A Novel Malicious Web Crawler Detector: Performance and Evaluation

DeXiang Zhang[1], DiFan Zhang[2] and Xun Liu[3]

[1] Information and Network Center, Qingdao University
Qingdao, Shandong 266071, China

[2] Information and Network Center, Qingdao University
Qingdao, Shandong 266071, China

[3] Library, Qingdao University
Qingdao, Shandong 266071, China

Abstract

Internet demands a robust and resilient protected communication and computing environment to enable information flows flawlessly with no down time. However, the Internet is exposed to the general public which will lead to loss of sensitive information as well as copyright protected content. To address this issue, in this paper, we proposed two schemes to fight against unwanted automatic web crawlers, TSSNBS (Too Simple Sometimes Naive Blocking Schema) and ABS (Adaptive Blocking Schema). We validated the effectiveness of the two schemes by implementing an advanced integrated crawler detection system and applied on a high trafficked site in the real world, exposed with real attackers.

Keywords: *Crawler, Internet, Network, Algorithm.*

1. Introduction

Internet demands a robust and resilient protected communication and computing environment to enable information flows flawlessly with no down time. Nevertheless, the openness nature of Internet causes security risk of information leakage because the information are accessible to anyone without proper access control. In recent years, much research has been devoted to the construction of web technologies; contrarily, few have investigated the construction of architecture. Fortunately, several access control schemes were introduced including WebDAV [1] and The PLAIN Simple Authentication and Security Layer (SASL) Mechanism [2].

Information flows in the Internet. To better organize the world's information and make it universally available and accessible, crawlers, or known as web spiders, are invented to traverse against the Internet to fetch information [3].

To keep information confidential and prevent automatic crawling programs not behaving normally, we proposed a novel method of detecting unwanted automatic crawlers. However, considering user anonymity and local law requirements, raw logs was never processed without stripping out user information.

To future help minimizing the effectiveness of this technique, we proposed a dynamic blocker to block the malicious request in real time.

2. System Design

2.1 Overview

Crawlers behave significantly different from normal users since they are automated programs with pre-defined routines, thus allowing researchers to use fingerprint based techniques to classify them. Per analysis of the behaviors of several commonly seen crawlers and robots, we concluded several commonly seen patterns. By detecting those patterns, we can figure out malicious traffic effectively.

By utilizing known HTTP and TCP features, active and passive network sensors can be put in the system to monitor those traffic and with HTTP features as well as TCP features, those traffic can be got rid of from the entire system with little computational resource consumption.

2.2 Crawler Pattern Analysis

Most crawlers are not script awareness and are simply traversing against all links found in a page with a fixed

interval. For those crawlers, we found the following patterns and are surprisingly high performing in detection.

(1) Continuous Requests: Many crawlers are programmed to parse an entry page, extract links in the entry page and visit each link immediately or after a fixed or random interval. For robots, in order to fetch the whole site as fast as possible, the interval is likely to be short. Regardless of the interval, in the access log, we can observe consequent and continuous requests. By defining an adequate threshold of visiting the site, we can figure out possible crawlers [4,5].

(2) Not Accepting Cookies: Since HTTP is stateless, to keep state of the user, cookies are used. However, due to the nature of crawlers which is stateless, it does not keep cookies sent from the server. Thus, requests from the same or similar (in the same C class) IP address which never send cookies information can be very suspicious.

(3) Bogus User Agents: Users cannot access the Internet directly. Instead, users use User Agents. Most commonly seen user agent is web browser. All user agents use an user agent string to identify itself. All browsers will send out User Agent information. However, many crawlers are omitting user agents; others are simply identifying themselves as crawlers or very old browsers including Internet Explorer 3.0 running on Windows 95 or Netscape4.78 on Solaris. Since those old browsers are not capable for the current Internet, we can safely define a blacklist of user agent or even use machine learning algorithms to automatically generate a white-list.

(4) Not Loading/Executing Scripts: Opposed to web browsers which has integrated scripting engine (mostly ECMAScript interpretation engine, whether fully functional and complying with standards or not), spiders are not equipped with scripting engines in most cases for simpler implementation and faster execution. Thus, by putting pitfalls and triggers in the source code, we may be able to implement traps for web spiders and automated bots. However, considering the instability nature of the Internet, thresholds should be set and timeouts should be available [6].

(5) High Fetch Rates: Another common approach in implementing web spiders and crawlers is to fetch pages as fast as possible. However, normal users tend to load several pages at a time, read the pages and load another batch of pages after a relatively long period.

2.3 Blockage of Requests

After detecting malicious traffic, traditionally, we implement firewall rules or system configuration rules to block the traffic. However, this requires human involvement and thus is offline. Also, it requires much human invocation. To address such issues, according to RFC2616 [7], we first use standard HTTP error responses, and then use connection based blockages.

(1) HTTP Error Message based Blockage: To prevent requests being made successfully, we use HTTP error messages. By returning a 40x error, the client won't be able to receive any useful message and the application server won't even receive such request, saving much system resources and traffic. We used a customized HTTP error message, code 444, as a respond. Such respond totally ignores the request and returns nothing but the HTTP header.

(2) Connection based Blockage: For most malicious requests, HTTP Error Message based blockage should be enough. However, more some DDoS aimed malicious requests, even returning HTTP Error Messages help to take down the server. Hence, we have connection based blockage. In such blockage mode, we maintain a table of blocked internet protocol addresses and scan this table every time accepting a new connection.

2.4 Strategies against Malicious Crawler

We proposed two different strategies in blocking malicious crawlers, TSSNBS (Too Simple Sometimes Naive Blocking Schema) and ABS (Adaptive Blocking Schema) to block malicious crawlers. We use TSSNBS as the primitive decision making algorithm, and for unclear crawlers, we use ABS which is powerful yet resources consuming[8].

(1) TSSNBS (Too Simple Sometimes Naive Blocking Schema): The TSSNBS algorithm aims to provide a quick and relatively inaccurate method to detect malicious web spiders. Thus, we choose to be stricter that ambiguous requests will be marked as malicious. This will increase false positive ratio, however, will block almost all real bots.

For user agents, we have an extended list of known bad keywords and another list of known-to-be-good client list which was represented in Regular Expressions to provide better compatibility and scalability. To provide better performance, we used JIT technologies and cached byte-codes generated from PCRE library.

(2) ABS (Adaptive Blocking Schema): The adaptive blocking schema is a combination of machine learning and advanced metrics mentioned in features of malicious crawlers above. Specifically, it is stateful. For each suspicious connection passed from the TSSNBS, it will create a session and keep record of it for an period of time. The session will be served a specially designed trap in JavaScript. Normal user agents will try to execute this script and respond accordingly.

For clients not accepting cookies and not responding the traps in JavaScript correctly, we accumulate a counter which is cleared after a certain period of time. If the counter reached a carefully designed threshold before it expires, we identify them as crawlers.

(3) Blocking Strategies: Initially, when a connection is identified as malicious, we stop it immediately by sending a HTTP error message, and keep a volatile counter with a fixed timeout value. When there's more connecting coming from the same requester, we increase this counter by one. When it reaches a certain threshold, we mark such requester as abuser and pass it to the kernel space driver. This driver will then drop the packet from such requester when it reaches the Ethernet buffer. When it's dropped in the buffer, the traffic will not be noticed in any user land applications.

2.5 Implementation

After several weeks of onerous coding, we finally have a working implementation of our system. The implementation was delivered as three decoupled parts – a web server module to dynamically load block list from shared memory which runs in nginx, a web service module which analyzes and serves designated requests to differentiate bots from users, and a backend server to generate reports and make final decisions.

The implementation has two modes – TSSNBS mode and dual mode. Under TSSNBS mode, only basic rules are applied. Under dual mode, TSSNBS are applied first and for ambiguous traffic, ABS is applied.

The HTTP Error Message based blockage module was implemented as a web server module, and connection based blockage was implemented with UNIX shared memory and net filter. Shared memory was used to provide a simple interface to communicate from user land to kernel space in a reasonably fast fashion.

3. Evaluation

Evaluating complex systems is difficult. Only with precise measurements might we convince the reader that performance matters. We use two metrics to evaluate the malicious crawlers, detection ratio P_d and false positive ratio P_f.

$$P_d = \frac{Count(crawlers\ \det ected)}{Count(crawlers)} \times 100\% \quad (1)$$

$$P_f = \frac{Count(false\ \ positive\ crawlers)}{Count(crawlers\ \det ected)} \times 100\% \quad (2)$$

3.1 Testing Environment

To evaluate the algorithm, we run the evaluation program on a testing platform. The testing platform is a famous technology media focused on mobile applications and in-depth analysis of relevant news. The site has two servers with a load balancing configuration and is running nginx [9], an open source light weight web server as front-end server.

The testing environment runs Debian Linux 6.0.6 with up-to-date patches, Nginx 1.2.4 and Redis 2.0. Redis was chosen as memory-cached temporary storage engine. The version of kernel is *Linux 2.6.32-5-amd64 #1 SMP Sun May 6 04:00:17 UTC 2012 x86_64 GNU/Linux*. The test was conducted on December, 2012.

3.2 Experiment Steps

(1) Deploy Crawler Blocking Plugin: To deploy the implementation, we chose clang compiler *clang version 4.0 (tags/clang-421.0.60) (based on LLVM 3.1svn)*, llvm-gcc *gcc version 4.2.1 (LLVM build 2336.11.00)* and gcc *gcc version 4.2.1 20070831 patched*. Standard UNIX tools including *sed,awk, autotools* and *m4* are also used in the deployment.

(2) Deploying the Blockage Plugin: We deploy the blockage plugin in two places – HTTP Error Message based blockage plugin on the web server, and connection based blockage in the kernel as a kernel extension. The kernel extension was compiled with Sun Studio compiler to provide the best performance.

(3) Running the Plugin: To minimize the impact of the experiment to the site availability, we used DNS rotation, and send about 20% traffic to the testing environment. We run the experiment for about 48 hours, collecting gigabytes of HTTP access log.

3.3 Data Metrics and Procession

We use several metrics to measure the performance and effectiveness of our system.

(1) False Positive: We define false positives as known-to-be-good clients identified as malicious traffic. Those traffic will be blocked once it reaches a certain threshold. Lower false positive ratio indicates better accuracy.

(2) Missed Crawlers: Those are malicious traffic labeled as good by our filter. They are considered harmful.

(3) Load Average: Load average is the metric to measure system utilization. We measure the load average against traffic volume and compare load average for different configurations. Lower load average indicates better performance. In our particular system, load average is calculated by counting context switching count:

$$L = \frac{N_{context\ switch}}{T} \quad (3)$$

We take T = 15 seconds to provide better accuracy and prevent bias resulting from accidental incidents such as garbage collection or memory reclaim.

(4) Data Visualization: We collect raw HTTP logs with several customized fields. However, raw data are never meant to be processed. Hence, we first normalize data and associate it with results from the detector. We use GNUPlot to visualize the data into EPS format. We use GNU GhostScript To distill the visualized results.

4. Discussion

Our evaluation strategy represents a valuable research contribution in and of itself and evaluation strives to make these points clear.

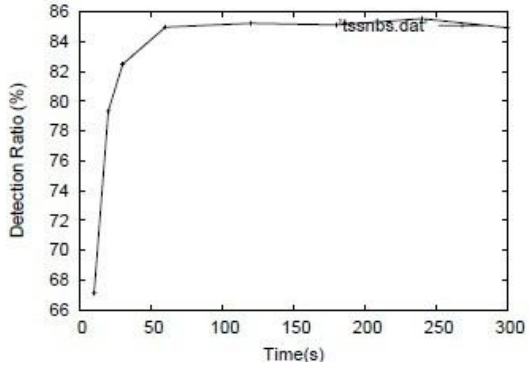

Fig.1 TSSNB Performance – Detection Ratio

4.1 Detection Algorithms

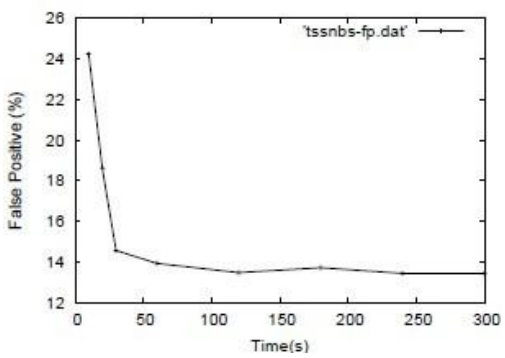

Fig.2 TSSNB Performance – False Positive Ratio

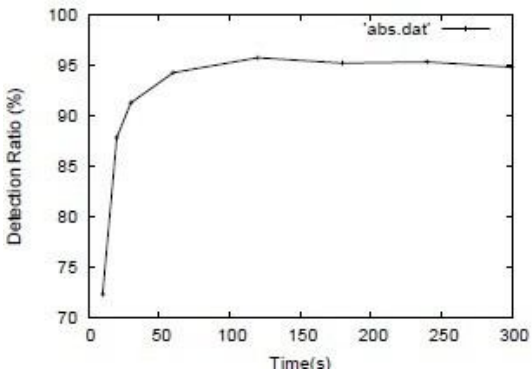

Fig.3 ABS Performance – Detection Ratio

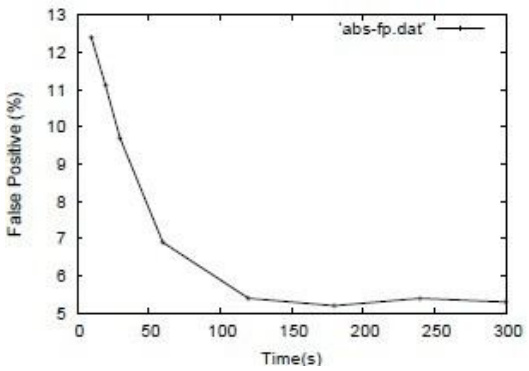

Fig.4 ABS Performance – False Positive Ratio

From Fig.1, we can see clearly that after about 60 seconds, the detection ratio is stabled at about 86%. While a detection ratio of 86% will be sufficient for most applications, for mission critical applications, it is not acceptable. Also, we may lost up to 15% legitimate traffic according to Fig.2.

However, from Fig.3 and Fig.4, we see a great performance improvement which is almost 97% detection ratio and 3% false positive ratio, which means after more optimization and white-listing functionality; it could be applied in production systems.

4.2 Blockage Algorithms

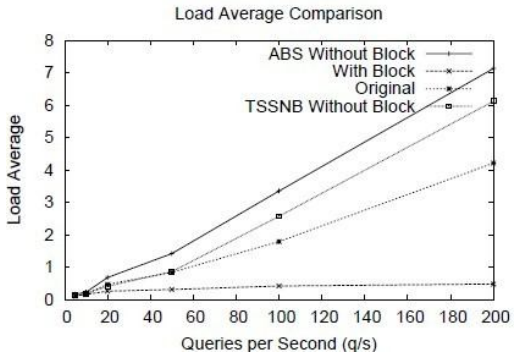

Fig.5 Load Average Comparison

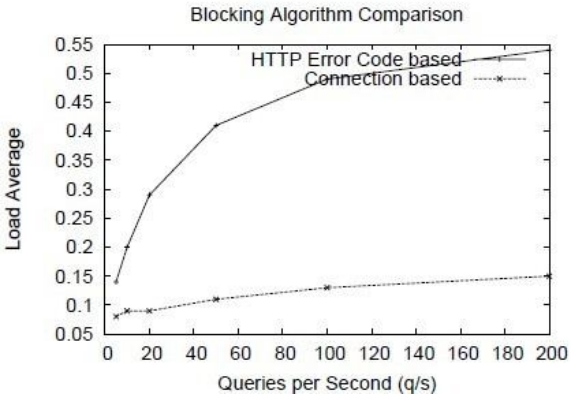

Fig.6 Blocking Algorithm Load Average Comparison

Fig.5 was plotted with data specially collected from known malicious bots. The load average is almost linear to QPS value. Since TSSNB consumes less computational resources, it provides better performance compared with ABS algorithm. However, both of them consumes considerable CPU time and will affect performance of the entire system significantly. However, with the blockage module deployed, the overhead is about 0:2 in load average, which does not affect the performance at all. Also, on peak hours, it will reduce the load average efficiently.

From Fig.6, we can draw two conclusions: 1) both algorithm has an excellent job in controlling load average, 2) HTTP Error Code based blockage still consumes much more computational resources.

Since the HTTP Error Code based blockage requires more memory to run by nature, and since the web server must be invoked to process, it's not surprising that HTTP based algorithm has bad performance. However, as the kernel module requires proprietary software to compile and has to be upgraded on every kernel upgrade, also considering the preliminary implementation lacks security audit, it's possible that buffer overflow may occur, corrupting the whole kernel space and causing downtime. Hence, on not quite heavily loaded sites, HTTP Error Code based blockage should be sufficient.

5.Conclusion

In this paper, we discussed about several approaches of implementing a situation aware anti bot system. After combining several different techniques and algorithms, we reached a good performance.

Even with the most simple algorithm described in this paper, we are able to get fairly acceptable performance. However, with the highly hand crafted and optimized algorithm and implementation, we are able to reduce server load significantly.

To further improve the detection ratio and reduce the false positive rate, we can use a white-list feature to exclude several known suspicious-thus-legitimate clients. Also, we may use SVM and machine learning algorithms to future detect unknown crawlers. In this way, we can archive better performance and will be production ready. We plan to explore more issues related to these issues in future work.

For large scale systems, we may consider implementing the algorithm in FPGA chipsets and deploy it as hardware to have even better performance and easier deployment. Also, we may consider utilizing watermarking techniques to trace the root of the attacks and cooperate with Internet Service Providers and local law enforcements to take down those servers.

Acknowledgments

The testing system was deployed on a famous Chinese IT media, ifanr.com.

References

[1] G. Clemm, J. Reschke, E. Sedlar, and J. Whitehead, "Web distributed authoring and versioning (webdav) access control protocol," 2004.

[2] K. Zeilenga, "The plain simple authentication and security layer (sasl) mechanism," 2006.

[3] Kyle Zeeuwen,Matei Ripeanu and Konstantin Beznosov,"Improving malicious URL re-evaluation scheduling through an empirical study of malware download centers",Proceeding of the 2011 Joint WICOW/AIRWEB Workshop on Web Quality,pages 42-49.

[4] Pang-Ning Tan and Vipin Kumar,"Discovery of Web Robot Sessions Based on their Navigational Patterns",Data Mining and Knowledge Discovery,vol.6,No.1,january 2012,pages 9-35.

[5] Shinil Kwon,Young-Gab Kim and Sungdeok Cha,"Web robot detection based on pattern-matching technique",Journal of Information Science,vol.38,No.2,April 2012,pages 118-126.

[6] Jingyu Zhou and Yu Ding,"An Analysis of URLs Generated from JavaScript Code",Proceedings of the 2012 IEEE/ACIS 11th International Conference on Computer and Information Science,page 688-693.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Rfc 2616, hypertext transfer protocol – http/1.1," 1999. [Online]. Available: http://www.rfc.net/rfc2616.html

[8] Ashif S.Harji,Peter A.Buhr and Tim Brecht,"Comparing high-performance multi-core web-server architectures",Proceeding of the 5th Annual International Systems and Storage Conference.

[9] I. Sysoev. [Online]. Available: http://www.nginx.org.

[10] Basma Zahra, Anis Sakly and Mohamed Benrejeb. Stability Study of Fuzzy Control Processes Application to a Nonlinear Second Order System, International Journal of Computer Science Issues, Vol. 9, No.2-2,(2012) pp. 97-106.

Dexiang Zhang woks in the Information and Network Center of Qingdao University since 2000.He graduated from Qingdao University with bachelor's degree at 2000 and with a master's degree at 2006.His current research interest includes internet security and information security. He published six papers and involved in the preparation of one book.

Difan Zhang is a researcher on Internet Security at Information and Network Center, Qingdao University. Graduated from Towson University with a Master of Science in Information Technology, his current research interest includes Distributed Intrustion Detection Systems and Mobile Device Security.

Xun Liu is a researcher on Internet Security at library of Qingdao University. He graduated from Qingdao University with a master of MPM.