Automatic NIDS Rule Generating System for Detecting HTTP-like Malware Communication

Chia-Nan Kao[1,3], Yung-Cheng Chang[2], Nen-Fu Huang[1,2],
I-Ju Liao[2], Rong-Tai Liu[3], Hsien-Wei Hung[3] and Che-Wei Lin[1]
[1]Institute of Communication Engineering, National Tsing Hua University, Taiwan, R.O.C.
[2]Department of Computer Science, National Tsing Hua University, Taiwan, R.O.C.
[3]Network Threat Defense Technology Group, Trend Micro Incorporated, Taiwan, R.O.C.
Email: canaan@totoro.cs.nthu.edu.tw

***Abstract*—HTTP is the main protocol of the Internet and many network applications rely on it. Malware also utilizes it as a covert channel through which to evade the firewall (FW) or network intrusion detection system (NIDS). We recognize a malware, which employs HTTP to communicate as the HTTP-like Botnet. Some parts of the network traffic of an HTTP-like Botnet are different from normal HTTP applications. Based on the differences between HTTP-like Botnet traffic and normal HTTP applications, we developed an Automatic NIDS Rule Generating System (ARGS). The ARGS is a proof of concept (POC), which generates the corresponding NIDS rules efficiently and precisely from the input malign traffic (MT). ARGS is an incremental method to generate and optimize the rules. It can generate rules quickly and precisely without first requiring the collection of many malware samples for clustering. For practical purposes, we adopt Snort as our IDS engine in ARGS. In our experiments, the time required by ARGS to process MTs and generate corresponding rules is significantly shorter than existing solution when the rule-optimization is not required. Besides, the generated rule set can detect more 30% malware traffic compared to SourceFire IDS full-set and thus can efficiently stop the spreading of malware in time.**

I. Introduction

In recent years, HTTP traffic has become the main stream of the Internet for web-based video streaming and other web-based applications. Botnet also utilizes HTTP as a communication protocol to evade detecting devices such as a firewall (FW) and network intrusion detection system (NIDS). Botnet is an overlay network consisting of bot hosts. A bot host is a computer that is controlled by the bot master. The communications between the bot hosts and their master are the so-called control and command (C&C) communications.

C&C can be a well-known protocol such as IRC, HTTP or a private protocol. However, C&C with less common UDP/TCP ports makes it easier to be detected and blocked because these ports are unusual. NIDS is a practical scheme to detect Botnet [1] but the creation of a NIDS rule set usually requires a lot of professional manpower and research time. In this paper, we focus on generating an NIDS rule set for detecting HTTP-like Botnet. Some parts of the HTTP-like Botnet traffic are similar to that of a real HTTP application but other parts are not. Based on the observed differences between HTTP-like Botnet traffic and typical HTTP applications, we developed the Automatic Rule Generating System (ARGS).

The main processing flow of ARGS is presented in Fig. 1. We adopted the captured traffic and the known Botnet NIDS rule set as the inputs of ARGS. A malign traffic (MT) was a captured network-activity trace from a malware/bot process. On the contrary, a benign traffic (BT) was a captured network-activity trace from a clean network environment. The BT is consisted of human daily networking behavior and network-device regular behavior such as system updates. We employ Snort [2] as our NIDS engine to match MT and BT with rules. It is a well-known NIDS to the open-source community. We adopted some parts of the Emerging Snort Rule collection [3] as our NIDS Botnet rule base. The Botnet rule base was recognized as the known rule set (KRS) in Fig. 1. If a new MT can be matched with the KRS, it is unnecessary to process the rule generating flow. If it cannot be matched, we select the HTTP-like connections from it and generate the corresponding NIDS rules.

ARGS selects rule candidates from MT with unusual HTTP communication patterns, so it can save the system resources and improve rule-generating performance. In our experiments, the time required by ARGS to process MTs and generate corresponding rules is significantly shorter than that of existing solutions. The processing time depends on the size of MT and BT. In general, MT is much smaller than BT so the size of BT is a key to the performance of ARGS.

The major contributions of this work can be described in terms of three aspects

1) We disclose the unusual HTTP communication types used by the HTTP-like Botnet.
2) We provide an incremental method to generate and optimize the rules. The method can generate rules quickly and precisely without collecting many malware samples first for clustering.
3) The false-positive rate of the rule set generated by ARGS is very low, and the malware detection rate is higher than official SourceFire NIDS rule set, so the method can help to shorten the zero-day attack period.

II. Related studies

In 2005, James Newsome et al. [4] developed a scheme to generate rules for matching polymorphic worms. We adopt

978-1-5090-0188-0/15 $31.00 © 2015 IEEE
DOI 10.1109/IIH-MSP.2015.10

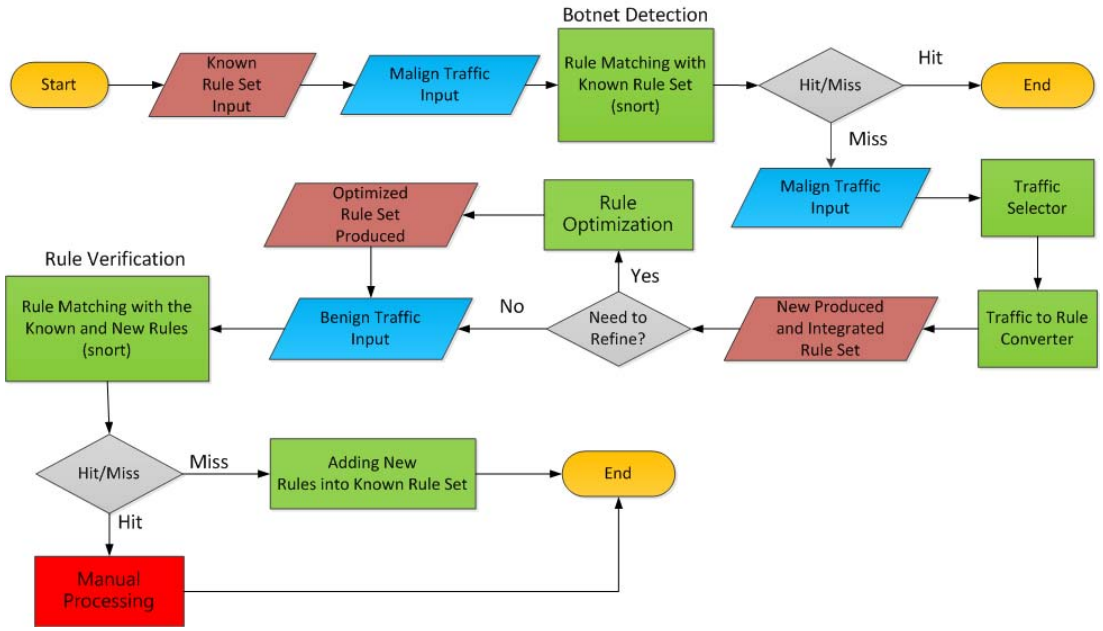

Fig. 1. The Main Automatic Rule Generating Flow

their token-subsequence algorithm to optimize our generated rules.

In 2009, Raghavan Muthuregunathan et al. [5] presented a Snort-based rule-generating scheme by grid computing. In this paper, we try to select the matching target packets by direct traffic analysis without grid computing.

In 2010, Roberto Perdisci et al. [6] (PLF method) presented a network-level HTTP-based malware clustering system and a scheme to generate network signatures automatically. However, the clustering system required many malware samples and a longer execution time in the order of hours.

In 2012, Venkatesh et al. [7] and Zhao et al. [8] employ machine learning techniques to automatically generate HTTP-botnet and P2P-botnet detection models respectively. However, in such systems, it's necessary to train the models with the extracted information from network traffic first, to make similar traffic behavior detectable later.

In 2014, Apostolis Zarras et al. [9] proposed BOTHOUND, an approach to detect HTTP-based malware at network level. This approach focuses on investigating each header of HTTP request and generates their corresponding model as pattern of specific malware or BT. However, this method could not work properly before being initialized with reliable MT and BT first, to establish "ground truth".

III. Design

The main automatic rule-generating flow was described in Fig. 1. The green parts are able to be automated, the red part requires manual processing, the blue parts are the input traffic, and the purple parts are the rule sets involved in the operation of the generating process.

A. HTTP-like Botnet Traffic Types

In this section, we present 5 HTTP-like Botnet traffic types that we observed. We take type 1-4 as criteria to decide which packets among malign traffic are truly behaviors of malware, rather than noise generated by them. For those cases that fall into these types, we would extract some static patterns, which are representative enough to make corresponding traffic can be detected later, such as domain name and HTTP URI to generate rules.

1) Asymmetric: In normal case, the communication between both hosts should use the same protocol, (i.e., HTTP in our case). If an HTTP-like Botnet communication has only one end using HTTP, it is classified into the asymmetric protocol type.

2) Forged Content: If an HTTP-like Botnet communication uses the forged content to evade the security devices or human's eyes, it is classified into the forged content type. For example, a file with filename extension specified as RAR file, however, there is no RAR file header data in that file.

3) Unusual Protocol Parameters: If an HTTP-like Botnet communication has unusual protocol parameters, it is classified into the unusual protocol parameter type. In general, the HTTP request headers from a browser usually include correct information such as Accept, Accept-Language, User-Agent, Accept-Encoding and Host. The HTTP server then produces a corresponding response to the browser based on the request headers. For example, a suspected host with only IE browser installed, however, this host sends HTTP requests with its User-Agent specified as Opera 9.64.

4) TCP Port 80 Used but Not HTTP: If an HTTP-like Botnet communication employs the TCP port 80 but does not

follows the HTTP protocol, it is classified into the TCP-port-80-used-only type. This malware usually adopts TCP port 80 connections as the covert channels to evade the basic firewall and IDS.

5) Similar to Real HTTP Applications: If HTTP-like Botnet communication in the malign traffic cannot find anything suspected or unusual, we still try to generate detection rule for it because it is a part of the Botnet communication in fact. The false positive problem will be dealt with the rule verification stage.

B. The Input and Output Data in ARGS

In this section, we introduce the main data used in ARGS. They are MT, BT, KRS, and the newly produced and optimized Rule Set.

1) Malign Traffic (MT): Malign Traffic (MT) comprises the network packets of the malware or bot, which we collected in county-level Taiwan Academic Network (TANet) through trap sets such as honeypots. When we get a malware or bot sample, we execute it in our sandbox machine and record its network I/O to collect the MT. In the implementation of rule generation, each MT for a malware or bot is a separate pcap file in TcpDump format. In general, the size of the single MT is usually smaller than 1 MB. The details of malware collection and MT recording are not in the scope of this paper. In this paper, we assume the MT is a ready item.

2) Benign Traffic (BT): Benign Traffic (BT) is the collection of harmless normal network traffic in daily life, which contains the network behaviors of humans (e.g., web browsing) and of normal network devices (e.g., automatic updates). We utilize the BT to check whether our generated rules can lead to false positives. Considering the fact that rules validation is one of the main time consumption in ARGS, we would use only parts of the traffic collected from 30 clean PCs (without malware/bot infected) and their operators in two weeks as our BT, whose size is about 284 MB. The details of our BT recording are not in the scope of this paper. In this paper, we assume the BT is a ready item too.

3) Known Rule Set (KRS): Before we execute the rule generation flow, we collect Botnet detection rules as our KRS from the Emerging rule set [3] and Snort official rule set [2]. If the new MT cannot be matched by the KRS, we would try to generate rules for it because not all new malicious network behaviors can be matched by old/known rule set.

4) Newly Produced and Integrated Rule Set (NPIRS): The NPIRS is prepared for false-positive verification only. It includes the newly produced rules and the KRS. If NPIRS matches BT, it means there may be a false-positive and needs manual processing. If NPIRS can pass the verification, the newly produced rules are able to be added into the KRS.

5) Optimized Rule Set (ORS): The ORS is the optimized NPIRS. The number of rules in the ORS is usually less than that of NPIRS. In general, ORS can provide a better performance.

C. The Botnet Detection Process

At this stage, we employ Snort as our matching engine with KRS to check whether the MT can be matched. If the result is positive, the KRS is assumed to have the ability to detect the MT and the ARGS process is terminated. If the result is negative, the process continues to the next stage.

D. Traffic Selector (TS)

The most important task before generating the rule is to choose the matching packets in MT. The TS selects target packets with the mentioned HTTP-like Botnet traffic types because the network behaviors of the current HTTP-like malware may not exactly identical to the normal browsers. Therefore, we have a chance to choose the matching targets without complex algorithms and much processing time.

E. Traffic to Rule Conversion

For the good compatibility to existed network security systems, the generated rule set uses Snort format. We adopt the two matching options to generate string-based patterns of our rules. They are CONTENT and URICONTENT. The CONTENT is used to match general patterns and the URI-CONTECT is used to match the HTTP URI.

F. The Rule Verification Process

We use the BT to test the NPIRS and the ORS to discover false positives. In the case of a false positive, we need to adjust the rule set manually. Otherwise, newly generated rules can be added to KRS.

G. Rule Optimization

We usually choose the longer matching patterns to reduce the false positive rate of rules; however, large number of this kind of rules would reduce the performance of NIDS. To address this issue, we use two techniques to cluster the rules according to their similarity of hostname, URI and the type of malware. First, we use the token-subsequence algorithms [4] to merge the rules which are literally similar with each other. Besides, we also introduce Support Vector Machine (SVM) to classify the rules, and then reasonably merge them. In ARGS, we include the library LIBSVM[10] to train our SVM model. Considering the fact that SVM can only process the data with vector format, we extract and quantify these properties in three parts of the rule, including hostname, URI and the name of malware.

After converting the rules into corresponding vectors, we can input them to SVM to train the classifier of malware. With this classifier, we can reasonably merge the group of rules which have similarity network behavior.

IV. Implementation and Experiments

The hardware system used for evaluation in this study was an HP ProLiant DL360 G6 with 4G DRAM. The CPU was an Intel Xeon E5504 2.00GHz with a cache size of 4096 KB. The software environment was Fedora Linux 11 and the GCC version was 4.4.1.

A. The Rule Generating Performance Comparison

We used the real MTs not only to evaluate the generating performance (Table I) but to explore the performance at a higher MT number. We synthesized 10,000 MTs for test case 3 of Table I. The MTs of case 3 are the mutations including possible noise traffic generated by malware of case 2. The main time consumption of our method is in the rule verification stage and the size of BT is the key to the performance. In this experiment, the size of BT is fixed to 284 MB.

TABLE I
GENERATING TIME WITH DIFFERENT MT NUMBERS

	Case 1 (Real)	Case 2 (Real)	Case 3 (synthesized)
MT number	100	1,000	10,000
Our Method	13sec	31sec	4min43sec

B. The Generated Rule Set Quality Comparison

We used false positive rate to evaluate the quality of rule set. Table II presents the experiment results. We employed the raw generated rule sets of the case 2 in the Table I as the KRSes for this experiment. The raw generated rule sets are the sets had not been manually tuned. If the KRSes had been manually tuned, the false positive rates are all zeros. The version of our testing Snort was 2.8.5.2. The BT set of case 1 based on the recording of one day and the BT set of case 2 based on two weeks. We could understand that our scheme required lesser time to get the similar quality by means of tables I and II.

TABLE II
FALSE POSITIVE RATE COMPARISON TABLE

	Case 1 (Real)	Case 2 (Real)
BT size	284M (1 day)	146G (2 weeks)
Our Raw Generated Rule Set	0%	0.27%

C. Malware Traffic Detection Comparison

A zero-day malware means the malware cannot be detected because no rule to do so in that time. If a malware in its zero-day period, it is absolutely harmful. By ARGS, we can generate the corresponding rules as fast as possible and help to shorten the zero-day period. For understanding the zero-day malware ratio and the difference between ARGS and SourceFire NIDS rule set, we tested them with the MT set, which generated by 100 malware random selected form honeypots. By the collected MT and ARGS, we generated the corresponding rules in time and got the 100% detection rate but the latest SourceFire Rule Set at the time got 68% only. This experiment shows ARGS is superior to the SourceFire NIDS in matching new malware traffic.

To know the performance of rules generated by ARGS, we applied the rules generated by ARGS to a county network of TANet for one month. We counted the utilized rule number of each day of that month to understand the utilization of rule set. The result shows that the average number of utilized rules in each day is about 75, which means about 10%-15% rules of our rule package are utilized every day. Since the generated rule set can be improved in time and can track the trend of malware communication, we can keep the utilization and prevent the aging of rule set. By this practical evaluation, we know that the rules generated by ARGS have excellent performance in detecting the network behavior of malware.

V. CONCLUSION

In this paper, we explored the network behavior differences between the HTTP-like Botnet and normal Web browsing. Using this information, we designed a practical scheme to generate HTTP-like Botnet detection rules. The rule optimization function can be triggered conditionally to maintain the rule set quality and the matching performance. The false positive rate of generated rules is very low and this generating system can help to shorten the zero-day attack period due to its high generating performance. In practical evaluation, the statistical data shows that the rules generated by ARGS can help to detect malware activities in real network.

ACKNOWLEDGMENT

This work was supported by Ministry of Science and Technology (MOST) of Taiwan under the grant numbers 101-2221-E-007-048-MY3 and 103-2622-E-009-012.

REFERENCES

[1] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "Bothunter: detecting malware infection through ids-driven dialog correlation," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2007, pp. 12:1–12:16. [Online]. Available: http://portal.acm.org/citation.cfm?id=1362903.1362915

[2] Snort. [Online]. Available: http://www.snort.org/

[3] Emerging threats.net open rulesets. Emerging Threats.net. [Online]. Available: http://rules.emergingthreats.net/open/

[4] J. Newsome, B. Karp, and D. Song, "Polygraph: automatically generating signatures for polymorphic worms," in *Security and Privacy, 2005 IEEE Symposium on*, May 2005, pp. 226 – 241.

[5] R. Muthuregunathan, S. S., S. R., and R. S.R., "Efficient snort rule generation using evolutionary computing for network intrusion detection," *Computational Intelligence, Communication Systems and Networks, International Conference on*, vol. 0, pp. 336–341, 2009.

[6] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 26–26. [Online]. Available: http://portal.acm.org/citation.cfm?id=1855711.1855737

[7] G. K. Venkatesh and R. A. Nadarajan, "Http botnet detection using adaptive learning rate multilayer feed-forward neural network," in *Information Security Theory and Practice. Security, Privacy and Trust in Computing Systems and Ambient Intelligent Ecosystems*. Springer, 2012, pp. 38–48.

[8] D. Zhao, I. Traore, B. Sayed, W. Lu, S. Saad, A. Ghorbani, and D. Garant, "Botnet detection based on traffic behavior analysis and flow intervals," *Computers & Security*, vol. 39, pp. 2–16, 2013.

[9] A. Zarras, A. Papadogiannakis, R. Gawlik, and T. Holz, "Automated generation of models for fast and precise detection of http-based malware," in *Privacy, Security and Trust (PST), 2014 Twelfth Annual International Conference on*, July 2014, pp. 249–256.

[10] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, pp. 27:1–27:27, May 2011. [Online]. Available: http://doi.acm.org/10.1145/1961189.1961199