* and *long session*, and then derive six new features to be used in machine learning algorithms to separate crawlers from human beings.

Long session and short session

PathMarker calculates the depth and width of an extended access log based on the session that the log belongs to. The length of a session is the number of log entries in the session. A group of continuous access logs belong to a *short session* if the accessing time interval between any two requests is less than a time threshold. As long as a new request is made after the time interval period, the following requests will be grouped in another short session. The length of a short session varies depending on the visiting pattern of the user.

When the number of continuous extended access logs of a visitor reaches a pre-determined length, the machine learning module will be triggered for further analysis. We consider this continuous group belonging to a *long session*. The length of a long session is fixed. A short session only belongs to one long session. When one short session runs across two long sessions, we separate it into two short sessions.

We discover that human beings have different patterns about path depth and width in their short sessions and long sessions while crawlers always behave similarly, which has been verified by our experiments in "Data collection" section.

Figure 3 provides an example on session concepts. It contains 64 extended access logs. The time interval threshold of short sessions is set to 10 s, and the long session's length is set to 60 logs. Note these numbers are adjustable for real-world deployment according to the principles we mentioned above. This piece of extended access logs contains one long session from log#0 to log#59 and three short sessions from log#0 to log#3, log#58 to log#59, and log#59 to log#62. Noted that the user starts its second short session at log#58 and the short sesstion actually continue to log#62; however since the long session stops at log#59, we have to separate these logs into two short sessions.

Session-based features

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 pattern regarding to the

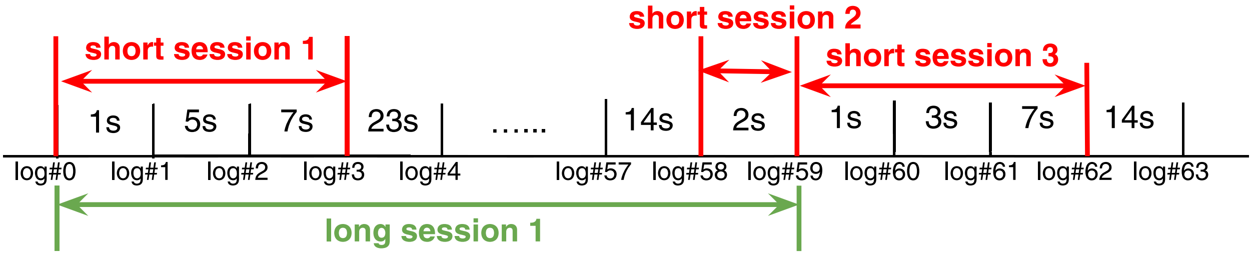

Fig. 3 Long Session vs. Short Session

path and timing. Four features are calculated based on the depth and width of the logs within a session. For each log in a given session, if its parent page is accessed prior to the log within the session, then this log's depth equals to its parent page's depth plus one and it's parent page's width will be incremented by one.

1 *The depth rate of long session* $\frac{max(D_L)}{L_{Long}}$. $max(D_L)$ represents the maximum log's 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.
2 *The width rate of long session* $\frac{max(W_L)}{L_{Long}}$. $max(W_L)$ represents the maximum log's width of a long session. This feature is similar to the depth rate of long session, yet in the width dimension.
3 *Time interval variation of long session* $\frac{Var(I_L)}{\overline{I_L}^2}$. 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. This feature describes the time interval's uncertainty of a user's long session.
4 *The absolute difference between depth rate of long session and depth rate of the 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 maximum log's depth in this longest short session $max(D_S)$ and the longest short session's total length L_{Short}. This feature describes how the depth pattern of one user's long session is different from this user's pattern of the longest short session. This feature is close to zero for crawlers, who have consistent depth rates for their short sessions and long sessions.
5 *The absolute difference between width rate of long session and width rate of the 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 the longest short session* $\frac{Var(I_S)}{\overline{I_S}^2}$. 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. This feature describes the time interval's uncertainty of a user's short session.

Feature 1, 2, 4, 5 are 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. First, one 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. Second, 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 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.

We also have two session-based timing features 3 and 6. Normal users have a small variance of time interval for their short sessions and a large variance for long sessions. When crawlers visit web pages in a more regulated pace, the variance of time interval is small compared to human visitors. Even for an armoured crawler that adds random delay in its visiting pattern, it still can be easily detected since it does not produce different interval variances in a short session and a long session as human beings.

Machine learning

Basically, our features well describe such difference so the SVM model is able to distinguish normal users from crawlers accurately. To get an ideal 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 the method in Jacob et al. (2012) for selecting the data of normal user. First, we use heuristic module to filter out most suspicious users. Then we manually check the logs of all users and remove logs with wrong URL markers.

CAPTCHA challenges

To avoid blocking a normal user, we add CAPTCHA challenges to suspicious users. When one user inputs the CAPTCHA correctly, we remove its suspicious mark. Since there are many types of CAPTCHAs, it's hard for one crawler to prepare itself for recognizing all types of advanced CAPTCHAs. In this case, we can identify the real crawlers that cannot answer correctly. On the other hand, those CAPTCHAs can be solved easily by normal users. Furthermore, we set another counter in "User Information Table" described in "Creating database tables" section. This counter records how many times one user inputs CAPTCHAs and the user would be blocked directly if he activates CAPTCHA module 3 times within one day. With this counter, even a crawler could recognize all CAPTCHAs of our system, it would be detected finally as long as the real-time detection module works.

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.

PathMarker can effectively detect different types of crawlers. First, the heuristic detection module can detect basic crawlers that are not designed for stealthily downloading web data, since these crawlers have much faster download rates than human beings. Second, some timing-aware crawlers can manipulate their download rates or even mimic the timing features of human visitors; however, PathMarker can detect them since their web page access paths are different from normal users. Third, when the crawlers are both path-aware and timing-aware, they can mimic human beings in both access timing patterns and visiting paths. Based on our observations that crawlers and human beings have different visiting patterns in short and long sessions, we still can detect this type of crawlers. Now suppose an armored crawler knows the internal algorithms of our defense mechanism and is capable of deriving the server configuration on the short session and long session. In this case, though the crawler may manager to customize its download behavior to escape our detection, our design can ensure that its crawling rate will be compressed to the level of human beings. In other words, the crawler has to sacrifice its download rate in order to escape from being detected.

PathMarker can suppress multiple distributed crawlers to the visiting rate of a single crawler. By checking the URL marker integrity, PathMarker can easily detect distributed crawlers that share the downloaded URLs in a pool. Malicious crawlers have two options to obtain URLs, either directly collecting URLs from the website or constructing the URLs based on URL patterns of the target system. However, PathMarker can prevent crawlers from constructing URLs since the attacker cannot forge a fake URL that is encrypted with a secret key, which is only known by the web server. Moreover, since we encrypt the URL with the URL marker, one web page may have different URLs in ciphertexts due to the different parent pages. Since the crawler cannot tell the web page content until visiting the URL, we can force the crawler, especially the distributed crawlers, to visit the same web page repeatedly and therefore suppress the crawler's downloading efficiency. We evaluate the suppress results in "Suppressing distributed crawlers" section.

System implementation

We implement PathMarker on an open source online forum website. Meanwhile, we install a monitoring program and a machine learning program at the server side. The monitoring program is responsible for tracking all extended access logs and calculating suspicious users' session-based features. The machine learning program classifies each suspicious user according to those features. We rent the server from DigitalOcean, where the server is configured with 1 GB Memory and 30 GB Disk. The operating system of the server is Ubuntu 14.04 × 64. We deploy the website based on the code of CodeIgnitor (EllisLab) that uses PHP as the server-side scripting language. We use MySQL as the database for saving data of the website.

Our implementation consists of five steps. First, we set up the website and connect the website with the MySQL according to the Installation Instruction of the CodeIgniter's website (EllisLab). Second, we create two MySQL tables named "Extended Access Log Table" and "User Information Table" that are responsible for recording accesses and saving users' information. Third, we modify the server-side scripts. We rewrite all functions on generating URLs to make sure all links of our website have URL markers and are encrypted. We also modify the *main controller* of the website. The *main controller* is the entry of the website and this class is responsible for generating pages after the server receives requests. To handle the encrypted URLs of each request, we add a new function in the *main controller* to decrypt the encrypted URL and separate markers from the entire link. Then the function inspects the user identity. If the user is not a known crawler, we record this access as an extended access log, return the related page to user, and conduct heuristic detection for this request. Fourth, we install and run the monitoring program on the server side. The program is written in C language and it is responsible for finding out suspicious users and passing their long sessions' features to machine learning program. When one long session is marked as suspicious, the monitoring program updates the corresponding user's identity in "User Information Table". Lastly, we install the machine learning program based on the code of LIBSVM (Chang and Lin 2011) at the server side to first detect crawlers and then further identify the type of the crawler. The results are sent to the monitoring program.

Creating database tables

The first table is "Extended Access Log Table", which saves all extended access logs. It has six columns: *Log ID, User ID, User IP, URL, URL marker,* and *timestamp*. For our online forum, whenever the *main controller* confirms the current user ID is not known crawler, PathMarker inserts one entry into this table.

The second table is "User Information Table", which saves all user related information. It has four columns: *User ID, User Identity, Wrong Heuristic Logs*, *User Total Pending Logs* , and *User CAPTCHA Times*. *User ID* is the key of this table while the other three columns are all integer variables and their default values are 0. For column *User Identity*, 0 represents the user is normal user while any number larger than 0 means the user is a crawler. *Wrong Heuristic Logs* records the number of requests that are abnormal for a user regarding to the heuristic detection. It resets to 0 at 12 am each day. If any user has more than 30 wrong logs, then our system sets its *User Identity* as 4. Thus, the server script knows this user is suspicious. *User Total Pending Logs* records the number of logs that one user has generated after last long session. If this column equals to 60, we know this user has generated a new long session and we should inspect this long session. *User CAPTCHA Times* records how many times the user has been checked by CAPTCHA. It resets to 0 at 12 am each day.

Modifying server script

URL generation functions

We use three functions to generate all URLs for the website: *site_url()*, *base_url()* and *redirect()*. These functions can dynamically output URLs according to different inputs. These URL generation functions requires at least two inputs, namely, *type of the page* and *content ID*. The server script predefines several types for web pages such as main page, post page, navigation page, and user information page. This input tells the server what type of page the user is requesting. In the database, each type of the pages has a corresponding table for saving data. Each page belongs to the type has an unique content ID. After taking the inputs, these functions output a normal URL without domain name. For example, for the website "A.com", if the page type is "post" and the content ID is "123", then the output is the string "post/123" and the entire URL is "A.com/post/123".

To add the marker and encrypt the URL, we add a new function *add_marker()* in those URL generation functions. This new function takes three inputs, namely, *the original URL*, *the parent URL*, and *the user ID*. It first generates a marker *"mk:"* | *parent URL* | *";"* | *user ID*. Then the function generates a new URL: *the original URL* | *"/"* | *marker*. Finally, *add_marker()* encrypts the new URL with AES-256-CBC and outputs the ciphertext. The key of AES is saved in the server side. By calling *add_marker()* at the end of those URL generation functions, we can achieve dynamic encrypted URLs with markers.

Website main controller

When the server is running, each received request is passed to the class *main controller*, which initializes the process of generating the page by finding related sub-controllers. The finding process is achieved by the component *Website Router*. Typically there is a one-to-one relationship between a URL string and its corresponding sub-controller class/method(EllisLab). Those sub-controllers fill data from database into the corresponding templates of the URL.

We add a new function called *before_routing()* in the *main controller* to achieve the tasks such as decrypting URL, extracting marker, recording extended access log, and conducting heuristic detection. When the server receives a request, *before_routing()* decrypts the URL by applying AES-256-CBC decryption with the key from the same key file in "URL generation functions" subsection. Then *before_routing()* separates the marker from URL by looking for the string "/mk:" and we save the marker in a string variable. After this step, *before_routing()* gets current user's identity from "User Information Table". If the user's identity value is not 0, then we would check the value of *User CAPTCHA Times* for this user. If *User CAPTCHA Times* is larger than 2, then we would return the message that current ID is blocked and the user should contact administrator for future issues. If *User CAPTCHA Times* is equal or smaller than 2, then we return the CAPTCHA page to the user and increment one to the user's *User CAPTCHA Times*. If the user inputs the CAPTCHA correctly within 30 s then the CAPTCHA page redirects to the target page and we reset the user's identity as 0.

For the cases that user's identity is normal, *before_routing()* passes the part before "/mk:" to the *Website Router* to find the corresponding sub-controller and we record an extended access log in the database. After recording the extended access log, we get the timestamp and IP address from the system default log. *before_routing()* conducts heuristic detection of current request. If any field is abnormal, we increment the current user's *Wrong Heuristic Logs*'s value in the "User Information Table".

Monitoring program

It is a tiny size program that keeps running on the server side to check the "Extended Access Log Table". When this table has a new entry, the monitoring program gets the user ID of this log as "current user". Then it checks the value of current user's *Wrong Heuristic Logs*. If this value is equal to or larger than 30, we set current user's *User Identity* as 4 in "User Information Table". Furthermore, the program increments the value of current user's *User Total Pending Logs* in "User Information Table". If the number of pending logs is equal to the length of a long session, we get the latest 60 logs of this user and take them as a long session. Next the monitoring program calculates the six features of this long session and then pass the features to the machine learning program in the server side. Finally, the monitoring program updates current user's *User Identity* as the returning classification result from machine learning program in "User Information Table".

Machine learning program

Our machine learning program is developed based on a library for support vector machines called LIBSVM (Chang and Lin 2011). The training set of the SVM program is created as described in "Data collection" section. We use two models of the LIBSVM: *one-class SVM* and *C-support vector classification (C-SVC)*. After receiving 6 features of a long session, we first use one-class SVM to decide if the user is a normal user or a crawler. If it is a normal user, we output result "0" into a file that will be read by the monitoring program.

When the one-class SVM identifies a crawler, we use the multi-class SVM named C-SVC to further classify the crawler using the long session information. We number three classes as 1, 2, and 3 in the training set of C-SVC for depth-first crawler, width-first crawler, and random-like crawler, respectively. We also output the specific class number as results into the file for the monitoring program to update the "User Information Table".

System evaluation

Data collection

After implementing our prototype in the server, we publicize the forum where students may exchange information and trade used products. All the experiments have gone through IRB review and all users have been requested to agree with our data policy to sign up. We collect user data from the forum in one month period. We use half of the data for training and the other half for testing. We ensure the user data is generated by real human users through heuristic detection and manual inspection.

Besides the data of normal users, we include crawlers' data in the training set by implementing 6 internal crawlers to crawl the system. We build internal crawlers provided by Frontera (Frontera 0.3), which relies on Scrapy (Scrapy 1.0). The 6 crawlers are depth-first, depth-first with delay, breadth-first, breadth-first with delay, random-like, and random-like with delay. The random-like crawlers randomly choose a link to visit from all links they gathered and put newly gained links in the link pool.

The crawlers with delay means that they 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 crawler data and the testing crawler consists of these 6 internal crawlers and 2 external crawlers Googlebots and Yahoo Slurps.

To better understand the pattern of outside crawlers' algorithms, we publicize all forum contents without requiring a valid user ID and password for authentication. As we remove the user ID authentication requirement, we use the IP address of each visit to replace the user ID in each URL marker. The website is released in one month period. We select the crawlers' logs that can be manually verified by checking the User-Agent field or IP address (Google).

Data analysis

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. First, 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 behaviours in long sessions and short sessions. Second, normal users may show a similar visiting pattern as crawlers in a short session. Most users have a clear depth-first pattern.

Third, if a user visits a group of web pages continuously, we observe that most time gaps between two continuously logs are less than 10 s. Therefore, we set 10 s as the time threshold for a short session. Fourth, the lengths of active users' longest short session are similar. Most active users' longest short sessions contain 20-30 log entries. We 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.

Most time when active user starts a new short session, they visit the site with a different path, so the depth and width will not keep growing across different short sessions. Therefore, we see that the depth rate and width rate of long session are usually smaller than short session for normal users. Since the time threshold of short sessions and the total length of long sessions can be changed according to the web structure, web contents, and web user behaviors, the sessions' settings are adjustable to web server developers/admins who adopt our mechanism. These adjustable settings make the system more difficult for the malicious crawler to understand or escape. Normal users express different behaviours in short sessions and long sessions. Meanwhile, crawlers perform similar behaviours 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.

To illustrate the effectiveness of our models, we show the four path features we use in Figs. 4 and 5, in which each shape is a data point represents a long session. The circles

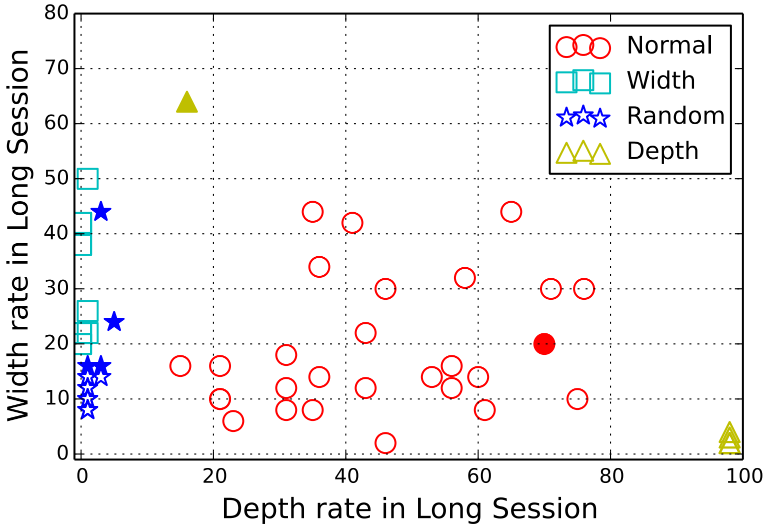

Fig. 4 Differences Between Crawlers and Users about Feature 1 and 2

represent the sessions of normal users, 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 cases our machine learning program misjudge a session's corresponding type. The depth rate and width rate in long sessions are shown in Fig. 4 and the depth rate difference and width rate difference are also shown in Fig. 5. Again, all the hollow points are the sessions we classify correctly. We see that within a long session, crawlers show extrusive path patterns in terms of width and depth. However, normal users have moderate width and depth rate in a long session. Also, the behaviour difference between the longest short session and the corresponding long session is small for crawlers, while the difference is large for normal users.

Performance evaluation

We evaluate three aspects of PathMarker, namely, the accuracy of crawler detection of PathMarker, the capabilities of PathMarker to reduce the efficiency of distributed crawlers, and the performance overhead added to the web system. Moreover, we conduct a case study on an external crawler, Googlebots.

Detection capabilities

We first study the effectiveness and implication of marker integrity checking. Then we show that our SVM model is

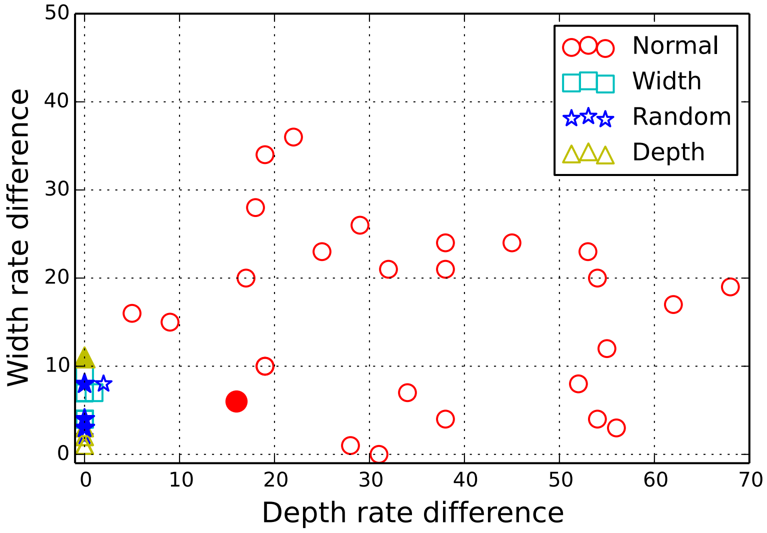

Fig. 5 Differences Between Crawlers and Users about Feature 4 and 5

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 in the past. The number of all log entries of logged-in users is 2608, 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 no 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 "Googlebots – a case study" section, 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 find two external crawlers. One is Googlebot and the other one is Yahoo Slurp. We believe they are the only two search engines that try to crawl our system by manually checking all public visitors that generate relatively abundant access logs.

We notice that these external crawlers are based on different crawling techniques. Both Yahoo and Google use distributed crawlers, which are verified by verifying the User-Agent 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 "Googlebots – a case study" section.

Table 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. For all 3.26% crawlers' long sessions that have been misjudged as normal user long sessions, there is at least one other long session of the same crawler that implies the visitor is not human being. Thus, we do not miss any crawler even we misjudge one crawler's behaviour for at most two long sessions. After identifying a crawler, we classify the path-patterns of the crawler, as shown in Table 1. All three types of crawlers are correctly identified with over 90% accuracy. Type 0 represents normal users. Type 1, 2, and 3 represents crawlers that expose extrusive breadth-first, depth-first, and random-like crawling features, respectively. We see that most bots' paths can be fit into the three patterns we define for crawlers. Note that Google and Yahoo crawlers expose random-like visiting paths since they use popularity concerned crawling algorithms.

Table 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%

Suppressing distributed crawlers

Suppressing crawlers mainly contains two cases. First, a single crawler may repeatedly visit the same page if the page is retrieved from different parent 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. We define these fixed URLs since most websites put links in the header, footer, and side bars of a website such as homepage and account management button and these links will not change for different pages. 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 3333, which is a third of the number of pages. Each page corresponds to a number between 0 and 9999. The probability of a page being selected is $P_n = (cdf(n) - cdf(n-1)) \times 2$, where n is the page number and cdf is the cumulative density function.

We conduct two sets of simulation experiments to study the impacts of PathMarker on distributed crawlers. 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 shows the efficiency about visiting first 100 new pages for crawlers with different total number of workers. When the crawler only has one worker, 8.2% of its downloads are duplicates. 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. In general, distributed crawlers get the URLs of one page several times from different workers

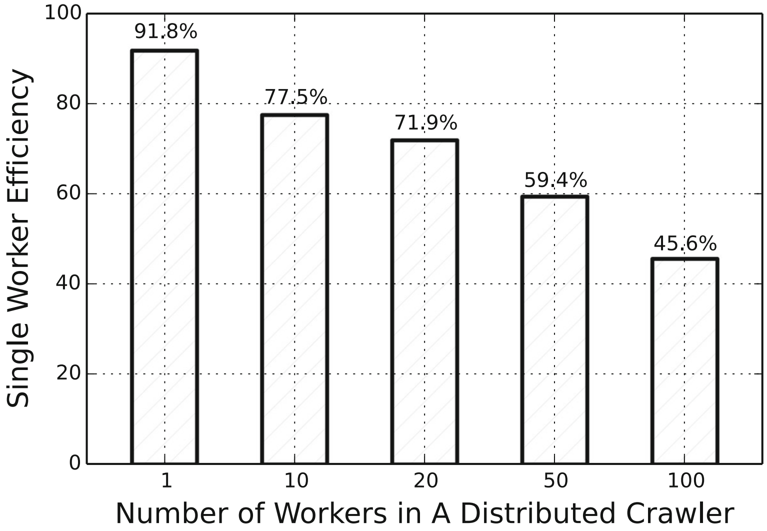

Fig. 6 Suppressing Distributed Crawlers

but it cannot detect it since the encrypted URLs are different.

Figure 7 illustrates the entire work of a 10-worker distributed crawler need to download a certain percentage of the entire website. 100% work means the workload for a crawler who always visits new page and extra work means this crawler visits repeated pages. We see that 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 2781% more queries are required to crawl 95% of all content, which indicates a 96.5% crawling power waste. It shows that even if armoured 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 are larger.

System overhead

Our defence 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

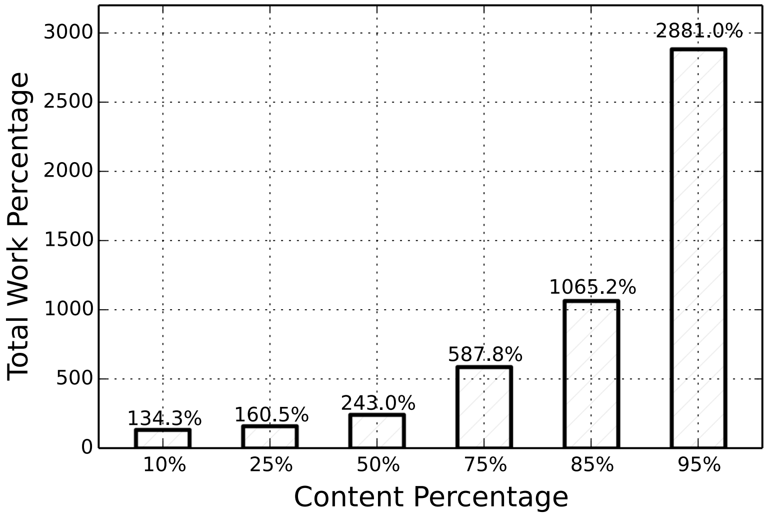

Fig. 7 Overhead for Distributed Crawlers

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 1000 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 ms and the average time needed to generate a page with PathMarker is 41.5ms. This increase is acceptable since for a normal user, they can barely feel a 10 ms difference and the number of links in each page is large enough for the website that contain confidential documents.

Googlebots – a case study

After we publicize the online forum, we notice Google search engine is actively visiting it by checking visitors' User-Agent and IP address lookup. Since Googlebots is the largest crawler that uses sophisticated and evolving algorithm, we study the behaviour of Googlebots to show how the both layers of PathMarker response with crawlers that armed with unknown algorithm and how PathMarker suppresses the crawlers.

During the one-month data collection, we discover Googlebots from over 50 IP addresses, which indicate that at least 50 crawling workers are crawling our system. Google crawler is not designed to escape from anti-crawler mechanisms, and it does not hide itself in HTTP requests by stating its identity as Googlebots. However, it does have an efficient rate control mechanism to avoid introducing a large overhead on web servers or being banned due to high visiting speed. With manual checking, we note that each worker waits for some time between two consecutive requests from several seconds to several hundred of seconds. Moreover, Googlebots are cooperating among a large number of workers, decreasing the number of total visiting pages for each single worker.

Since most existing anti-crawler mechanisms use the visiting rate or total visiting number as the key factors to detect crawlers, Googlebots can escape these mechanisms successfully if the Googlebots hide its identity in the User-Agent field. Because of the features of Googlebots, we consider Googlebot as a representative of the distributed crawlers that have rate control mechanism. Therefore, by presenting the detection results on Googlebots, we can show how well PathMarker performs on those distributed crawlers that visit web pages slowly.

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 only 9 workers are responsible for most of the requests (they visited more than 25 pages). 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 scrapes the content of our system using a distributed crawler consisting of 9 active workers[3].

Now we explain how Googlebots can be detected by our heuristic detection. Though the Googlebot is able to escape visiting rate detection, we can easily discover that it is a distributed crawler by URL marker integrity checking which belongs to the heuristic detection of PathMarker. The total number of URLs with wrong URL markers are 12,271, which is 62% of all requests by Googlebots. This result means that 62% of the URLs a worker visits is collected by other workers. We show several examples of wrong URL markers in Table 2. From the last field of the URL markers we see that the collector of this URL marker is from a different IP address than the visitor IP address, so we could tell one IP is abnormal if it happens multiple times.

We also analyze the path features of Googlebots to illustrate how these features differentiate them from normal users. We gather totally 381 long sessions from Googlebots, which means the SVM module is invoked 381 times to analyze the owner of each long session. Figure 8 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

Table 2 Detecting Googlebots by checking URL markers

Visitor IP	URL	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

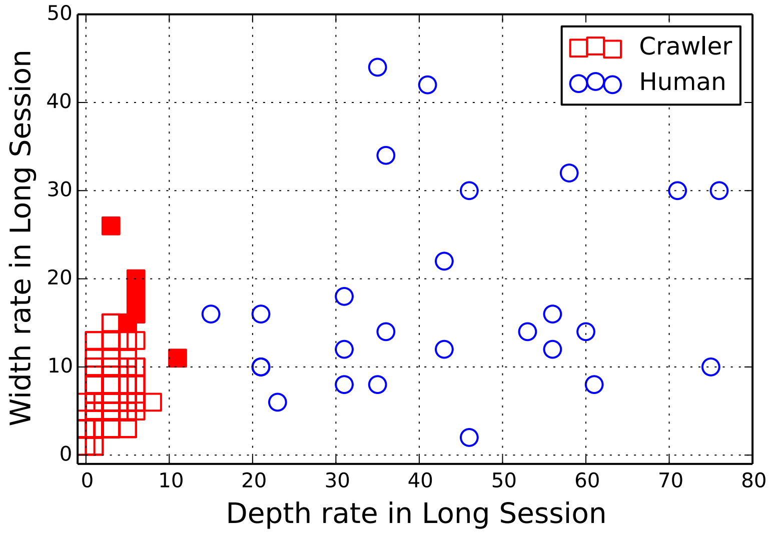

Fig. 8 Depth and width rate in long session for Google Bots

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 have at least one long sessions will be prompted a CAPTCHA in a full functionality PathMarker system.

We also show how PathMarker suppresses the efficiency of Googlebots. Among the 19,844 requests, only 1010 pages (around 40% of total pages) are unique, which means that Google has wasted almost 95% of its crawling power on visiting repeated pages.

Discussion and limitations

Usability of URL marker

In PathMarker, the path and URL marker information in URLs are encrypted to protect the URL marker security and lower the efficiency of distributed crawlers. Meanwhile, since normal users cannot know the plaintext of URLs, it is difficult for the users to remember the URLs or infer the content of the web page. However, users can save the encrypted URLs in bookmarks and reopen them later. Also, since current web pages typically have their own titles to identify the content, the plaintext of URL may not be necessary.

Another issue is the sharing of URLs between normal users. However, we can solve this problem by setting a relatively high threshold for users to visit URLs shared from others without being tagged as crawlers. Meanwhile, we still can identify distributed crawlers in a short time. Also, since PathMarker targets on protecting valuable private contents like confidential documents, the case that users share URLs will not happen many times.

Deployability of PathMarker

There are many ways to implement the web servers and generate web pages. In general, there are two types of web pages: static web pages and dynamic web pages. For static web pages, PathMarker can automatically make the required modification using a simple script to preprocess links before publicize the website. However, for dynamic web pages, since there are various server-side scripting languages like PHP and Python to generate dynamic website structures, it is difficult, if not impossible, to design a generic tool to automatically deploy PathMarker for all types of web servers. However, after studying two famous open-source website framework, Discuz! (Comsenz Inc.) and CodeIgniter (EllisLab), we discover that there are usually one or two most common functions to generate most URLs and the total number of such functions is usually less than 10. Therefore, with some light-weight customization, most dynamic web page servers are able to integrate PathMarker in their system.

Related work

Both crawlers and anti-crawler mechanisms evolve in their arms race. 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 armoured 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 (Olston and Najork 2010; Kausar et al. 2013). For instance, (Doran et al. 2013) investigates the difference between resources such as images crawler. Dikaiakos et al. (2005) focuses on analyzing the features and preferences on search engine crawlers. Some works have been done to detect crawlers from a large scale network service (Yu et al. 2010; Lee et al. 2009). Frontier that can determine the crawling behaviour becomes a core component of crawlers (Olston and Najork 2010). On the other hand, Frontier has been adopted by attackers to help crawlers achieve better crawling results (Baeza-Yates et al. 2005; De Groc 2011; Jin et al. 2013a; Batsakis et al. 2009). For instance, (Jin et al. 2013a) proposes a novel solution for mimicking human behaviours according to the human observational proofs so the new crawler could escape the detection of other defence systems.

There are several research works targeting at defending crawlers. One recently work (Stringhini et al. 2015) observes that cyber-criminals might misuse several accounts on stealing sensitive information and they propose a solution to capture these crawlers by mapping between an online account and an IP address. The most popular trend in this area is utilizing machine learning technique to detect sophisticated crawlers. Researchers have developed numerous anti-crawling artifacts that explore machine learning techniques to suppress the efficiency of crawlers (Yu et al. 2010; Lee et al. 2009) or even completely block them (Stevanovic et al. 2013; Jacob et al. 2012; Tan and Kumar 2004; Doran and Gokhale 2011), based on the observations that crawlers behave differently from human beings (Tan and Kumar 2004; Jacob et al. 2012). 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 (Tan and Kumar 2004), 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 (Zhang et al. 2013; Aghamohammadi and Eydgahi 2013; Guo et al. 2005; Bomhardt et al. 2005; Yu et al. 2010; Lee et al. 2009; Jacob et al. 2012). For example, Jacob et al. (2012) 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 (Stevanovic et al. 2012;2013). 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 Jacob et al. (2012). There are also other comprehensive features that profile the visiting behaviour of crawlers, including traffic timing shape (Jacob et al. 2012), page popularity index, standard deviation in visiting depth Stevanovic et al. (2013), clickstream related features (Lourenço and Belo 2007; Ahmadi-Abkenari and Selamat 2012) and some special features for the Bayesian network to recognize crawlers (Stassopoulou and Dikaiakos 2006; 2009; Suchacka and Sobkow 2015). Some others are even trying to understand the crawlers to capture them (Xie et al. 2014; Rubinstein et al. 2009).

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. (2013) 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 (2004) 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 another essential problem. Setting traps in pages is a common method to catch crawlers (Barbosa and Freire 2007). 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. (2006) 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 (Kluever and Zanibbi 2008; Gossweiler et al. 2009). Gossweiler et al. (2009) also requires people to recognize more complex content of image such as the orientation.

Conclusions

In this paper, we develop an anti-crawler system named PathMarker to help server administrators capture and suppress stealthy persistent crawlers who may collude to download the contents of servers. PathMarker can 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 by mimicking human beings, their crawling efficiency can be dramatically suppressed to the level of human beings. We evaluate PathMarker on an online forum website. We are able to detect 6 popular crawlers with a high accuracy as well as external crawlers such as Yahoo and Google bots.

Endnotes

[1] If the user accesses web link A from page B, then B is the parent page of link A.

[2] CAPTCHA: Completely Automated Public Turing test to tell Computers and Humans Apart

[3] 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.

Abbreviations

CAPTCHA: Completely automated public turing test to tell computers and humans apart; HTTP: Hypertext transfer protocol; SVM: Support vector machine; URL: Uniform resource locator

Acknowledgments

We thank all reviewers for their valuable comments and suggestions.

Funding

This work is supported by U.S. Office of Naval Research under grants N00014-16-1-3214 and N00014-16-1-3216.

Availability of data and materials

We presented all available experiment setting in "System implementation" and "System evaluation" sections. However, the visiting logs of real users cannot be shared since our data policy promised the data will only be used in this study.

Authors' contributions

SW: design and implement the system. YL: experiment. All authors: review and revise the manuscript. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Author details

[1]Department of Computer Science College of William and Mary, Williamsburg 23187-8795, VA, USA. [2]Department of Information Sciences and Technology George Mason University, Fairfax 22030, VA, USA.

Received: 12 November 2018 Accepted: 15 January 2019
Published online: 20 February 2019

References

Aghamohammadi A, Eydgahi A (2013) A novel defense mechanism against web crawlers intrusion. In: Electronics, Computer and Computation (ICECCO), 2013 International Conference On. IEEE, Ankara. pp 269–272

Ahmadi-Abkenari F, Selamat A (2012) An architecture for a focused trend parallel web crawler with the application of clickstream analysis. Inf Sci 184(1):266–281

Baeza-Yates R, Castillo C, Marin M, Rodriguez A (2005) 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. ACM, New York. pp 864–872

Barbosa L, Freire J (2007) An adaptive crawler for locating hidden-web entry points. In: Proceedings of the 16th International Conference on World Wide Web. ACM, New York. pp 441–450

Batsakis S, Petrakis EG, Milios E (2009) Improving the performance of focused web crawlers. Data Knowl Eng 68(10):1001–1013

Bomhardt C, Gaul W, Schmidt-Thieme L (2005) Web Robot Detection-preprocessing Web Logfiles for Robot Detection. Springer, Berlin, Heidelberg

Chang C-C, Lin C-J (2011) LIBSVM: A library for support vector machines. ACM Trans Intell Syst Technol 2:27–12727. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

Comsenz Inc. Discuz Forum. http://www.discuz.net/forum.php. Accessed 23 Oct 2018

De Groc C (2011) 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. IEEE Computer Society, Lyon. pp 497–498

Dikaiakos MD, Stassopoulou A, Papageorgiou L (2005) An investigation of web crawlerbehavior:characterization and metrics. Comput Commun 28(8):880–897

Doran D, Gokhale SS (2011) Web robot detection techniques: overview and limitations. Data Min Knowl Disc 22(1-2):183–210

Doran D, Morillo K, Gokhale SS (2013) 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. ACM, New York. pp 1374–1380

EllisLab CodeIgniter. https://codeigniter.com/. Accessed 23 Oct 2018

Frontera 0.3. http://frontera.readthedocs.org/en/latest/index.html. Accessed 23 Oct 2018

Gianvecchio S, Wu Z, Xie M, Wang H (2009) 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. ACM, New York. pp 256–268

Gianvecchio S, Xie M, Wu Z, Wang H (2008) Measurement and classification of humans and bots in internet chat. In: Proceedings of the 17th USENIX Conference on Security Symposium. USENIX Association, Berkeley. pp 155–169

Google Verifying Googlebot. https://support.google.com/webmasters/answer/80553?hl=en. Accessed 23 Oct 2018

Gossweiler R, Kamvar M, Baluja S (2009) What's up captcha?: a captcha based on image orientation. In: Proceedings of the 18th International Conference on World Wide Web. ACM, New York. pp 841–850

Guo W, Ju S, Gu Y (2005) 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. IEEE, Coventry Vol. 1. pp 302–306

Jacob G, Kirda E, Kruegel C, Vigna G (2012) Pubcrawl: Protecting users and businesses from crawlers. In: USENIX Security Symposium. USENIX Association, Berkeley. pp 507–522

Jin J, Offutt J, Zheng N, Mao F, Koehl A, Wang H (2013a) Evasive bots masquerading as human beings on the web. In: Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference On. IEEE, Budapest. pp 1–12

Kausar MA, Dhaka V, Singh SK (2013) Web crawler: a review. Int J Comput Appl 63(2):31–36

Kluever KA, Zanibbi R (2008) Video captchas: usability vs. security. IEEE Western New York Image Processing Workshop, Rochester

Lee J, Cha S, Lee D, Lee H (2009) Classification of web robots: An empirical study based on over one billion requests. Comput Secur 28(8):795–802

Lourenço A, Belo O (2007) Applying clickstream data mining to real-time web crawler detection and containment using clicktips platform. In: Decker R, Lenz H-J (eds). Advances in Data Analysis, Studies in Classification, Data Analysis, and Knowledge Organization. Springer, Berlin. pp 351–358

Network D (2018) 2018 Bad Bot Report. https://resources.distilnetworks.com/whitepapers/2018-bad-bot-report. Accessed 23 Oct 2018

Olston C, Najork M (2010) Web crawling. Found Trends Inf Retr 4(3):175–246

Park K, Pai VS, Lee K-W, Calo SB (2006) Securing web service by automatic robot detection. In: USENIX Annual Technical Conference, General Track. USENIX Association, Berkeley. pp 255–260

Rubinstein BIP, Nelson B, Huang L, Joseph AD (2009) Antidote: Understanding and defending against poisoning of anomaly detectors. In: In Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement. ACM, New York

Scrapy 1.0. http://scrapy.org/. Accessed 23 Oct 2018

Sexton P The Googlebot guide. https://varvy.com/googlebot.html. Accessed 23 Oct 2018

Snowden used common web crawler tool to collect NSA files. https://www.rt.com/usa/snowden-crawler-nsa-files-227/. Accessed 23 Oct 2018

Stassopoulou A, Dikaiakos MD (2006) Crawler detection: A bayesian approach. In: Internet Surveillance and Protection, 2006. ICISP'06. International Conference On. IEEE, New York. pp 16–16

Stassopoulou, A, Dikaiakos MD (2009) Web robot detection: A probabilistic reasoning approach. Comput Netw 53(3):265–278

Stevanovic D, An A, Vlajic N (2012) Feature evaluation for web crawler detection with data mining techniques. Expert Syst Appl 39(10):8707–8717

Stevanovic D, Vlajic N, An A (2013) Detection of malicious and non-malicious website visitors using unsupervised neural network learning. Appl Soft Comput 13(1):698–708

Stringhini G, Mourlanne P, Jacob G, Egele M, Kruegel C, Vigna G (2015) Evilcohort: detecting communities of malicious accounts on online services. In: 24th USENIX Security Symposium (USENIX Security 15). USENIX Association, Berkeley. pp 563–578

Suchacka G, Sobkow M (2015) Detection of internet robots using a bayesian approach. In: Cybernetics (CYBCONF), 2015 IEEE 2nd International Conference On. IEEE, Gdynia. pp 365–370

Tan P-N, Kumar V (2004) Discovery of web robot sessions based on their navigational patterns. In: Intelligent Technologies for Information Analysis. Springer, Berlin, Heidelberg. pp 193–222

Xie G, Hang H, Faloutsos M (2014) Scanner hunter: Understanding http scanning traffic. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '14. ACM, New York. pp 27–38

Yahoo What is Slurp. https://help.yahoo.com/kb/SLN22600.html. Accessed 23 Oct 2018

Yu F, Xie Y, Ke Q (2010) Sbotminer: large scale search bot detection. In: Proceedings of the Third ACM International Conference on Web Search and Data Mining. ACM, New York. pp 421–430

Zhang D, Zhang D, Liu X (2013) A novel malicious web crawler detector: Performance and evaluation. Int J Comput Sci Issues (IJCSI) 10(1):31–36