Artif Life Robotics (2011) 16:44–47
DOI 10.1007/s10015-011-0880-5

ORIGINAL ARTICLE

Takeshi Okamoto

An artificial intelligence membrane to detect network intrusion

Received and accepted: February 10, 2011

Abstract We propose an artificial intelligence membrane to detect network intrusion, which is analogous to a biological membrane that prevents viruses from entering cells. This artificial membrane is designed to monitor incoming packets and to prevent a malicious program code (e.g., a shellcode) from breaking into a stack or heap in a memory. While monitoring incoming TCP packets, the artificial membrane constructs a TCP segment of incoming packets, and derives the byte frequency of the TCP segment (from 0 to 255 bytes) as well as the entropy and size of the segment. These features of the segment can be classified by a data-mining technique such as a decision tree or neural network. If the data-mining method finds a suspicious byte sequence, the sequence is emulated to ensure that it is just a shellcode. If the byte sequence is a shellcode, the sequence is dropped. At the same time, an alert is communicated to the system administrator. Our experiments examined seven data-mining methods for normal and malicious network traffic. The malicious traffic included 114 shellcodes, provided by the Metasploit framework, and including 10 types of metamorphic or polymorphic shellcodes. In addition, real network traffic involving shellcodes was examined. We found that a random forest method outperformed all the other data-mining methods and had a very high detection accuracy, including a true-positive rate of 99.6% and a false-positive rate of 0.4%.

Key words Network intrusion detection · Malicious software · Shellcode · Data mining

T. Okamoto (✉)
Department of Information Network and Communication, Kanagawa Institute of Technology, 1030 Shimo-ogino, Atsugi, Kanagawa 243-0292, Japan
e-mail: take4@nw.kanagawa-it.ac.jp

This work was presented in part at the 15th International Symposium on Artificial Life and Robotics, Oita, Japan, February 4–6, 2010

1 Introduction

Antivirus systems protect computers and networks from malicious programs, such as computer viruses and worms, by discriminating between malicious and harmless programs and by removing only the former. Therefore, antivirus systems can be considered as a computer's immune system.

An innovative method called a "virus throttle"[1] has been found to halt high-speed worms without affecting normal network traffic. We have previously proposed a "worm filter" to prevent the spread of both slow- and high-speed worms.[2] This worm filter limits the number of unacknowledged requests rather than the rate of connections to new computers. In addition, we have proposed an immunity-based anomaly detection method to detect worms in network traffic.[3]

All of these methods monitor outgoing packets from an internal network, i.e., they detect internal anomalies. Other methods are needed to monitor incoming packets and to detect intrusive attacks. The four types of method used to detect intrusive attacks include pattern matching,[4] heuristic,[5] emulation,[6] and data-mining[7,8] methods. Since pattern-matching methods require signatures to detect intrusive attacks, they may miss new attacks owing to an absence of signatures. In addition, metamorphic and polymorphic codes can produce so many patterns that it may be difficult to cover them all.[9] Heuristic methods attempt to detect intrusive code sequences such as consecutive NOP sequences (i.e., NOP sleds) and sequences that get a program counter (i.e., getPC). However, some NOP sleds are polymorphic,[9,10] and intrusive attacks may not get the program counter. Emulation methods, which emulate incoming packets as program code, can correctly detect an intrusive program code, but these processes are very slow. Data-mining methods use a classifier, such as a decision tree or neural network, to distinguish between benign and malicious traffic using the features of network traffic. Although these methods detect malicious traffic at a high rate, their false-positive rates may be high.

Here we propose an artificial intelligence membrane to detect network intrusion, which is analogous to a biological membrane that prevents viruses from entering cells. The artificial membrane was designed to prevent a malicious program code (e.g., a shellcode) from breaking into a stack or heap in the memory. Our experiments examined seven data-mining methods for normal and malicious network traffic. The malicious traffic included 114 shellcodes, provided by the Metasploit framework,[10] and 10 kinds of metamorphic and polymorphic shellcode. In addition, real network traffic involving intrusive network attacks was examined.

2 Network intrusion

Most network intrusion attacks are composed of a vulnerability attack and a shellcode execution (Fig. 1). The vulnerability attack is used for network intrusion, following which the shellcode is executed. The shellcode is a tiny program code for operating anything on the computer, such as the download of malicious software and its execution.

The appearance of a shellcode is often disguised by an encoder (Fig. 1). This type of shellcode is called a metamorphic or polymorphic shellcode. In metamorphic shellcodes, a set of instructions is replaced by an equivalent set of different instructions, whereas in polymorphic shellcodes, a set of instructions is hidden by encryption (Fig. 1). These techniques make such shellcodes difficult to detect because their appearances differ from each other and are different on each occasion. Although signature-based detection works in some cases, polymorphism will eventually defeat such detection methods.[9]

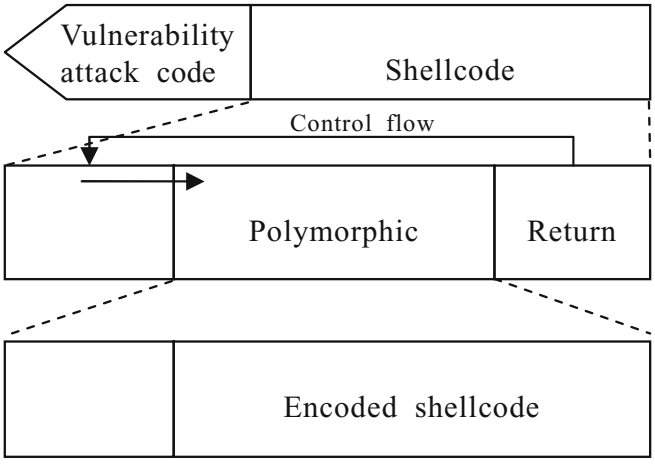

Fig. 1. Structure of a polymorphic shellcode. The NOP area contains consecutive NOPs, indicating no operation and control flowing to the shellcode area. The return address area contains the address to which the program counter returns (i.e., the address of the NOP area). If a vulnerability attack is successful, the program counter will jump to the NOP area and enter the shellcode area

3 An artificial intelligence membrane to detect network intrusion

To detect shellcodes, we propose an artificial intelligence membrane to detect network intrusions. The membrane plays a role similar to a cell membrane that protects a cell from nonself molecules.

Figure 2 illustrates the algorithm of the artificial intelligence membrane. The artificial membrane monitors incoming TCP packets, and it constructs a TCP segment consisting of incoming packets. It then derives a byte frequency of the TCP segment (from 0 to 255 bytes) as well as the entropy and size of the segment. These features of the segment are classified by a data-mining method such as a decision tree or a multilayer perceptron (i.e., a neural network). If the data-mining method identifies a suspicious segment, that segment is emulated to ensure that it includes just a shellcode. The emulation plays the role of eliminating false-positives. If the segment includes the shellcode, the segment is dropped. At the same time, the system administrator is alerted.

Emulation is performed by "libemu 0.2.0," a small library offering ×86 architecture emulation for shellcode detec-

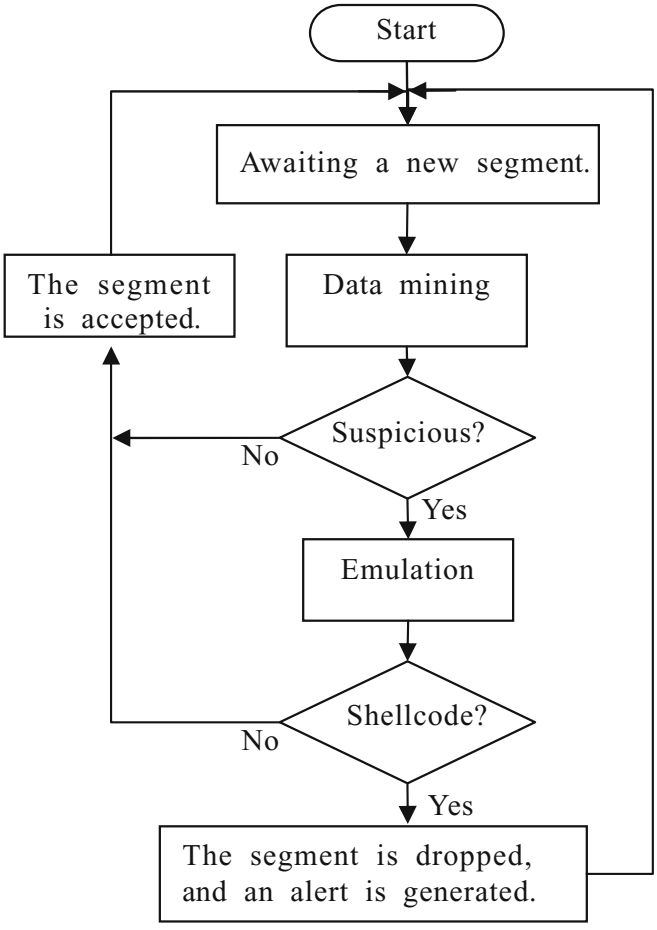

Fig. 2. Algorithm of the artificial intelligence membrane. The parameters of the data mining method are trained in advance

tion.[12] This libemu has been partially modified for brute-force detection of a suspicious byte sequence. Thus, the modified libemu can detect all ×86 architecture-based shellcodes provided by the Metasploit framework, including all metamorphic and polymorphic shellcodes. Although emulation is very slow, the entire performance of the proposed method is not slow because normal packets are eliminated in advance by the data-mining method, and most of the packets would not be suspicious.

4 Evaluation of detection accuracy

To find the most accurate data-mining methods, we evaluated the detection accuracy of seven methods equipped with Wakaito Environment for Knowledge Acquisition (WEKA)[13]: an instance-based learner (IBk), two decision trees (J48 and random forest), naïve bayes, an inductive rule learner (JRpp), and a support vector machine using sequential minimal optimization (SMO). The algorithm of each data-mining method has been described.[13] All parameters of the data-mining methods were default settings of WEKA.

Each evaluation has been used for 10-fold cross-validation. The data set was randomly divided into 10 subsets, with nine subsets used for training and one for testing. The process was repeated 10 times for every combination. This methodology can be used to evaluate the robustness of a given approach to detecting shellcodes. Experiments were performed on dual Intel Xeon E5410 2.33 GHz processors with 12 GB RAM. The operating system was Debian GNU/Linux Squeeze.

4.1 Evaluation of simulated traffic

The experimental data consisted of normal http traffic and simulated malicious http traffic. The normal traffic consisted of 544 segments, whereas the malicious traffic was simulated by combining one normal http segment with one shellcode for each of the 114 shellcodes provided by the Metasploit framework, version 3.5.1. Other types of malicious traffic included metamorphic or polymorphic shellcodes encoded by 10 engines: "ADMmutate,"[14] "CLET,"[11] "alpha mixed," "alpha upper," "call4 dword xor," "context cpuid," "countdown," "fnstenv mov," "jmp call additive," and "shikata ga nai," with the last eight engines provided by the Metasploit framework.[10]

"CLET" can disguise a shellcode as normal network traffic by padding bytes close to the statistical properties of the normal traffic between the shellcode and the return address sequence (Fig. 1). In this experiment, the padding size was 500 bytes. Larger padding results in closer byte frequency between normal and malicious traffic,[9] but it also makes it more difficult for the shellcode to control the target computer because the size of the buffer is not always sufficient to intrude into a stack in a memory. Note that only CLET-encoded shellcodes were encoded in advance by the "shikata ga nai" encoder to remove 0×00 byte codes from the original shellcodes.

In addition, the encoders of "alpha mixed" and "alpha upper" can disguise a shellcode as real traffic by recoding the shellcode in a form that contains bytes matching the statistical properties of real traffic.

The simulated traffic was examined using WEKA. Table 1 shows the detection accuracy of simulated traffic. The true-positive rate (TPR) was defined as the rate at which a suspicious segment was correctly classified as suspicious, whereas the false-positive rate (FPR) was the rate at which a normal segment was falsely classified as suspicious. We found that the random forest method outperformed all other data-mining methods (Table 1), with a TPR of 99.9% and an FPR of 0%.

Table 2 shows the detection accuracy of metamorphic and polymorphic shellcodes. The data-mining method used in this experiment was a random forest method. All detection accuracies were very high, with all metamorphic and polymorphic shellcodes other than (4) having a TPR of 100% and an FPR of 0%. In addition, the polymorphic shellcodes encoded by CLET had a TPR of 100% and an FPR of 0%, whereas metamorphic shellcodes encoded by "alpha mixed" and "alpha upper" had a TPR ≥ 99.8% and

Table 1. Detection accuracy of simulated traffic (not all shellcodes were encoded). Each value is a weighted average

	IBk (%)	J48 (%)	Random forest (%)	Multilayer perceptron (%)	JRpp (%)	SMO (%)	Naïve bayes (%)
TPR	99.6	99.6	99.9	99.6	99	99.6	99.3
FPR	0.7	0.7	0	0.7	2.7	1.3	3.2

Table 2. Accuracy of detecting simulated traffic for metamorphic and polymorphic shellcodes. Each value is a weighted average. Shellcodes were encoded by the following engines: (1) "ADMmutate," (2) "CLET," (3) alpha mixed, (4) alpha upper, (5) call4 dword xor, (6) context cpuid, (7) countdown, (8) fnstenv mov, (9) jmp call additive, and (10) shikata ga nai

	(1) (%)	(2) (%)	(3) (%)	(4) (%)	(5) (%)	(6) (%)	(7) (%)	(8) (%)	(9) (%)	(10) (%)
TPR	100	100	100	99.8	100	100	100	100	100	100
FPR	0	0	0	0	0	0	0	0	0	0

Table 3. Accuracy of the detection and testing time of real traffic. Each value is a weighted average

	IBk	J48	Random forest	Multilayer perceptron	JRpp	SMO	Naïve Bayes
TPR	99.4%	99.3%	99.6%	99%	99.2%	98.9%	98.5%
FPR	0.6%	0.7%	0.4%	0.7%	0.8%	0.8%	1.2%
Time (s)	0.66951	0.00054	0.00071	0.20002	0.00049	0.00307	0.02604

an FPR of 0%, although the statistical properties of these shellcodes were similar to those of normal traffic.

4.2 Evaluation of real traffic

Experimental data were captured from a highly interactive honeypot on VMware Workstation 6.0.3. The guest operating systems were Microsoft Windows XP Professional, SP1 and SP2. After extracting segment data from the captured data, we examined all the segment data using the modified libemu, and found 1469 normal segments and 976 malicious segments, including shellcodes.

To evaluate the performance of the data-mining method, we examined all the above segments using the data-mining methods of WEKA. Table 3 shows the accuracy of the detection and testing time for real traffic. Again, we found that the random forest method outperformed all other data-mining methods, with a TPR of 99.6% and an FPR of 0.4%. Eventually, there would be no false-positives, because the suspicious segments can be analyzed by emulation.

5 Conclusions

We have proposed an artificial intelligence membrane to detect network intrusion. This membrane is analogous to a biological membrane, which prevents viruses from entering cells. Similarly, the artificial membrane prevents shellcodes from breaking into a memory.

Our experiments indicated that the random forest method outperformed all other methods. In addition, all metamorphic and polymorphic shellcodes were detected at a high rate. For real traffic, the TPR was 99.6% and the FPR was 0.4%. This high detection accuracy is considered to be due to the training with both normal and malicious traffic data.[9]

We are currently planning to implement this artificial intelligence membrane for practical use.

References

1. Williamson MM (2002) Throttling viruses: restricting propagation to defeat malicious mobile code. ACSAC Security Conference 2002, pp 61–68
2. Okamoto T (2005) A worm filter based on the number of unacknowledged requests. KES'05, LNAI 3682:93–99
3. Okamoto T, Ishida Y (2006) Towards an immunity-based anomaly detection system for network traffic. KES'06, LNAI 4252:123–130
4. Roesch M (1999) Snort: lightweight intrusion detection for networks. LISA'99, 229–238
5. Pasupulati A, Coit J, Levitt K, et al (2004) Buttercup: on network-based detection of polymorphic buffer overflow vulnerabilities. NOMS 1:235–248
6. Polychronakis M, Anagnostakis KG, Markatos EP (2007) Network-level polymorphic shellcode detection using emulation. J Comput Virol 2(4):257–274
7. Payer U, Teufl P, Lamberger M (2005) Hybrid engine for polymorphic shellcode detection. LNC S 3548(200):19–31
8. Masud M, Khan L, Thuraisingham B, et al (2008) Detecting remote exploits using data mining. IFIP 285:177–189
9. Song Y, Locasto ME, Stavrou A, et al (2007) On the infeasibility of modeling polymorphic shellcode. Proceedings of the 14th ACM CCS'07, pp 541–551
10. Metasploit project (2006) http://www.metasploit.com/
11. Detristan T, Ulenspiegel T, Malcom Y, et al (2003) Polymorphic shellcode engine using spectrum analysis. Phrack 11(61)
12. Baecher P, Koetter M (2007) libemu. http://libemu.carnivore.it/
13. Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques. Morgan Kaufmann, Los Altos, 2nd edn
14. K2 (2001) ADMmutate. http://www.ktwo.ca/ADM mutate-0.8.4.tar.gz