Protecting Web Contents against Persistent Distributed Crawlers

Shengye Wan
College of William and Mary
Williamsburg, Virginia 23187–8795
Email: simonwan@cs.wm.edu

Yue Li
College of William and Mary
Williamsburg, Virginia 23187–8795
Email: yli@cs.wm.edu

Kun Sun
George Mason University
Fairfax, Virginia 22030
Email:ksun3@gmu.edu

***Abstract*—Web crawlers have been misused for several malicious purposes such as downloading server data without permission from the website administrator. In this paper, based on one observation that normal users and malicious crawlers have different short-term and long-term download behaviors, we develop a new anti-crawler mechanism called PathMarker to detect and constrain persistent distributed crawlers. For each URL, by adding a marker to record its parent page that leads to the access to this URL and the user identity who accesses this URL, we can not only perform more accurate heuristic detection and Support Vector Machine (SVM) based machine learning detection to detect malicious crawlers at an earlier stage, but also dramatically suppress the efficiency of crawlers before they are detected. We deploy our approach on a forum website, and the evaluation results show that PathMarker can quickly capture all 6 open-source and in-house crawlers.**

I. Introduction

The 2016 bot traffic report from Incapsula points out that bots account for 51% of all website traffic, and malicious bots contribute almost 30% of the web traffic [10]. Meanwhile, an increasing number of new algorithms have been adopted by malicious crawlers to increase the download efficiency and decrease the chance of being detected [12]. To protect confidential documents or sensitive data, most websites require users to authenticate themselves. Though this layer of authentication can successfully block external 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 [3]. To prevent from being easily detected, those insiders could launch a persistent download in a long time or corporate with other insiders.

To detect and constrain malicious crawlers, researchers have developed a number of anti-crawler mechanisms [17], [19], [21], 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 detect abnormal crawling behaviors [21]. However, they cannot guarantee to detect crawlers that are able to manipulate those observed features. Machine learning detection mechanisms can detect malicious crawlers based on the different visiting patterns of normal users and malicious crawlers [17]. Most recent anti-crawlers mechanisms combine both techniques to better defeat malicious crawlers [19], [11]. However, it is still a challenge to detect and constrain armoured inside crawlers that can collude in a persistent web scraping.

In this paper, we develop a new anti-crawler mechanism called *PathMarker* that aims to detect and constrain persistent distributed inside crawlers. Moreover, we manage to accurately detect those armoured crawlers at their earliest crawling stage. The basic idea is based on one key observation that crawlers have similar accessing patterns for both short-term and long-term downloads, while human beings have obvious accessing patterns only in the short term and have no certain patterns in the long term. It is well known that existing path based anti-crawler solutions have the difficulty in accurately calculating the depth and width of one access [19]. Assume one user just visits a link. We do not know that from which page the user obtains this link so we can only guess the depth and width of the access. We solve this problem in PathMarker by automatically appending a marker to each web page URL. We call the marker as *URL marker*, which records the *parent page* that leads to the access to this link and the user ID who accesses this link. To further protect the URL markers from being misused, we encrypt the URL markers. URL markers enable us to quickly detect distributed crawlers that share links with other workers through a user ID mismatch. Moreover, with the aid of URL marker, we develop a Support Vector Machine (SVM) based machine learning detection to model and detect the different patterns of URL visiting paths between human beings and malicious crawlers. URL markers can also decrease the crawler's download efficiency since the crawlers may download the same web page multiple times.

We develop a PathMarker prototype on an online open source forum website. We train SVM models based on one-month access log data, and test the model using 6 open-source crawlers and one set of normal user data. The experimental results show that our anti-crawler technique can effectively detect all those crawlers.

II. Related Work

Web crawlers have been studied for a long time [15], [14]. For instance, [8] investigates the difference between resources such as images crawler. [7] focuses on analyzing the features and preferences on search engine crawlers. Some research works have been done to detect crawlers from a

978-1-4673-8999-0/17/$31.00 ©2017 IEEE

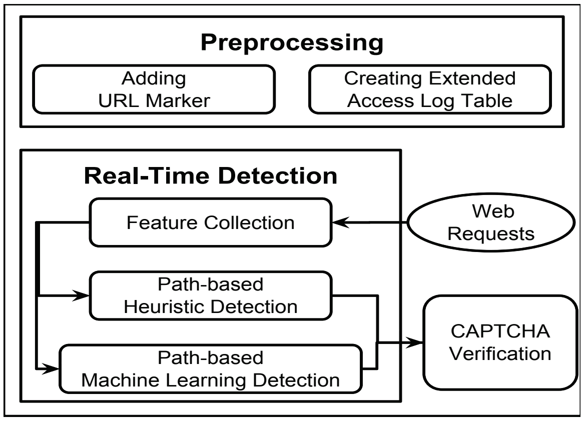
Preprocessing
Adding URL Marker
Creating Extended Access Log Table
Real-Time Detection
Feature Collection
Web Requests
Path-based Heuristic Detection
CAPTCHA Verification
Path-based Machine Learning Detection

Fig. 1: PathMarker Architecture

large scale network service [21]. Frontier that can determine the crawling behavior has been adopted by attackers to help crawlers achieve better crawling results [12]. For instance, [13] proposes a novel solution for mimicking human behaviors according to the human observational proofs so the new crawler could escape the detection of other defence systems.

The most popular trend in this area is utilizing machine learning technique to detect sophisticated crawlers. Researchers have developed numerous anti-crawling artifacts to defeat crawl [21], [17], [11], [19]. One challenge for machine learning based solutions is to select a set of effective features to train the machine learning model. In one of the earliest work [19], 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 [4], [21], [11]. For example, Jacob et al. [11] use multiple timing features to characterize crawlers. Numerous features have been proposed and proven to be effective for specific use cases [17]. The usage of request-related features such as the percentage of GET request and POST request, the percentage of error responses, and the total number of pages requested has been proposed in [11]. There are also other comprehensive features that profile the visiting behavior of crawlers, including traffic timing shape [11], page popularity index, standard deviation in visiting depth [17], clickstream related features[5] and some special features for the Bayesian network to recognize crawlers [18]. Some others are even trying to understand the crawlers to capture them [20], [16].

III. Threat Model and Assumptions

We target 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 (e.g., www.wikipedia.org) are out of the scope of this paper. The attacker can be persistent on crawling the entire website in a long time period. We assume the attacker may control one or multiple user accounts, and multiple insiders may collude to scrape the website contents.

IV. PathMarker Architecture

Figure 1 shows the PathMarker architecture, which consists of two major components, namely, *Website Preprocessing* and *Real-Time Detection*. Website Preprocessing adds URL markers into website URL links and supports website server to log accurate website visiting information. Real-Time Detection performs crawler detection by inspecting incoming HTTP requests. When a request arrives, PathMarker first extracts the URL markers from the original URLs and collects features for crawler detections. Then the system conducts both heuristic detection and path-based machine learning detection module to find suspicious users. After a suspect is detected, PathMarker prompts CAPTCHAs to further reduce the false positive rate.

A. Website Preprocessing

Given a website to be protected, PathMarker needs to add URL markers into all URLs and extend access log table for retrieving path information.

1) 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 user ID in URL markers, we are able to verify if a user is visiting a URL that is obtained by other users. We can also learn the causal relationship between two links according to URL markers and then accurately determine the width and depth of each access link.

URL markers are protected with encryption to prevent malicious users from manipulating the URL markers. The server uses 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.

We show one example of URL Marker here. Given one URL on the domain A: *http://A.com/B/C.html*, after adding the URL marker, it changes to: *http://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"* by user *"User1"*. The entire URL after encryption is: *http://A.com/en:bf37cf8f8f6cb5f3924825013e3f79c04086d1e569a7891686fd7e3fa3818a8e*.

2) Creating Extended Access Log Table: PathMarker creates a new log table in the database to record more information than the normal access log. We call this 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 inserts a log with the visitor's user ID and corresponding marker's information into the extended access log table.

B. Heuristic Detection

This 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, PathMarker also performs URL marker integrity checking, a new heuristic detection feature we proposed. Specifically, after the server extracts the marker from the URL, it first compares the visitor ID with the one recorded in the URL marker. If these two IDs are not the same, then we flag this log entry and mark this user as a potential crawler. If the user's logs are flagged multiple times within a time period, we mark this user as a suspect and prompt it with a CAPTCHA.

C. Machine Learning Detection

This module classifies each user into four categories: normal user, depth-first crawler, width-first crawler, and random-like crawler. 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. *Breadth-first crawler* visits all links of a page before visiting links of another page. For all the crawlers whose crawling algorithms are path irrelevant, we classify them as *random-like crawler*.

1) Supervised SVM: We adopt the supervised Support Vector Machine (SVM) to build the machine learning model in PathMarker. The machine learning model is first trained using tagged data from normal users and crawlers. We develop six new session-based features used in the machine learning algorithms to separate crawlers from human beings. Before introducing these new features, we first define two closely related concepts, namely, *short session* and *long session*. Then we present how to use these concepts to differentiate crawlers from users. Finally we would present the details of our new features.

2) Long Session, Short Session, and Their Attributions: A group of continuous access logs is called as a session. The length of a session is the number of log entries in the session. A session is a *short session* if the accessing time interval between any two continuous requests is less than a time threshold. When 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 logs of a visitor reaches a pre-determined length, the machine learning module will be triggered for further analysis. We consider this fixed-length continuous group as a *long session*. 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.

For each session, we care about the following attributions:

a) Log's depth and width: For each log in a given session, if the link in its marker has been accessed prior to the log within the session, then this log's depth equals to its the marker's link's depth plus one and the marker's link's width will be incremented by one.

b) Session's depth rate and width rate: A session's depth rate equals to the length of the session divides the maximum log's depth within it. Similar, it's width rate equals to the length divides the maximum log's width.

3) Key Observations of Sessions: By analyzing the training set we used, we find 4 major differences between crawlers and users regarding to their short sessions and long sessions:

a) The lengths of active users' longest short sessions are similar. Most active users' longest short sessions contain 20-30 log entries. In this case, we set the length of a long session as 60 since we recommend the long session to be twice as a user's average longest short session.

b) For a short session, human beings usually have one of two obvious patterns. First, one user may open multiple web pages at one time, so the width rate of this short session would be large. Second, the user prefers to jump to another page after he or she takes a glance at one page, so this short session will have large depth rate.

c) For a long session, normal users' long sessions would have smaller depth rate or smaller width rate than it's longest short 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 one long session of crawler would always have similar rates as the short sessions within it.

d) The time intervals of a user's short session would be similar and the time intervals of a user's long session would be vary from a large range. Meanwhile, crawlers' time intervals of short session and long session always have similar range even for an armored crawler that adds random delay in its visiting pattern.

4) Session-based Features: Based on the observation in section IV-C3, we identify six new features to train the SVM-based detection model.

a) $\frac{max(D_L)}{L_{Long}}$. $max(D_L)$ represents the maximum log's depth in a long session and L_{Long} represents the fixed length of the long session. This feature is the depth rate of a long session. It describes how visitors keep reaching new pages as deep as possible in a long session.

b) $\frac{max(W_L)}{L_{Long}}$. $max(W_L)$ represents the maximum log's width of a long session. This feature is the width rate of a long session. Its similar to feature 1, yet in the width dimension.

c) $\frac{Var(I_L)}{\overline{I_L}^2}$. (I_L) are time intervals within the long session and $Var(I_L)$ is the variation of the time intervals. This feature describes the time intervals' uncertainty of a user's long session.

d) $\left|\frac{max(D_L)}{L_{Long}} - \frac{max(D_S)}{L_{Short}}\right|$. The absolute value of the different between the depth rate of a long session and the longest short session in it. 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.

e) $\left|\frac{max(W_L)}{L_{Long}} - \frac{max(W_S)}{L_{Short}}\right|$. This feature describes width pattern difference between the longest short session and a long session of a user.

f) $\frac{Var(I_S)}{\overline{I_S}^2}$. This feature is similar to feature 3; however, the time interval is computed based on the longest short session in the long session. This feature describes the time interval uncertainty of a user's short session.

V. Security Analysis

PathMarker can effectively detect different types of crawlers. First, the heuristic detection module can detect crawlers have any one of these features: download fast, any field such as User-Agent is abnormal, or share links with other crawlers. Second, for crawlers that are capable of constraining their behaviors to avoid all detection above, PathMarker can detect them since their web page access patterns are different from normal users.

Suppose an armored crawler knows our detailed defence mechanisms and may be capable of adapting its download behavior to escape our detection. In this case, we can successfully compress its crawling rate since each crawling worker can only use the links retrieved by itself and it would download multiple copies of the same web page.

VI. System Implementation

We implement PathMarker on an open source online forum website. Meanwhile, we install a monitoring program at the server side to track all extended access logs and conduct SVM detection. We deploy the website based on the code of CodeIgnitor [9] that uses PHP as the server-side scripting language. We set up the website and connect the website with one MySQL database. After that, three steps are performed to deploy PathMarker on the online forum website.

A. Creating Database Tables

We create two new 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 server 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*, and *User Total Pending Logs*. 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. *User Total Pending Logs* records the number of logs that one user has generated after last long session.

B. Modifying Server Script

1) 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.

To add URL marker and encrypt the URL, we add a new function *add_marker()* in those URL generation functions. The function *add_marker()* takes three inputs, namely, *the original URL*, *the parent URL*, and *the user ID*. After using AES-256-CBC to generate a marker: *"mk:"* | *parent URL* | *";"* | *user ID*, it outputs a new URL: *the original URL* | *"/"* | *marker*. The AES key is only stored on the server side.

2) 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*.

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 with the same key in VI-B1. 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()* obtains current user's identity from "User Information Table". If the user's identity value is not 0, we return the CAPTCHA page to the user and block the user if he or she cannot input the CAPTCHA. If the user inputs the CAPTCHA correctly within 30 seconds then the CAPTCHA page is redirected 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. *before_routing()* conducts heuristic detection of current request. If any field is abnormal, we increment the current user's *Wrong Heuristic Logs*'s value.

C. Monitoring Program

This program 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 60, 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 call the machine learning function which is developed based on a library for SVM called LIBSVM [6]. We use two models of the LIBSVM: *one-class SVM* and *C-support vector classification (C-SVC)*. 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". Otherwise, We also output the specific class number (1, 2, and 3 for depth-first crawler, width-first crawler, and random-like crawler, respectively) as the result. Finally, the monitoring program updates current user's *User Identity* as the output number.

VII. System Evaluation

A. Data Collection

After implementing our prototype, we publicize the online forum to the students at College of William and Mary. We collect user data of students in one month period. We use half

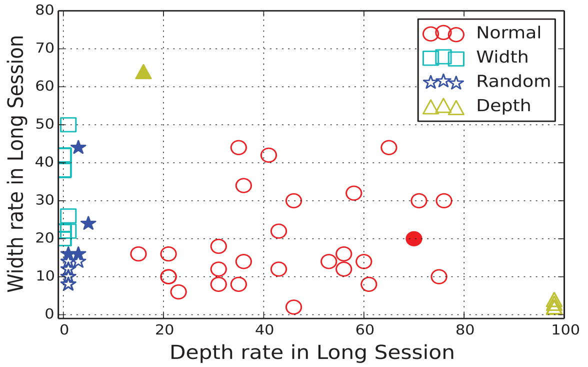

Fig. 2: Differences on Feature 1 and 2

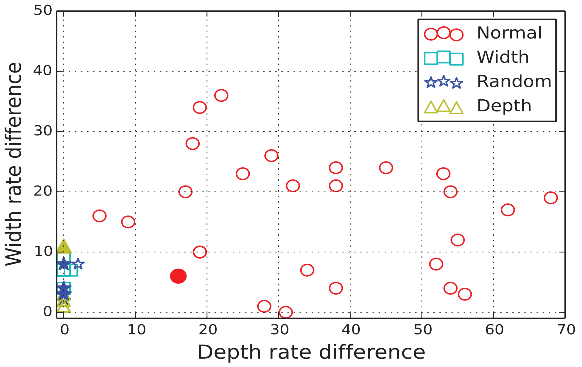

Fig. 3: Differences on Feature 4 and 5

of the collected data for training and the other half for testing. All experiments have gone through official IRB review.

We add crawlers' data in the training set by implementing 6 crawlers to crawl the system. These crawlers are provided by Frontera [1], which relies on Scrapy [2]. 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. The crawlers with delay means that there is a random time interval between any consecutive requests.

B. Data Analysis

Our data analysis verifies that crawlers and normal users behave differently as we described in IV-C3. To illustrate the effectiveness of our SVM models, we show normal users and crawlers' feature 1 and 2 in Figure 2 and feature 4 and 5 in Figure 3. In both figures, each data point represents a long session. The circles 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. 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 behavior difference between the longest short session and the corresponding long session is small for crawlers, meanwhile the difference is large for normal users.

C. Performance Evaluation

We evaluate three aspects of PathMarker, namely, the capabilities 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.

TABLE I: 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%

1) Detection Capabilities: Table I shows that our SVM model is able to correctly classify normal users and crawlers with a high 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. 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 behavior for at most two long sessions. After identifying a crawler, we classify the path-patterns of the crawler, as shown in Table I. We see that most bots' paths can be fit into the three patterns we define for crawlers with over 90% accuracy. For the 3.57% users' misjudged long sessions, we found they all contain some logs that happen to have depth-relationship or width-relationship with other irrelavent short sessions. These logs confuse our feature collection module. However, all these false positive alerts will not affect the system much since they could be removed by CAPTCHA module.

2) Suppressing 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. We assume the links satisfy a Gaussian probability distribution with mean 0 and standard deviation 3,333, so 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. Figure 4 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.

Figure 5 illustrates the entire work of a 10-worker distributed crawler need to download a certain percentage of the entire website. Percentage 100 means the crawler only visits new page and percentage larger than 100 means this crawler

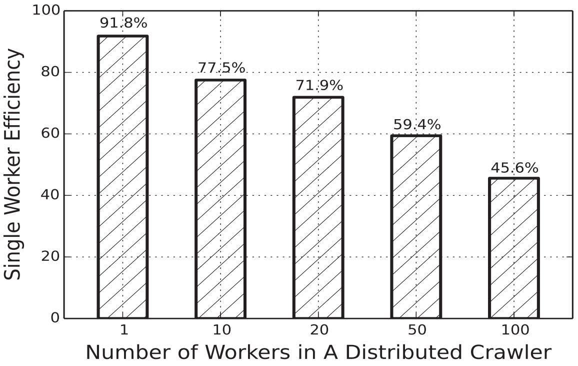

Fig. 4: Suppressing Distributed Crawlers

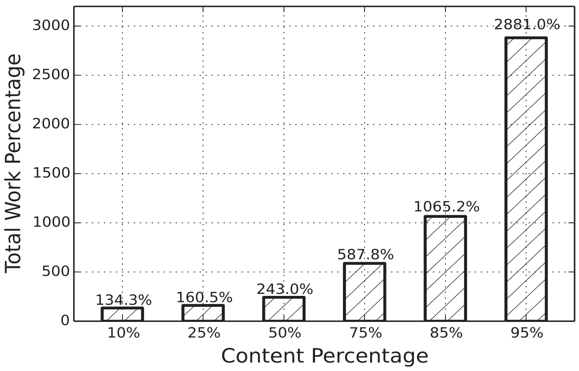

Fig. 5: Overhead for Distributed Crawlers

visits repeated pages. We see that crawling half of all content requires 143% more queries than crawling an unencrypted website. 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.

3) System Overhead: 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 set-up 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. 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. We believe this increase is acceptable since for a normal user who wants to view confident documents.

VIII. Conclusions

In this paper, we develop an anti-crawler system named PathMarker to enable 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. It 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.

IX. Acknowledgement

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

References

[1] Frontera 0.3. http://frontera.readthedocs.org/en/latest/index.html.

[2] Scrapy 1.0. http://scrapy.org/.

[3] Snowden used common web crawler tool to collect NSA files. https://www.rt.com/usa/snowden-crawler-nsa-files-227/.

[4] A. Aghamohammadi and A. Eydgahi. A novel defense mechanism against web crawlers intrusion. In *Electronics, Computer and Computation (ICECCO), 2013 International Conference on*, pages 269–272. IEEE, 2013.

[5] F. Ahmadi-Abkenari and A. Selamat. An architecture for a focused trend parallel web crawler with the application of clickstream analysis. *Information Sciences*, 184(1):266–281, 2012.

[6] C.-C. Chang and C.-J. 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.

[7] M. D. Dikaiakos, A. Stassopoulou, and L. Papageorgiou. An investigation of web crawler behavior: characterization and metrics. *Computer Communications*, 28(8):880–897, 2005.

[8] D. Doran, K. Morillo, and S. 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.

[9] EllisLab. Codeigniter. https://codeigniter.com/.

[10] I. Inc. Bot traffic report 2016. https://www.incapsula.com/blog/bot-traffic-report-2016.html.

[11] G. Jacob, E. Kirda, C. Kruegel, and G. Vigna. Pubcrawl: Protecting users and businesses from crawlers. In *USENIX Security Symposium*, pages 507–522, 2012.

[12] J. Jin, J. Offutt, N. Zheng, F. Mao, A. Koehl, and H. 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.

[13] J. Jin, J. Offutt, N. Zheng, F. Mao, A. Koehl, and H. 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.

[14] M. A. Kausar, V. Dhaka, and S. K. Singh. Web crawler: a review. *International Journal of Computer Applications*, 63(2), 2013.

[15] C. Olston and M. Najork. Web crawling. *Foundations and Trends in Information Retrieval*, 4(3):175–246, 2010.

[16] B. I. P. Rubinstein, B. Nelson, L. Huang, and A. D. Joseph. Antidote: Understanding and defending against poisoning of anomaly detectors. In *In Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, 2009.

[17] D. Stevanovic, N. Vlajic, and A. An. Detection of malicious and non-malicious website visitors using unsupervised neural network learning. *Applied Soft Computing*, 13(1):698–708, 2013.

[18] G. Suchacka and M. Sobkow. Detection of internet robots using a bayesian approach. In *Cybernetics (CYBCONF), 2015 IEEE 2nd International Conference on*, pages 365–370. IEEE, 2015.

[19] P.-N. Tan and V. Kumar. Discovery of web robot sessions based on their navigational patterns. In *Intelligent Technologies for Information Analysis*, pages 193–222. Springer, 2004.

[20] G. Xie, H. Hang, and M. 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.

[21] F. Yu, Y. Xie, and Q. 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.