

Detecting Worm Propagation Using Traffic Concentration Analysis and Inductive Learning*

Sanguk Noh¹, Cheolho Lee², Keywon Ryu³, Kyunghee Choi³, and Gihyun Jung⁴

¹School of Computer Science and Information Engineering,
The Catholic University of Korea, Korea
sunoh@catholic.ac.kr

²National Security Research Institute, Korea
chlee@etri.re.kr

³Graduate School of Information and Communication,
Ajou University, Korea
{ryujeen, khchoi}@ajou.ac.kr

⁴Division of Electronics Engineering,
Ajou University, Korea
khchung@ajou.ac.kr

Abstract. As a vast number of services have been flooding into the Internet, it is more likely for the Internet resources to be exposed to various hacking activities such as Code Red and SQL Slammer worm. Since various worms quickly spread over the Internet using self-propagation mechanism, it is crucial to detect worm propagation and protect them for secure network infrastructure. In this paper, we propose a mechanism to detect worm propagation using the computation of entropy of network traffic and the compilation of network traffic. In experiments, we tested our framework in simulated network settings and could successfully detect worm propagation.

1 Introduction

As a vast number of services have been flooding into the Internet, it is more likely for the Internet resources to be exposed to various hacking activities such as Code Red in 2001 and SQL Slammer worm in 2003, respectively [3, 7, 8, 10]. Since various worms quickly spread over the Internet using self-propagation mechanism, it is crucial to detect worm propagation and protect them for secure network infrastructure. In our previous work, we have suggested our method to detect distributed denial-of-service (DDoS) attacks through the analysis of network traffic, and reported its resulting performance in [11]. We, in this paper, extend our efforts to detect worm propagation using the computation of entropy of network traffic and the compilation of network traffic.

* This work has been supported by the Korea Research Foundation under grant KRF-2003-041-D20465, and by the KISTEP under National Research Laboratory program.

For self-propagation, most worms scan target ports, identify a specific port, connect a host using the port chosen, for example, tcp/80 port in case of Code Red, and infect the host [3, 10]. The infected host, further, tries to copy its worm to others. From this phenomenon, we could observe that the egress network traffic on the specific port of the infected host abruptly increases. Our intuition to this observation leads to the analysis of network traffic using the computation of entropy representing the volatility of network traffic. To detect the worm propagation, then, we compile a pair of network traffic and worm infection into a set of reactive rules using various learning algorithms.

There has been lots of related work to analyze worm pattern and detect worm propagation [1, 14, 15]. Most approaches model the collateral effects of self-propagation while our approach focuses on the worm propagation itself. In other words, since our framework models the pattern of network traffic on infected hosts, we could apply our framework to general worm propagation situations, regardless of the various behavior patterns of unknown worms.

In the next section, we will address the question of how to detect worm propagation. Towards this end, this paper proposes traffic concentration analysis and network traffic compilation method. In section 3, we describe experimental setting and experimental results. We summarize our conclusions and further research issues in section 4.

2 Detecting Worm Propagation

To identify worm propagation on the Internet, we need to keep track of session information, as consisting of ports and IP addresses for source and destination hosts. A worm randomly generates a pool of IP addresses for destinations, scans their exploiting ports, for example, port tcp/80 in case of Code Red, and exploits vulnerabilities of the open ports. In this section, we will introduce a framework of traffic concentration analysis to represent any changes of the session information in the presence/absence of a worm. Further, using inductive learning algorithms, we compile a tuple of session information and worm infection into rules to detect self-propagating worms.

2.1 Traffic Concentration Analysis

Based upon the observation of the features of a fast spreading worm, as mentioned in section 1, it is notable that the randomness (or volatility) of IP addresses increases in order to infect vulnerable hosts as many as possible, and the randomness of ports decreases, since the exploiting ports for propagation are limited. To present the randomness of network traffic, we use the computation of entropy, which is originally reported in [4], as follows:

$$E(s) = - \sum_{i=1}^n p_i \log_2(p_i), \quad (1)$$

where

$S = \{\text{srcAddr}, \text{srcPort}, \text{dstAddr}, \text{dstPort}\};$
 $E(s)$ is the entropy of the class $s \in S$;
 p_i is the probability of the i -th member in a class s .

In equation 1, S can be one of session information, and p_i can be the ratio of the number of the member i of a class, say, srcAddr , in S to the total number of the members of the class. The entropies of session information provide the randomness of network traffic. The lower entropy of target addresses and ports describes a network status that the less number of target addresses and ports are shown given a set of sessions.

2.2 Detecting Worms Using Inductive Learning Algorithms

We propose a network monitoring agent architecture, as consisting of a packet collecting agent and an adaptive reasoning agent - an alarming agent - that analyze network traffic, detect the spread of worms upon the network traffic concentration, and finally issue an alarm in case of the worms. To inspire adaptability into our alarming agents, we use various machine learning algorithms and, using learning techniques, compile the models of network traffic into a set of rules.

Bayes rule examines whether or not a property observed as evidence belongs to a specific hypothesis (or class), given a set of data distribution. Bayes theorem [6] can be defined as follows:

$$P(h_j | x_i) = \frac{P(x_i | h_j) P(h_j)}{\sum_{j=1}^m P(x_i | h_j) P(h_j)}, \quad (2)$$

where

A set of observable attributes, $X = \{x_1, x_2, \dots, x_n\}$;
 A set of hypotheses in a domain, $H = \{h_1, h_2, \dots, h_m\}$;
 $P(h_j|x_i)$ is the posterior probability of the hypothesis h_j , $h_j \in H$,
 given that x_i , $x_i \in X$, is an observable event.

In our framework, the set of observable attributes, X , consists of the entropies of session information, i.e., ports and addresses for source and destination hosts, and the hypotheses are either no worm infection or worm infection. Given the set of data as evidence Bayes rule allows us to assign probabilities of hypotheses, $P(h_j|x_i)$. Our alarming agents compute $P(h_j|x_i)$ during on-line, and set an alarm when the probability of worm infection given input is greater than that of no infection.

The decision tree approach such as C4.5 [12] and CN2 [2] is to divide the domain space into classified regions, which are given by a set of classes $C = \{c_1, c_2, \dots, c_m\}$. The basic idea of the decision tree-based induction algorithms finds out a set of ordered attributes, $X = \{x_1, x_2, \dots, x_n\}$, which split the datasets into a correct classification with the highest information gain first. A decision tree has internal nodes labeled with attributes $x_i \in X$, arcs associated with the parent attributes, and leaf nodes corresponding to classes $c_j \in C$.

The decision tree-based induction algorithms, thus, generate a tree representing a model of network traffic in the simulated network setting. Once the tree is built using training data, the optimal rules from the tree can be obtained and are applied to the new network traffic to determine whether or not a host is infected by a worm.

Let S be the set of traffic states that the adaptive reasoning agent can discriminate among. Let L be the set of compilation methods (learning algorithms) that the agent employs. Given a learning method $l \in L$, a compilation procedure of an adaptive reasoning agent implements a function $\rho_l: S \rightarrow \{\text{worm infection}, \text{no worm infection}\}$, representing whether or not a worm spreads in the state $s \in S$. Thus, various machine learning algorithms compile the models of network traffic into different functions ρ_l . We generate the training examples for these learning algorithms from TCP-based network environments.

3 Experiments

We have implemented a simulated network environment using SPECweb99 [13], Code Red [3], and libpcap [9]. In the simulated, Web-based setting, the SPECweb99 located in Web clients generates web traffic, the Code Red on Web server simulates worm's self-propagating activities, and the libpcap used by a packet collecting agent captures the stream of network traffic. While the Web clients request the Web server that they should be serviced, the Code Red performs network scanning for its self-propagation.

We construct the simulated network environment on LINUX machines, which consist of Web server using Apache, Web clients using SPECweb99, Code Red using newly created data set, based on the worm propagation pattern in [3], a network monitoring device including a packet collecting agent and an alarming agent (700 MHz Pentium III, 256 MB memory), and the network bandwidth of 100 Mbps. In the simulated network architecture, our agents working on the network monitoring device capture the egress network traffic from the infected Web server, and set an alarm if they detect a worm.

We tested our framework in the simulated network environment and measured traffic concentration degrees of network traffic from the Web server. The network traffic models were generated in two settings: the normal Web server without worm's activity and the Web server infected by Code Red. For each network traffic setting, we changed two factors, Simultaneous Connections (SC) and Requests per Connection (R/C), to get various Web traffic patterns. The SC indicates the number of HTTP connections at a given time, which approximates the number of users. The R/C represents the number of requests to be issued in a HTTP connection. In the experiment, we used 5, 10, 25, 50, 75 and 100 for SC and 1, 2, 5, and 10 for R/C. The sampling time to compute traffic concentration degrees was 1 second.

3.1 The Analysis of Network Traffic

Using our Traffic Concentration Analysis (TCA) mechanism, we measured the degree of network traffic concentration on the egress network traffic from the web server, in absence of worm propagation and in presence of worm propagation, respectively. The experimental results of normal Web traffic are illustrated in figure 1 (a). Even if SC ranges from 5 to 100, the results are almost identical.

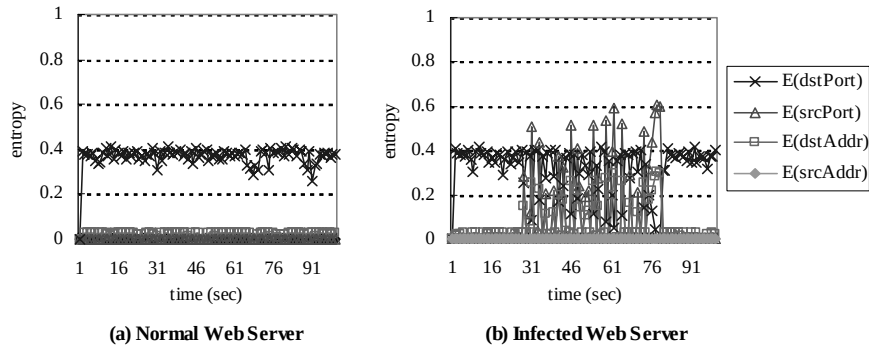


Fig. 1. Entropy distributions in the normal Web server and the infected one. Entropy changed when infected by Code Red. The self-propagation activities were performed from 25 to 75 seconds in (b).

In figure 1 (a), since there was only one server, both of $E(srcAddr)$ and $E(srcPort)$ were clearly 0 while $E(dstPort)$ was, on the average, 0.3483, and $E(dstAddr)$ was almost 0. This result indicated that a specific web server using a tcp/80 port made connections with various destination ports for a unique session between the web server and many web clients. On the other hand, figure 1 (b) presents the distinctive pattern of network traffic when the Code Red spread. During the worm propagation ranging from 25 to 75 seconds, the average of $E(srcPort)$ was 0.3212, and the average of $E(dstAddr)$ was 0.1175. The entropy of destination port, $E(dstPort)$, varied from 0 to 0.4, due to the fact that the worm process on the infected hosts used a specific port, i.e., tcp/80 in our experiment, on the destination.

3.2 Worm Detection Performance

To construct compiled rules for our alarming agents, we used three machine learning algorithms: C4.5 [12], CN2 [2], and naïve Bayesian classifier [5]. C4.5 represents its output as a decision tree, and the output of CN2 is an ordered set of if-then rules. For the Bayesian classifier, the results are represented as rules specifying the probability of occurrence of each attribute value given a class [2], in our case “attacks” and “no attacks.”

Using the three learning algorithms [6] and the training examples as inputs, we could get the compiled rules. To evaluate the quality of various rule sets generated by different learning algorithms the performance obtained was expressed in terms of the

ratio of $\{\text{total number of alarms} - (\text{number of false alarms} + \text{number of missed alarms})\}$ to the total number of alarms. The false alarm is defined as the alert turns on when the worm propagation attack does not occur, and the missed alarm is defined as the alert does not turn on when the worm propagation does occur.

To find a meaningful size of the training set which could guarantee the soundness of the learning hypothesis, we generated several sets of training examples whose size was 400 through 2400 by interval 200, respectively. The resulting performances (accuracy) and the sizes of training examples are shown in figure 2.

For benchmark, we measured the detection performance using Valdes' work [15], which considers only source addresses and destination ports of all the ingress packets through routers into the network infrastructure. In figure 2, the size of training examples was 2600 tuples for both mechanisms. As far as using TCA mechanism, the naïve Bayesian classifier showed the best performance of 90%, on the other hand, using Valdes' mechanism, the best performance achieved by the naïve Bayesian classifier was 82%. It turned out that our mechanism slightly outperformed the Valdes' work in the simulated, small scaled network setting.

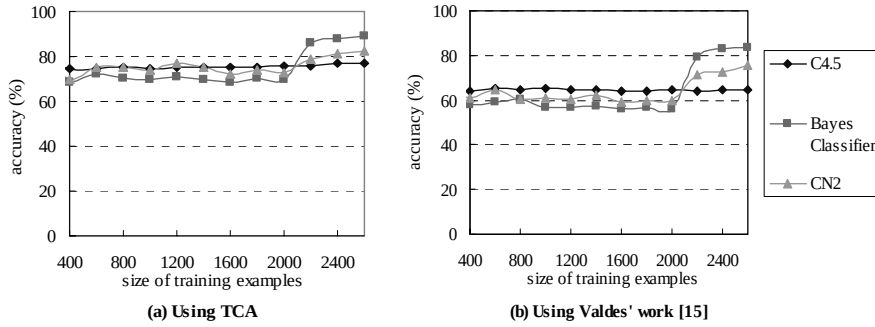


Fig. 2. Worm detection performance using the compiled rules in TCA and Valdes' work

We analyzed the performance results using the standard analysis of variance (ANOVA) method. Since naïve Bayesian classifier showed best performances, as depicted in figure 2, in both schemes, its ten best performances were used for ANOVA test. The average performance of worm monitoring through TCA was 87.61 ± 1.49 while that of worm monitoring through Valdes' work was 81.74 ± 2.12 , and then, the resulting f value of ANOVA method was 17.09. Since the computed value of $f=17.09$ in ANOVA exceeds $8.28 (=f_{.01,1,18})$, we know that the two mechanisms were not all equally effective at the 0.01 level of significance, i.e., the differences in their performance were not due to chance with probability of 0.99. We, thus, conclude that our alarming agent's performance using the TCA mechanism was better than that of Valdes' worm detection mechanism.

4 Conclusions

In this paper, we have presented the network traffic concentration in terms of entropy, when a worm quickly spreads across Web servers, and compiled a pair of network traffic concentration and the presence/absence of worm into a set of state-action rules using the inductive learning algorithms. We applied the resulting compiled rules to a simulated Code Red, and tested the detection performance of our framework. The experimental results showed that our scheme slightly outperformed a benchmarked one by about 10%. We believe that our framework is applicable when a randomness of network traffic becomes prohibitive, and is valuable to detect self-propagating worms on the Internet. For future work, we will extend our framework to various kinds of worm propagation in scaled-up network settings. We are analyzing the features of non autonomous propagation worms, for example, email attachment worms using SMTP protocol.

References

1. Berk, V.H. et al.: Using Sensor Networks and Data Fusion for Early Detection of Active Worms. SPIE AeroSense (2003)
2. Clark, P. and Niblett, T.: The CN2 Induction Algorithm. *Machine Learning Journal* 3(4) (1989) 261-283
3. Danyliw, R. and Householder, A.: CERT Advisory CA-2001-19 "Code Red" Worm Exploiting Buffer Overflow in IIS Indexing Service DLL. CERT Coordination Center (2001)
4. Gray, R.M.: Entropy and Information Theory. Springer-Verlag (1990) 39-40
5. Hanson, R., Stutz, J., and Cheeseman, P.: Bayesian Classification Theory. Technical Report FIA-90-12-7-01, NASA Ames Research Center, AI Branch (1991)
6. Holder, L.: ML v2.0: Machine Learning Program Evaluator. available on-line: <http://www-cse.uta.edu/~holder/ftp/ml2.0.tar.gz>
7. Houle, J.K., and Weaver, M.G.: Trends in Denial of Service Attack Technology. CERT Coordination Center (2001)
8. Lan, K. et al.: Effect of Malicious Traffic on the Network. PAM (2003)
9. Lawrence Berkeley National Labs Network Research Group.: libpcap. available on-line: <http://ftp.ee.lbl.gov>
10. Moore, D. et al.: The Spread of the Sapphire/Slammer Worm. available on-line: <http://www.cs.berkeley.edu/~nweaver/sapphire/> (2003)
11. Noh, S. et al.: Detecting Distributed Denial of Service (DDoS) Attacks through Inductive Learning. Lecture Notes in Computer Science, Vol. 2690, Springer-Verlag (2003) 286-295
12. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers (1993)
13. Standard Performance Evaluation Corporation: SPECweb99 Benchmark. available on-line: <http://www.spec.org/osg/web99>
14. Toth, T. and Kruegel, C.: Connection-history based Anomaly Detection. The 2002 IEEE Workshop on Information Assurance and Security (2002)
15. Valdes, A.: Entropy Characteristics of Propagating Internet Phenomena. The Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection (2003)