Anti-Methods for Distributed Web-Crawler: Long-tail Threshold Model

Anonymous Author(s)

ABSTRACT

In this paper, we propose a countermeasure against distributed crawlers. We first introduce known crawler detection methods in general and how distributed crawlers bypass these detection methods. Then, we propose a new method that can detect distributed crawlers by focusing on the property that web traffics follow the power distribution. This means when we sort the items by the number of requests, most of the requests are concentrated on the most frequently requested items.[1] And there will be a long-tail area that legitimate users do not generally request but crawlers will. Because crawler algorithms are generally intend collect every target items. So crawlers request iteratively and recursively request items by parsing web service architecture. By following these two assumptions, we can assume that if some IPs are frequently requesting the items that are located in longtail area, those IPs could be classified as crawler nodes. We tested this theory by simulating with real world web traffic data that NASA released and found the effective and low false positive results.

KEYWORDS

Web Crawler, Traffic Analysis, Power Law, Information Theory

1 INTRODUCTION

Web-crawling is used in various fields for collecting data.[8] Some web-crawlers collect data even though the target site is prohibiting crawlers by robot.txt. And even though web services try to detect and prevent crawlers by anti-crawler methods, these malicious web-crawlers bypass detection by modifying their header values or distributing IPs to masquerade as if they are legitimated users. It is a matter of availability and data property issue because even though any service permits viewing of each data, it is prohibited to duplicate an entire dataset. And malicious web-crawlers can cause significant traffic and intellectual property infringement by collecting the entire data from web services. This can have a serious impact on the availability of the target service. In this paper, we introduce the conventional anti-crawling methods and its countermeasures, and show that the conventional anti-crawling methods cannot defend against the distributed crawler. Then we propose the new anti-crawling technique that we call 'node-reducing' which gradually adds the distributed crawler node IPs to the block-list.

2 RELATED WORKS

In this section, we will describe about conventional anti-crawling methods and their counter crawling measures.

(1) HTTP Header Check
A basic crawler will send requests without modifying its header information. And web servers can distinguishes a legitimate user from a crawler by checking the request header, especially User-Agent value has been set properly. This header checking method is a basic anti-crawling method. But if a crawler attempts to masquerade itself as a legitimate user, it will replay the header information from the web browser or form the http header information similar to a browser. This makes it difficult for a web server to determine whether a client is a crawler or a legitimate user by simply checking the request header.

(2) Access Pattern based Anti-Crawling
Access pattern based anti-crawling is a method of classifying legitimate users and crawlers based on the pattern requested by the client. If a client requests only a specific resource continuously without a call to a resource that should normally be requested, the corresponding could be regarded as a crawler. An attacker performing an aggressive crawling predefines the core data that the web service wants to collect, and implements a crawler that requests specific data without requesting unnecessary resources. In this case, the web server can recognize that the client is not a legitimate user. In the case of a web service using advanced approach to access pattern recognition, the service is viewed as a set of consecutive requests from the viewpoint of the user UX, and requests and responses belonging to the same set are chained by including a specific hash value in the cookie. Although this approach can recognize a crawler based on access pattern, some crawlers even masquerade their access pattern by analyzing network logs. [8]

(3) Access Frequency based Anti-Crawling
Access frequency based anti-crawling is a method that determines a client is whether a crawler or a legitimate user by access frequency rate. A web server can set a threshold limit of access count in an unit time. If a client with a specific IP requests exceeds this limit in pre-defined time, the web server determines that this IP as a crowler node. This method has two well known problems. First, it has vulnerability against distributed crawler.[8] If an attackers use distributed crawler service such as Crawlera, access rate for a crawler node IP will be reduced enough to bypass threshold limit. Second, it could raise false positive error if many users share a public IP.

3 BLOCKING DISTRIBUTED CRAWLER

As described in previous section, distributed crawler can bypass every conventional anti-crawling methods. In this section, we propose a new technique to detect and block distributed crawlers that could not be defended by existing anti-crawling techniques.

ACSAC'18, December 2018, San Juan, Puerto Rico
2018.

3.1 Required Number of Crawler Nodes

In order for the Distributed crawler to collect the entire data of a website, the following conditions must be met.

$$Cn \geq Um/(Td * 30)$$

The above formula can be described as follows.

- Um: Number of items u›ated in month
- Td: Maximum number of request per IP
- Cn: Number of crawler node(IP)

For an example, if there is a web service updates Um(30,000) items in a month and the service has a restriction rule that if an IP requests more then Td(100) times will be banned, than an attacker who tries to collect every items from the service use at least Cn(10) crawler nodes to avoid restriction.
Therefore, as Um increases or Td decreases, Cn increases. And Cn numerically indicates the level at which the website is difficult to crawl.

3.2 Generating Long-tail Td zone

The number of items that are updated in a month can not be arbitrarily increased. Simple way to prevent distributed crawler is to decrease Td. But this will also increase false positive significantly. In this paper, we solve this problem by reversing the general characteristics of web traffic and the fact that the crawler tries to replicate the entire data.
If you sort the items by access rate, you can see the exponentially decreasing form as shown in the following figure. This means most of the web traffic is concentrated on most frequently requested items. And there is a long-tail region that has low access rate. We calculated max request count of this longtail region and set this value as Td3.

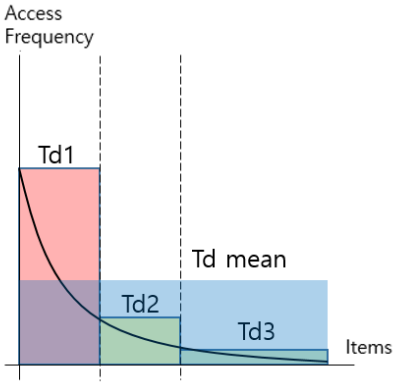

Figure 1: Access Frequency per number of connections

In information theory, unlikely event is more informative then likely event. And the event that a request to an item in long-tail region is much unlikely then the upper items. This means the web service can find more information from a request to the long-tail region.
Hence, when a client keep request the items in long-tail region, the web service increase the count until it reaches Td3, instead of reaching Td mean. This means a web service can set much more sensitive threshold without increasing false positive error rate.

3.3 Node Reducing with Long-tail Region

In order for an attacker to collect the entire data from the service, he or she must also access the items in the long-tail (Td3) interval. However, the attacker does not know exactly which item the item he is accessing belongs to. Using this information asymmetry, service providers can easily identify IPs that are accessed more frequently than long-tailed segments. These identified crawler IPs will included in block-list and the number of IPs in blcok-list will be called Cm. If we start to increase the Cm value through the long-tail interval, the attacker will crawl with a smaller number of IPs, and Cm will increase in the Td3 interval.

- Cm: Number of crawler node(IP) blocked by service
- long_t: Ratio of items included in long-tail region

So attackers must satisfy the following inequality. Cn - Cm is the number of non-blocked crawler nodes and this should be greater then the right term.

$$Cn - Cm \geq (Um * long\_t)/(Td3 * 30)$$

On the service provider side, Cm should be greater then right term to block distributed crawlers.

$$Cm > Cn - (Um * long\_t)/(Td3 * 30)$$

If a particular IP accesses an item in the long-tail region with more than the Td3 value determined by the above formula, it can be included in the block-list.

3.4 Dummy Items

The service provider may include a dummy item to detect the crawler in addition to the actual service target item. Dummy items are inaccessible for the legitmiate user because there are no user interfaces for dummy items or hidden. There are few ways to generate dummy items, it may exists as an HTML tag but it is not displayed on the screen by the attribute setting or it may contains a garbage information that normal users shell not be interested. But a crawler that performs sequential access to the service will may access the dummy items. By this characteristic, dummy items can work as extension of the long-tail region.
In this paper, we will not include dummy items in experiments due to fair comparsion with real traffic logs which will never access dummy items.

4 EXPERIMENT

Our experiment was designed to verify the classification performance of crawler detection module between crawlers and legitimate web traffics. We compared LTM(Long-tail Threshold Model) with

normal access frequency based anti-crawling on maximum number of crawler nodes and false positive.
We used the real web traffic log that NASA released in 1995. And then we performed pre-processing, modeling and simulation.

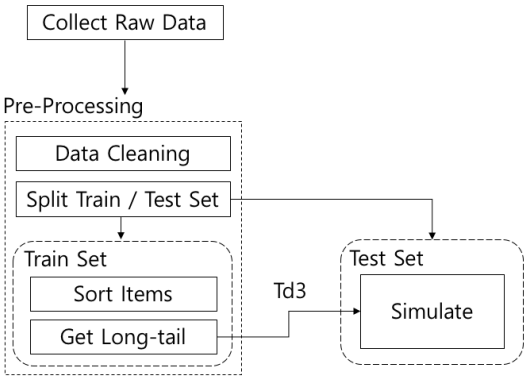

Figure 2: Experiment Design

4.1 Web Traffic Data

(1) Source
NASA released a total of 1,891,715 access logs for the month of July 1995. We parsed this log into csv format and composed of 4 columns including IP, date, access target and access result. The total number of connected IPs are 81,978 and the number of items are 21,649.

(2) Data Pre-Processing and Traffic Distribution
In order to prevent duplication of data used for in modeling and experimental data in the time series data, we split the log by time. July 1 to 24 was used for LTM modeling and 25 to 30 was used for testing. When a user accesses a html file, they also get accesses to gif files that are linked. This can force a multi increase of access count. Hence we removed some gif files from the long-tail. For the last we excluded the request logs that the access results are not success from the experiment. After doing the pre-processing we have got the test set as below table.

Table 1: Pre-Processed Web Traffic Data

Number of Total Items	7,649
Number of Long-tail	5,355
Mean of Total Items	184.76
Mean of Long-tail	1.88

As a result, we made a pre-processed traffic data set consists of 7,649 items from 21,649 raw data. Which has 5,355 items as a long-tail region. The mean of the total request count was 184.76 while the mean of long-tail region was 1.88. It means simply we can set about 6880% more sensitive threshold to the crawler detection algorithm.

Table 2: Experiment Data

	Ratio	mean	max	count
Td1	> 0.5%	21,250	76,040	38
Td2	0.5% - 30%	264	7,043	2,256
Td3	30% >	1.88	9	5,355

The key part was the real web traffic will show the power distribution after sorting with access frequency or not. And we could verify the NASA traffic data also has power distribution as shown in Figures 2, 3 and 4.

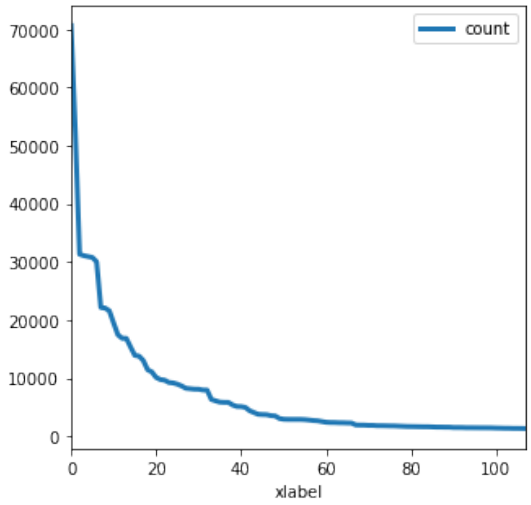

Figure 3: Access Count in Td1

Figure 2 shows the group of the most frequently requtestd items.

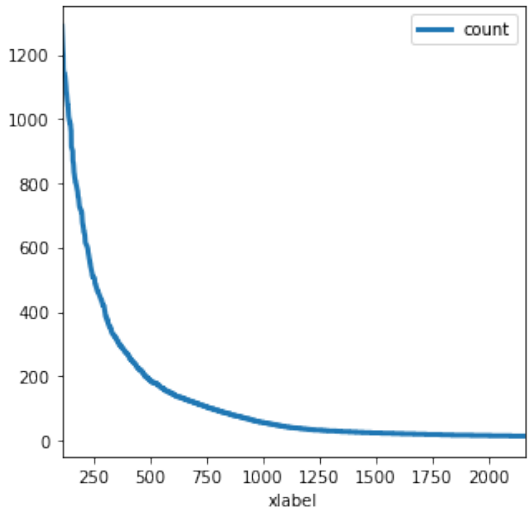

Figure 4: Access Count in Td2

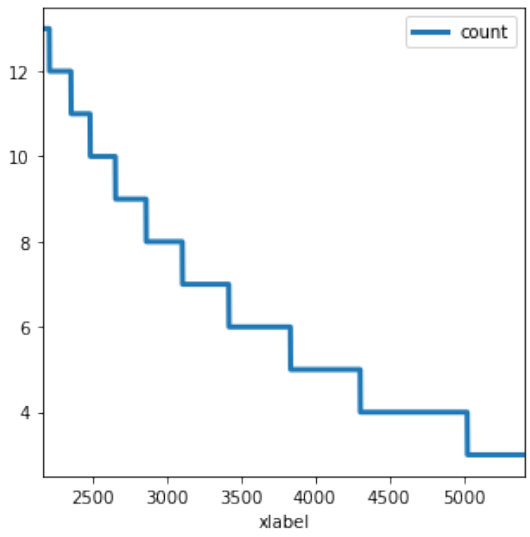

Figure 5: Access Count in Long-tail

Figure 4 shows the long-tail region of sorted result. We set Td3 threshold value as 20 which is about 2 times larger than the maximum access rate of long-tail region. Setting a Td3 value in LTM has a heuristic part because web services has different purposes and circumstances.

4.2 Simulation

In this paper, we implemented simulation for two purpose. One is to check whether it is possible to detect and disable the crawler IP group with LTM, and the other one is to check false positive when input the actual web traffic to LTM.

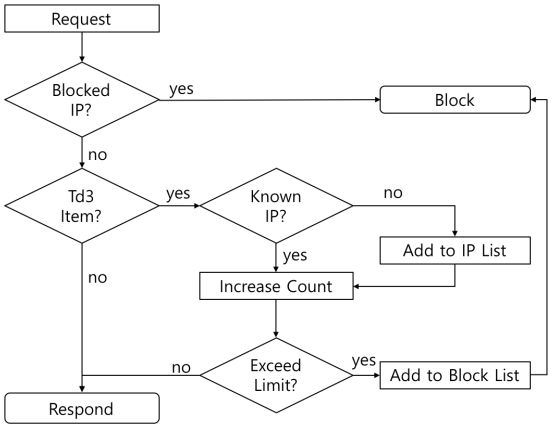

Figure 6: Crawler Detection Flow

(1) Distributed Crawler Detecting Simulation
We used python for implementing the LTM simulator. The required parameters are the 1) size of the distributed IP set used by the crawler, 2) the long-tail list, 3) the entire item list, and 4) threshold values used for detection.
The LTM simulator watched the Crawler IP Set to access each item by traversing the entire item list, and increased the access count for the IP at each access to the long-tail entry. When the crawler accesses a long-tail entry, LTM increases the access count of IP. Exceptionally, if a same IP accesses the same item again, LTM does not increase the access count considering that it is not related to the purpose of crawling. When an access count of IP exceeds the threshold value, LTM adds the corresponding IP to the block list. Figure 5 below shows an example of running a crawler using 100 distributed IPs for 7,649 items. The number of long-tail items is 5,355 and the threshold is set to 20. The following figure is a graph of the process of reducing 222 crawler sets on the simulator.

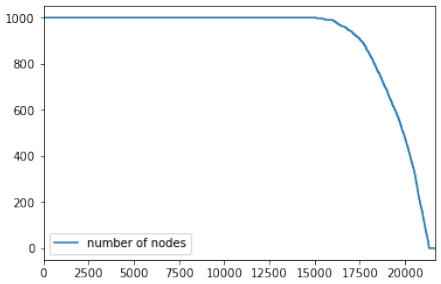

Figure 7: Number of IPs reduced by detection

We can verify that the crawler IP set is gradually reduced until get totally blocked. When the first crawler node IP access count exceeds the Td3 and blocked, node reducing count increases exponentially. This is because when a crawler node blocked, other crawler nodes get more burden and has to access more items.

(2) Node Reducing Result
Experiments were performed with threshold set to 20, and the crawler set consisting of 222 nodes was completely detectable and false positives were 1.33 cases per day, which was 0.0312% of the daily IP number. In below table, we compared the result of LTM with normal FBA(Frequency Based Anti-crawling) method.

Table 3: Experiment Data

	Threshold	Max Node	False Positive
LTM	10	426	0.1239%
LTM	20	222	0.0275%
LTM	35	128	0.0046%
FBA	10	573	8.8064%
FBA	20	299	2.8819%
FBA	35	173	0.8903%
FBA	100	60	0.0367%

We can detect more or less number of crawler nodes depending on the number of items the site has or the length of the

long tail. Since this simulation is based on old NASA traffic data(1995), total number of items were quite small compared to modern web services. If a service has 10 times more items than our test data and has equal distribution that service could detect more than 2000 crawler nodes.
In the experiment, we compared the detection capability between the LTM and the FBA when the same threshold was set, and compared false positives under the same conditions.

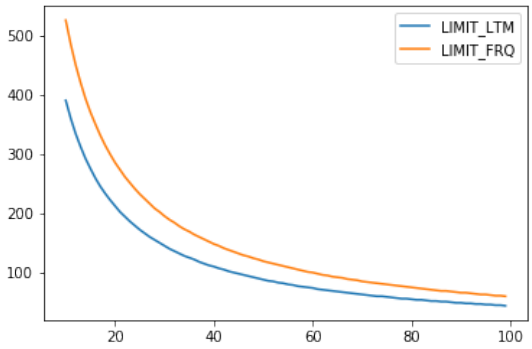

Figure 8: Number of Detectable Crawler Nodes

Since LTM uses long-tail region instead of entire items, FBA detects more crawler nodes with equivalent threshold as figure 8. But as shown in the following figure, it is meaningless to simply compare the detection results without considering false positives.

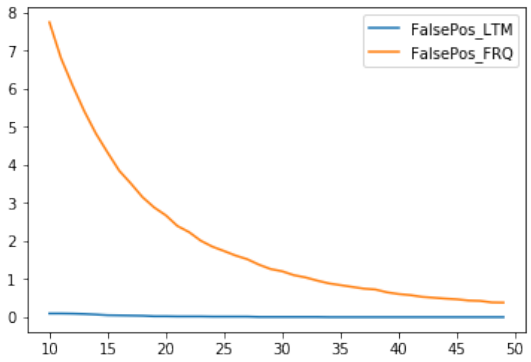

Figure 9: False Positive: 1 to 50

As we can verify from figure 8 and 9, as threshold in x-axis increases false positive and number of detectable crawler nodes are both decrease. In the case of LTM, we can verify that the false positive value is less than 0.15 % from the low threshold region of less than 10.

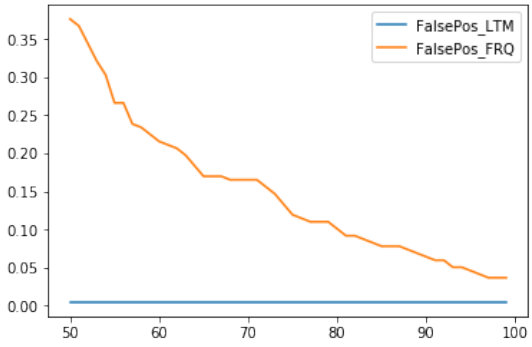

Figure 10: False Positive: 50 to 100

However, in case of FBA, the threshold should be set to 76 or more to have similar false positive error performance by trading off the detection performance.

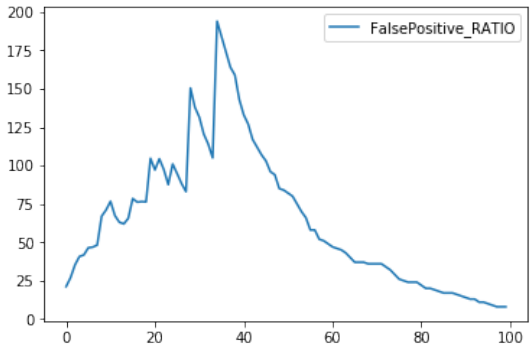

Figure 11: Number of Detectable Crawler Nodes

In this experiment, LTM reached the minimum false positive error rate of 0.0046% when FBA reached only 0.0367% which is 798% larger than the best score of LTM. This means within the similar level of false positive, LTM performs 400~500% better detection score than classic FBA method. And even though FBA trade off the detection score, FBA generates 798% times more false positive error. When we set the threshold to 35 which is the value that LTM reached the minimum false positive, FBA generated 19400% times more false positive than LTM.

5 CONCLUSION

In this paper, we introduced LTM(Long-tail Threshold Model) a node reducing method that identifies the IP set of distributed crawlers and gradually reduces IP by using the long-tail region.
By simulating with the real web traffic data, LTM effectively identified distributed crawler and showed a very low level of false positive error. Web crawling against the term of web service or impairs quality is a serious security threat. And considering that there are some crawler developer using distributed crawler proxy service for illegal purposes, LTM could improve web service data security.

6 FUTURE WORKS

Web traffic generally tends to generate traffic bursts at certain times. [1] Although the experiment of this paper is based on actual traffic log, since the time point of the data used in the experiment is one month, it does not include cases where a new item is added or an issue occurs and a traffic burst occurs. In order to apply the results of this paper more securely to actual services, it is necessary to study whether the item movement level and threshold value of long-tail area can be maintained based on actual traffic data for traffic burst occurrence cases.

REFERENCES

[1] Andoena Balla, Athena Stassopoulou, and Marios D Dikaiakos. 2011. Real-time web crawler detection. In *Telecommunications (ICT), 2011 18th International Conference on*. IEEE, 428–432.

[2] Mark E Crovella and Azer Bestavros. 1995. *Explaining world wide web traffic self-similarity*. Technical Report. Boston University Computer Science Department.

[3] Marios D Dikaiakos, Athena Stassopoulou, and Loizos Papageorgiou. 2005. An investigation of web crawler behavior: characterization and metrics. *Computer Communications* 28, 8 (2005), 880–897.

[4] M Sunil Kumar and P Neelima. 2011. Design and implementation of scalable, fully distributed web crawler for a web search engine. *International Journal of Computer Applications (0975-8887)* 15, 7 (2011), 8–13.

[5] scrapinghub. 2018. Crawlera - The World's Smartest Proxy Network.

[6] Vladislav Shkapenyuk and Torsten Suel. 2002. Design and implementation of a high-performance distributed web crawler. In *Data Engineering, 2002. Proceedings. 18th International Conference on*. IEEE, 357–368.

[7] Bernard W Silverman. 2018. *Density estimation for statistics and data analysis*. Routledge.

[8] MV Simkin and VP Roychowdhury. 2008. A theory of web traffic. *EPL (Europhysics Letters)* 82, 2 (2008), 28006.

[9] Athena Stassopoulou and Marios D Dikaiakos. 2006. Crawler detection: A bayesian approach. In *Internet Surveillance and Protection, 2006. ICISP'06. International Conference on*. IEEE, 16–16.

[10] Dusan Stevanovic, Aijun An, and Natalija Vlajic. 2012. Feature evaluation for web crawler detection with data mining techniques. *Expert Systems with Applications* 39, 10 (2012), 8707–8717.

[11] Yuan Wan and Hengqing Tong. 2008. URL assignment algorithm of crawler in distributed system based on hash. In *Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference on*. IEEE, 1632–1635.

[12] Demetrios Zeinalipour-Yazti and Marios Dikaiakos. 2002. Design and implementation of a distributed crawler and filtering processor. In *International Workshop on Next Generation Information Technologies and Systems*. Springer, 58–74.

[13] Hongyan Zhao. 2015. Research on Detection Algorithm of WEB Crawler. *International Journal of Security and Its Applications* 9, 10 (2015), 137–146.