SCIENCE CHINA
Information Sciences

• RESEARCH PAPERS •

December 2010 Vol. 53 No. 12: 2461–2471
doi: 10.1007/s11432-010-4123-5

A malware detection model based on a negative selection algorithm with penalty factor

ZHANG PengTao[1,2], WANG Wei[1,2] & TAN Ying[1,2*]

[1]*Department of Machine Intelligence, School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871*, China;*
[2]*Key Laboratory of Machine Perception, Ministry of Education, Peking University, Beijing* 100871*, China*

Received March 21, 2010; accepted September 5, 2010

Abstract A malware detection model based on a negative selection algorithm with penalty factor (NSAPF) is proposed in this paper. This model extracts a malware instruction library (MIL), containing instructions that tend to appear in malware, through deep instruction analysis with respect to instruction frequency and file frequency. From the MIL, the proposed model creates a malware candidate signature library (MCSL) and a benign program malware-like signature library (BPMSL) by splitting programs orderly into various short bit strings. Depending on whether a signature matches "self", the NSAPF further divides the MCSL into two malware detection signature libraries (MDSL1 and MDSL2), and uses these as a two-dimensional reference for detecting suspicious programs. The model classifies suspicious programs as malware and benign programs by matching values of the suspicious programs with MDSL1 and MDSL2. Introduction of a penalty factor C in the negative selection algorithm enables this model to overcome the drawback of traditional negative selection algorithms in defining the harmfulness of "self" and "nonself", and focus on the harmfulness of the code, thus greatly improving the effectiveness of the model and also enabling the model to satisfy the different requirements of users in terms of true positive and false positive rates. Experimental results confirm that the proposed model achieves a better true positive rate on completely unknown malware and a better generalization ability while keeping a low false positive rate. The model can balance and adjust the true positive and false positive rates by adjusting the penalty factor C to achieve better performance.

Keywords penalty factor, negative selection algorithm, signature extraction, artificial immune system, malware detection

Citation Zhang P T, Wang W, Tan Y. A malware detection model based on a negative selection algorithm with penalty factor. Sci China Inf Sci, 2010, 53: 2461–2471, doi: 10.1007/s11432-010-4123-5

1 Introduction

With the rapid development of computer technology, new anti-malware technologies are required because malware is becoming more complex with a faster propagation speed and a stronger ability for latency, destruction, and infection.

Many companies have released anti-malware software, most of which is based on signatures and can detect known malware very quickly. However, the software often fails to detect new variations and

*Corresponding author (email: ytan@pku.edu.cn)

unknown malware. Based on metamorphic and polymorphous techniques, even a layman is able to develop new variations of known malware easily using malware automaton. Thus, traditional malware detection methods based on signatures are no longer suitable for new environments; as well, heuristics have started to emerge.

Data mining algorithms try to mine frequent patterns or association rules to detect malware using classic classifiers. This has led to some success. However, data mining loses the semantic information of the code and cannot easily recognize unknown malware.

For the past few years, applying immune mechanisms to computer security has developed into a new field, attracting many researchers. Forrest et al. [1] applied immune theory to computer abnormality detection for the first time in 1994. Since then, many researchers have proposed various different malware detection models and achieved some success.

The negative selection algorithm (NSA) is one of the most important algorithms in artificial immune systems (AIS). After deleting detectors that match "self", the NSA obtains a detector set, in which none of the items matches "self", and which is then used to detect malware. A traditional NSA assumes that all "self" is harmless and all "nonself" is harmful. However, in organisms this is not always the case. Taking cancer cells as an example, not all "self" is harmless; and similarly, not all "nonself" is harmful, for example, food. A computer security system, therefore, only has to identify dangerous malware instead of reacting to all "nonself".

Unlike danger theory, the proposed model detects malware through dangerous signatures extracted from programs. Instead of deleting "nonself" that matches "self", the negative selection algorithm with penalty factor (NSAPF) penalizes the "nonself" using penalty factor C and keeps these items in a library. In this way, the effectiveness of the proposed model is improved using the dangerous signatures that would have been discarded in the traditional NSA.

This paper is organized as follows. In section 2, various related work is introduced. Our model and experimental results are presented in detail in sections 3 and 4, respectively. In section 5, conclusions and future work are discussed.

2 Related works

Henchiri and Japkowicz [2] adopted a data mining approach to extract frequent patterns (FPs) for detecting malware. Based on intra-family support and inter-family support, they filtered FPs twice, trying to obtain more general FPs. They verified the effectiveness of their model using 5-fold cross validation showing some good results. Nevertheless, FPs are merely fixed-length bit strings with no definite meaning and cannot represent real malware signatures.

Tabish et al. [3] proposed a malware detection model using statistical analysis of the byte-level file content. This model is not based on signatures. It neither memorizes specific strings appearing in the file content nor depends on prior knowledge of file types. Although better results can be obtained in this way, there is a high false positive rate, because this method only uses statistics on the training set.

Researchers have proposed different kinds of heuristic approaches to detect malware with some success [4–7].

With the development of immunology, immune mechanisms have begun to be applied in computer security. Forrest et al. first proposed an NSA to detect abnormal modification on protected data [1] and later applied it to UNIX process detection [8]. Since then, more and more researchers have devoted themselves to the study of computer immune systems based on immune mechanisms and many immune based computer malware detection models have been proposed [9–13].

Li [14] proposed a dynamic detection model for computer viruses based on an immune system. Through dynamic evolution of "self", an antibody gene library, and detectors, this model reduces the size of the "self" set, raises the generating efficiency of detectors, and resolves the problem of detector training time being exponential with respect to the size of "self".

Wang et al. [15] have sought to use the relativity of different features in a virus sample to identify unknown viruses. They succeeded in constructing a hierarchical artificial immune model (HAIM). How-

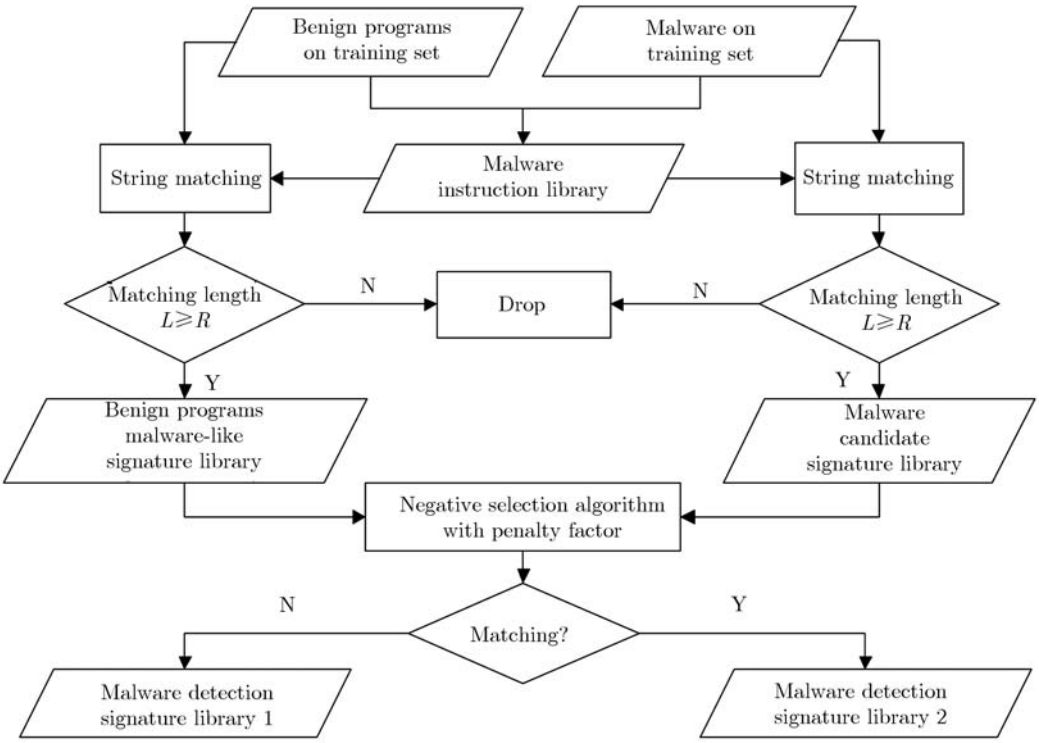

Figure 1 Flowchart for MSEM.

ever, the problem of repeated storage of the same features and huge training time still need further study.

3 Proposed malware detection model

3.1 Overview

The proposed model consists of a malware signature extraction module (MSEM) and a suspicious program detection module (SPDM). A flowchart for the MSEM is shown in Figure 1.

In the MSEM, a malware candidate signature library (MCSL) and a benign program malware-like signature library (BPMSL) are extracted, respectively, from the malware and benign programs of the training set after generating the malware instruction library (MIL). Taking the MCSL as "nonself" and the BPMSL as "self", an NSAPF is introduced to extract the malware detection signature library (MDSL) consisting of MDSL1 and MDSL2. More detailed information is given below.

In the SPDM, signatures of suspicious programs are extracted using the MIL. Then r-contiguous bit matching is computed between the signatures of the suspicious program and the MDSL. If the matching value exceeds the given program classification threshold, we classify the programs as malware; otherwise it is considered a benign program.

3.2 Malware signature extraction module

3.2.1 *MIL*

In this paper, we represent instructions as bit strings with 2 bytes (In the process of signature extraction, several instructions form one signature, so the length of the instruction does not affect the results greatly). We traverse all the programs in the training set to obtain the frequency statistical information of instruction i in both the malware and benign programs, denoted by I_n^i and I_s^i, respectively. Meanwhile, the numbers of malware and benign programs that contain instructions i are computed and denoted by F_n^i and F_s^i, respectively. Graphs of some real experimental data are shown in Figures 2 and 3, where every point denotes an instruction.

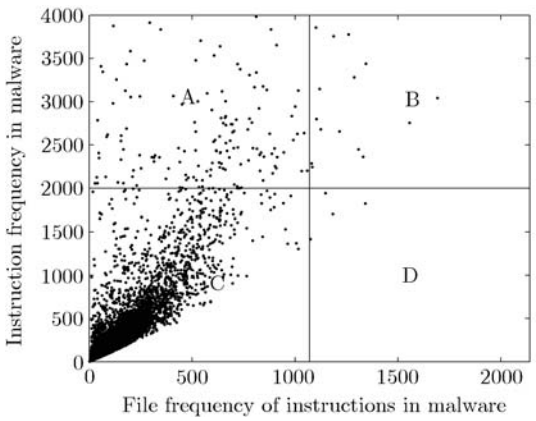

Figure 2 Instruction distribution in malware.

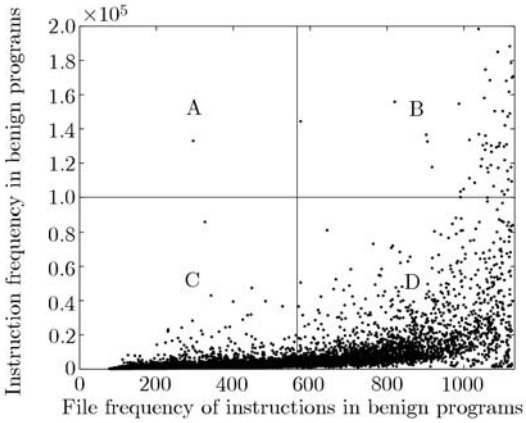

Figure 3 Instruction distribution in benign programs.

As illustrated in Figures 2 and 3, the instruction distributions in malware and benign programs differ significantly. If we wish to extract instructions with higher tendencies to malware, we have to select instructions in region B of Figure 2 and region C of Figure 3, and combine the two regions into one through eq. (1). I^i and F^i denote the tendencies of instruction i to malware on instruction frequency and file frequency, respectively. Taking I^i as the x-axis and F^i as the y-axis, the resulting graph is shown in Figure 4.

$$I^i = \frac{I_n^i/I_n}{I_n^i/I_n + I_s^i/I_s}, \quad F^i = \frac{F_n^i/F_n}{F_n^i/F_n + F_s^i/F_s}, \tag{1}$$

where I_n and I_s denote, respectively, the number of instructions in malware and benign programs in the training set, and F_n and F_s denote the number of the malware and benign programs in the training set.

This paper uses eq. (2) to take both I^i and F^i into consideration.

$$T^i = \sqrt{(I^i)^2 + (F^i)^2}. \tag{2}$$

When the tendency of instruction i to malware T^i exceeds the malware instruction threshold T_1, we consider instruction i as tending to appear in malware. All these instructions make up the MIL. In other words, all instructions excluded by the curve in Figure 4 are included in the MIL.

3.2.2 *MCSL*

In this section, a sliding window is used to split the malware bit string to obtain the MCSL. We set the window size of the sliding window to 2 bytes, and the sliding window moves forward 1 byte at a time. Each program is considered to be a bit string and is traversed by the sliding window as shown in Figure 5.

When the sliding window, in the process of traversing the bit string, encounters the first instruction that belongs to the MIL, it begins to generate a signature. If instructions in two adjacent sliding windows do not belong to the MIL, the next signature cannot be part of the current signature and thus, the current signature is terminated. The sliding window keeps moving forward, and on encountering another instruction belonging to the MIL, it begins to generate the next signature. This process is repeated until the entire bit string has been traversed.

If the number of instructions contained in a signature belonging to the MIL exceeds threshold R, the signature is deemed to contain enough dangerous information to represent malware and it is considered a malware candidate signature. Here $R = 3$. Because the length of a candidate signature auto adjusts according to the program bit string and the MIL, R does not affect greatly on the model's accuracy.

By traversing malware and benign programs in the training set, the proposed model can extract the MCSL and BPMSL.

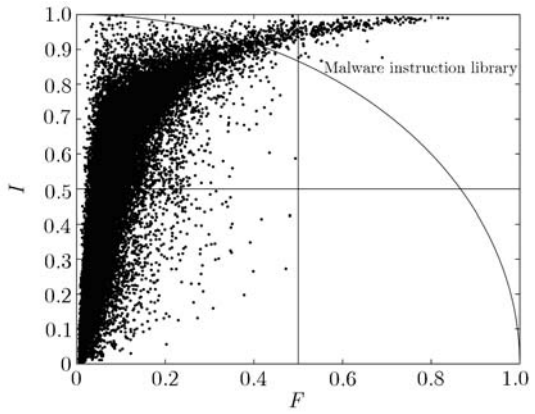

Figure 4 Instruction tendencies.

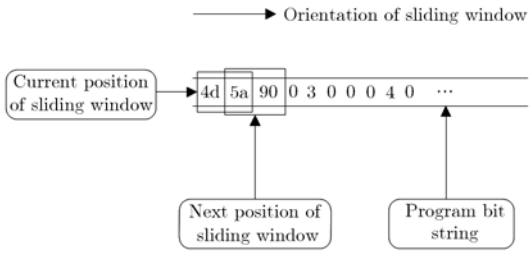

Figure 5 Schematic diagram of the sliding window traversing a program bit string.

3.2.3 *NSAPF and MDSL*

This paper considers the MCSL as "nonself" and the BPMSL as "self" and generates an MDSL using NSAPF.

The traditional NSA deletes detectors matching "self" directly and obtains a detector set in which no signatures match "self". This paper takes the view that the traditional NSA is not completely suited to malware detection. Consider formatting a disk as an example. This operation is dangerous; programs implementing this operation are considerably "dangerous". If a program implementing this operation neither reads any command line parameters nor asks the user to confirm, it could be malware. This type of dangerous signature provides some useful information. In fact, the operation of formatting a disk can be included in both malware and benign programs. Deleting such dangerous code snippets from the MCSL, as is done by the traditional NSA, destroys useful information, which is obviously a disadvantage for the malware detection model.

Theoretically, every program, regardless of whether it is a benign program or malware, can use almost any of the instructions and functions in a computer system. Moreover, almost all the functions used in malware are also used by specific benign programs, for example, formatting a disk, modifying the registry. If a "perfect" "self" set is given, the traditional NSA would be ineffective due to delete too many detectors.

The NSAPF saves "nonself" signatures that do not match "self" in the MDSL1 and "nonself" signatures matching "self" in the MDSL2. Together the MDSL1 and MDSL2 make up the MDSL.

Signatures in the MDSL1 are characteristic signatures of "nonself", whereas signatures in the MDSL2 are dangerous ones belonging to both "self" and "nonself", and which should be penalized by penalty factor C after ascertaining, through probabilistic methods, to what extent they represent malware.

3.3 Suspicious program detection module

3.3.1 *Signature matching*

It is easy to extract signatures of suspicious programs by adopting the approach used in the process of generating the MCSL. The matching value between a signature of a suspicious program and the malware detection signature is proportional to the matching length and weights of the two signatures.

Eq. (3) gives the matching value between signatures of suspicious programs and signatures in the MDSL1.

$$M_1 = \begin{cases} 0, & l < R, \\ (l - R + 1) \times w_1 \times w_2, & l \geqslant R, \end{cases} \tag{3}$$

where l is the number of matching instructions in the two signatures, R is the minimal number of matching instructions for two signatures to match each other and it has the same value as the matching length

threshold R, which is set to 3 in this paper, and w_1 and w_2 are the weights of the two signatures expressed as the number of signatures appearing in the files.

Eq. (4) gives the matching value between signatures of suspicious programs and signatures in the MDSL2.

$$M_2 = \begin{cases} 0, & l < R, \\ [(l - R + 1) \times w_1 \times w_2 \times p] \times (1 - C), & l \geqslant R, \end{cases} \tag{4}$$

where $p = w_2/(w_2 + w_3)$ (w_2 is the weight of the signature in the MDSL2 and w_3 is the weight of the signature in the BPMSL previously matching the signature before) denotes the probability of representing malware of the signature in the MDSL2, and C is the penalty factor with its interval [0,1].

As penalty factor C increases, signatures in the MDSL2 are penalized more severely and the extent to which they represent malware decreases. When $C = 1$, the NSAPF degenerates to the traditional NSA.

3.3.2 *Matching between suspicious programs and the MDSL*

The matching value between a suspicious program and the MDSL is calculated using eq. (5).

$$M = \frac{M_{MDSL1} + (1 - C) \times M_{MDSL2}}{[L_{MDSL1} + (1 - C) \times L_{MDSL2}] \times \sum w}, \tag{5}$$

where M_{MDSL1} and $(1 - C) \times M_{MDSL2}$ denote the total sum of the matching values of all the signatures in a suspicious program with signatures in the MDSL1 and MDSL2, respectively. L_{MDSL1} and $(1 - C) \times L_{MDSL2}$ are the maximal matching values provided by the MDSL1 and MDSL2, respectively. C is the penalty factor, and $\sum w$ is the sum of the weights of all signatures in a suspicious program.

If the M value of a suspicious program is greater than the program classification threshold T_2, the associated program is classified as malware, otherwise it is deemed to be a benign program.

3.3.3 *Analysis of the penalty factor*

L_{MDSL1} and L_{MDSL2} are constants for a specific training set. For a specific suspicious program, M_{MDSL1} and M_{MDSL2} are also constants. At this time, eq.(5) is a function with independent variable penalty factor C and dependent variable matching value M. If the derivation of eq. (5) is greater than 0, we obtain eq. (6).

$$L_{MDSL1}/L_{MDSL2} < M_{MDSL1}/M_{MDSL2}. \tag{6}$$

In cases where eq. (6) is correct, M is monotonically increasing together with increasing C. When $C = 1$, $(1 - C) \times M_{MDSL2}$ and $(1 - C) \times L_{MDSL2}$ both are 0. The NSAPF is now equivalent to the traditional NSA. Let $C = 1$ be the comparable benchmark.

When $C = 1$, recognizable malware usually has a greater M_{MDSL1} and benign programs have a smaller M_{MDSL1}. By decreasing C, the M values of malware and benign programs would tend to decrease and increase, respectively, making it difficult to recognize such programs. When $C = 1$, unrecognizable malware and benign programs have a smaller M_{MDSL1} and greater M_{MDSL1}, respectively. By decreasing C, the M values of malware and benign programs tend to increase and decrease, respectively. Thus, decreasing C is beneficial for classifying such programs.

An optimal penalty factor C is obtained on the training set and this is helpful to improve the model's performance. Furthermore, two constraints are necessary for penalty factor C to play a positive role: (1) MDSL2 contains enough signatures; (2) the percent of signatures in the MDSL2 relative to the MDSL must be large enough. If and only if these two constraints are satisfied, the MDSL2 can change the matching values of suspicious programs with the MDSL and recognize unknown malware.

4 Experiments and analysis

4.1 Experimental datasets

Experiments in this paper were conducted using the three datasets, which could be downloaded from the following website: http://www.cil.pku.edu.cn/resources/.

Table 1 Henchiri dataset

	Filetype	Qty.	Avg. Size	Min. Size	Max. Size
B^*	EXE	1414	107	16	501
M^*	Virus	2880	6.2	22	93.4
	Trojan	88	9.4	49	72.5
	Constructor	6	10	528	33.6
	Other	20	11.6	456	88.5

Table 2 CILPKU08 dataset

	Filetype	Qty.	Avg. Size	Min. Size	Max. Size
B^*	EXE	915	138.5	817	997
M^*	Virus	3465	4.8	23	59.5
	Trojan	39	4.4	49	5.93
	Other	43	6.8	48	31.2

Table 3 Benign programs in VX Heavens dataset

Filetype	Qty.	Avg. Size	Min. Size	Max. Size
DOC	300	103	11000	1587
EXE	300	82.7	6000	498
JPG	300	43.7	447	416
MP3	300	61.5	735	6586
PDF	300	113.3	46	16657
ZIP	300	100	546	2941

4.1.1 *Henchiri dataset*

The Henchiri dataset consists of 2994 malware and 1414 benign programs. The malware were provided by Henchiri and Japkowicz [2], while the benign programs consist of executable (EXE) system files from Windows XP and EXE files from a series of applications. These files cover EXE files of mainstream operating systems and applications, and are the main targets for attack by malware. Detailed information on the Henchiri dataset is given in Table 1.

Here M^* and B^* indicate malware and benign programs, respectively. The units of both the average size and the maximum size is KB, while the unit of minimum size is byte (these units are also applicable to other tables in this paper).

4.1.2 *CILPKU08 dataset*

We previously used the CILPKU08 dataset in [15]. The process of collecting benign programs is the same as in section 4.1.1. The details of this dataset are presented in Table 2.

4.1.3 *VX Heavens dataset*

The benign programs in the VX Heavens dataset are general programs collected from Windows XP and are listed in Table 3. Malware for this dataset comes from the "VX Heavens Virus Collection" (http://vx.netlux.org). The paper only considers malware based on the PE format of Win32. The VX Heavens dataset used in this paper contains 7128 malware, details of which are given in Table 4. Here "Others" includes malware such as DoS, Nuker, Exploit, Hacktool, and Flooder.

4.2 Experiments on the Henchiri dataset

Here we adopt 5-fold cross validation to estimate the performance of the proposed model as accurately as possible.

Table 4 Malware in VX Heavens dataset

Filetype	Qty.	Avg. Size	Min. Size	Max. Size
Backdoor	2200	48	3500	9227
Constructor	172	392.9	5060	2391
Trojan	2350	147.7	215	3800
Virus	1048	71.1	1500	1278
Worm	351	199.3	394	11899
Others	1007	151.4	1090	3087

4.2.1 *Cross validation*

According to the malware's name, 2994 malware is divided into 880 families. Based on family, the malware is divided into 5 fold referred to as M^i $(i = 1, 2, \ldots, 5)$. 1414 benign programs are divided into 5 fold in a similar manner and are referred to as B^i $(i = 1, 2, \ldots, 5)$. The detailed process of cross validation is shown in Algorithm 1.

Algorithm 1 Cross validation algorithm
$M = \bigcup_{i=1}^{5} M^i, B = \bigcup_{i=1}^{5} B^i$
for $i = 1$ to 5 **do**
  $DS = M^i \cup B^i, TS = (M - M^i) \cup (B - B^i)$;
  Train model on TS;
  Trained model detects suspicious programs in TS and DS;
end for
$FP\_TS = BW\_TS/(4 \times \|B\|), TP\_TS = MC\_TS/(4 \times \|M\|)$
$FP\_DS = BW\_DS/\|B\|, TP\_DS = MC\_DS/\|M\|$

In Algorithm 1, TS and DS denote the training set and test set; BW_TS and MC_TS denote the number of misclassified benign programs and the number of correctly recognized malware in the TS; BW_DS and MC_DS are the number of misclassified benign programs and the number of correctly recognized malware in the DS; FP_TS and TP_TS denote the false positive and true positive rates in the TS, while FP_DS and TP_DS denote the false positive and true positive rates in the DS.

Results for the training and test sets are given in Tables 5 and 6. Experimental results of Henchiri and Japkowicz [2] are shown in Table 7 for comparison.

Experimental results on the training set show that when penalty factor C lies within [0.90, 0.99], the proposed model achieves good detection accuracies with lower false positive rates (FPRs), less than 3%, and higher true positive rates (TPRs), above 95.9%. The proposed model also obtains very good results on the test set with an FPR below 4%, which is lower than the FPR obtained by Henchiri and Japkowicz, and the average TPR at 95%. The optimal overall accuracy (OA) on the test set is 96.1%, higher than that achieved by Henchiri and Japkowicz (93.65%) [2].

It is easy to ascertain from Tables 5 and 6 that with a decrease in penalty factor C, the penalty to signatures in MDSL2 decreases. The MDSL2 provides more and more false information while still providing correct information. As a result, the FPR of the proposed model increases, while the TPR decreases. The OA increases at first and finally drops. With $C = 0.90$, the OA of the proposed model is at its maximum, 96.1%. The results demonstrate that the MDSL2 plays a positive role and improves the effectiveness of the proposed model.

The performance of the proposed model on the test set is similar to its performance on the training set, proving that the model has a good training function.

4.2.2 *Negative direction cross validation*

In this section, we take the training set used in the cross validation as the test set and vice versa. Negative direction cross validation is done to verify the model's generalization ability. Experimental results is shown in Table 8.

Table 5 Experimental results on the training set

C	OA(%)	FPR(%)	TPR(%)
0.00	95.5	7.3	96.8
0.50	95.9	5.9	96.8
0.90	96.7	2.9	96.6
0.95	96.7	1.7	95.9
0.99	96.6	0.8	95.3
1.00	95.6	0.0	93.6

Table 6 Experimental results on the test set

C	OA(%)	FPR(%)	TP(%)
0.00	95.0	7.4	96.2
0.50	95.5	5.9	96.2
0.90	96.1	3.3	95.8
0.95	95.8	2.5	95.0
0.99	95.4	2.5	94.4
1.00	94.2	1.3	92.0

Table 7 Experimental results of Henchiri and Japkowicz

Classifier	OA(%)	FPR(%)	TPR(%)
ID3	93.29	4.16	90.56
J48	93.65	5.24	92.56
Naïve Bayes	69.51	0.13	37.17
SMO	93.39	5.71	92.26

Table 8 Experimental results on the test set

C	OA(%)	FPR(%)	TPR(%)
0.00	94.3	6.8	94.8
0.50	94.5	6.5	94.9
0.90	95.3	4.5	95.2
0.95	95.2	3.8	94.7
0.99	94.7	3.2	93.6
1.00	94.2	3.0	92.8

From Table 8, we can see that the OA still remains above 94% on the test set. The proposed model trained with a small training set also achieves good performance on a larger test set, showing that this model has strong generalization ability. With $C = 0.90$, the OA on the test set achieves its maximum value: 95.3%.

4.3 Experiments on the CILPKU08 dataset

Comparable experiments were done on the CILPKU08 dataset we used in [15]. In this study, we selected six experiments from our previous study for comparison. The proposed model is referred to as the MDM-NSAPF, while the approach in [15] is referred to as HAIM. The penalty factor is set to 1 and graphs of the experimental results are shown in Figures 6 and 7.

In Figure 7, $R = Time_1/Time_2$, $Time_1$ is the training time used by MDM-NSAPF and $Time_2$ is HAIM's training time.

As illustrated in Figures 6 and 7, the OA of the proposed model is about 1%–3% higher than that of HAIM; yet the training time of the proposed model is only 1/10 of that of HAIM.

4.4 Experiments on the VX Heavens dataset

The specific training set for "virus" consists of 50 benign programs and 50 "viruses" which were randomly selected from the VX Heavens dataset. The remaining benign programs and "viruses" make up the specific test set for "virus". We divided the worm, trojan, backdoor, constructor and other malware in a similar way. Six datasets were obtained and used to train and verify the proposed model.

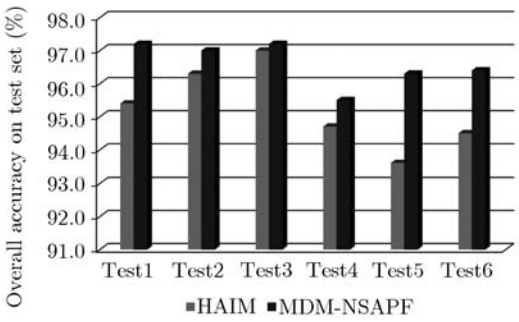

Figure 6 Overall accuracy comparison.

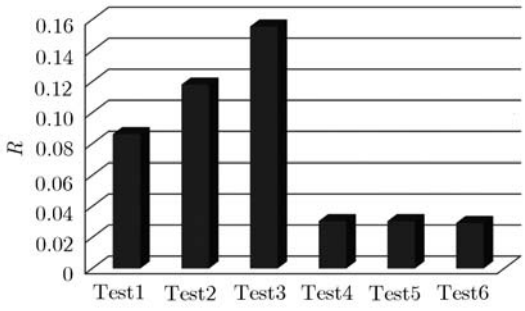

Figure 7 Training time comparison.

Table 9 Experimental results on the VX Heavens dataset

C	Virus	Worm	Trojan	Backdoor	Constructor	Others
0.00	0.918	0.965	0.91	0.739	0.96	0.867
0.50	0.986	0.962	0.908	0.739	0.96	0.912
0.90	**0.987**	0.961	**0.913**	0.923	**0.963**	**0.917**
0.95	0.986	**0.962**	0.911	0.923	0.963	0.917
0.99	0.98	0.961	0.911	**0.924**	0.963	0.907
1.00	0.931	0.939	0.852	0.918	0.922	0.867

Table 10 Experimental results obtained by Tabish

Virus	Worm	Trojan	Backdoor	Constructor	Others
0.945	0.919	0.881	0.849	0.925	0.903

To compare the resuluts obtained by Tabish et al. [3], the area under the receiver operating characteristic curve (AUC) was set as the measure of the effectiveness of the proposed model. Table 9 gives the experimental results in detail with the bold font in each column indicating the optimal AUC in the corresponding models.

Compared with the results of Tabish et al. [3] shown in Table 10, the optimal AUC of the proposed model is on average 0.04 higher. This is because the proposed model generates the MDSL using the NSAPF, which decreases the FPR and achieves a better tradeoff between FPR and TPR by adjusting penalty factor C.

4.5 Parameter analysis

Experimental results demonstrate that when the number of signatures in the MDSL2 is greater than 2000 and the percentage of these signatures contributing to the MDSL is greater than 30%, penalty factor C plays a positive role improving the model's performance significantly. This paper suggests that penalty factor C should be set to a value in the interval $[0.9, 0.99]$.

The malware instruction threshold T_1 usually obtains its value in the interval $[0.9, 1]$, while the malware classification threshold T_2 should be set to a value in the interval $[0.00001, 0.0001]$.

Figure 8 shows the detection accuracy of the proposed model for different values of T_1. With $T_1 = 0.95$, the model achieves optimal detection accuracy. When T_1 is small, malware instructions with lower tendencies increase the FPR. When T_1 is large, there are too few malware instructions to provide enough malware detection signatures to cover the space of malware detection signatures. In summary, an appropriate T_1 ensures that the proposed model contains enough malware instructions with marked tendencies, thus generating an optimized MDSL to help the model achieve better performance.

Figure 9 shows the effect of the malware classification threshold T_2 on the detection accuracy of the proposed model. By increasing T_2, the TPR of the proposed model decreases monotonically, while the FPR increases monotonically. With $C = 0.00005$, the proposed model achieves optimal overall accuracy.

5 Conclusions

The malware detection model based on the NSAPF overcomes the drawback of the traditional NSA in

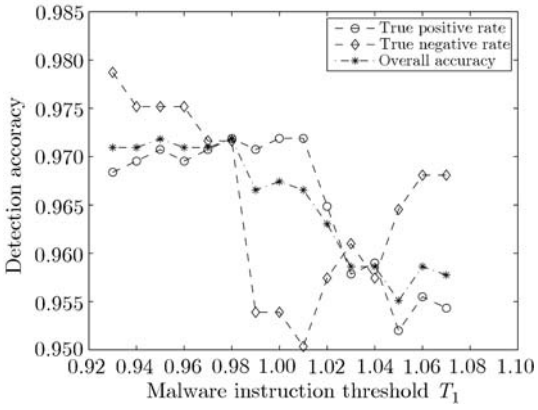

Figure 8 Detection accuracy with different T_1.

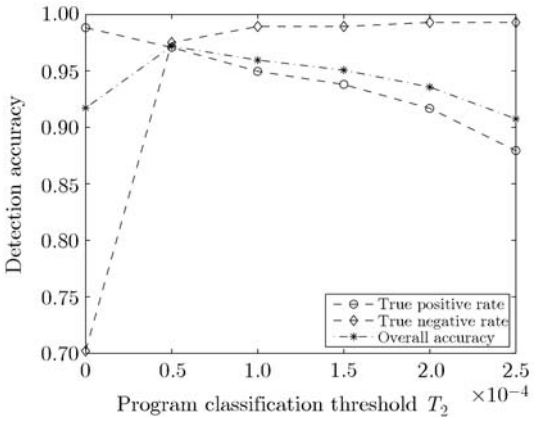

Figure 9 Detection accuracy with different T_2.

defining harmfulness of "self" and "nonself". It focuses on the harmfulness of the code and extracts dangerous signatures, which are included in the MDSL. By adjusting the penalty factor C, the model achieves a tradeoff between the TPR and FPR to satisfy the requirements of various users in terms of TPR and FPR. Comprehensive experimental results demonstrate that the proposed model is effective in detecting unknown malware with a lower FPR.

Further work includes making use of the relativity of signatures, and integrating other heuristic intelligent algorithms.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grant Nos. 60673020, 60875080), and partially supported by the National High-Tech Research & Development Program of China (Grant No. 2007AA01Z453).

References

1 Forrest S, Perelson A S, Allen L, et al. Self-nonself discrimination in a computer. In: 1994 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, 1994. 202–212
2 Henchiri O, Japkowicz N. A feature selection and evaluation scheme for computer virus detection. In: Sixth International Conference on Data Mining, HongKong, 2006. 891–895
3 Tabish S M, Shafiq M Z, Farooq M. Malware detection using statistical analysis of byte-level file content. In: CSI-KDD'09, Paris, 2009. 23–31
4 Ye Y F, Jiang Q S, Zhuang W W. Associative classification and post-processing techniques used for malware detection. In: 2nd International Conference on Anti-counterfeiting, Security and Identification, Gaiyang 2008. 276–279
5 Ye Y F, Wang D D, Li T, et al. IMDS: Intelligent malware detection system. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, 2007. 1043–1047
6 Deng P S H, Wang J H, Shieh W G, et al. Intelligent automatic malicious code signatures extraction. In: Proceedings of IEEE 37th Annual 2003 International Carnahan Conference on Security Technology, Washington, 2003. 600–603
7 Karnik A, Goswami S, Guha P. Detecting obfuscated viruses using cosine similarity analysis. In: Proceedings of the First Asia International Conference on Modeling & Simulation, Phuket, 2007. 165–170
8 Forrest S, Hofmeyr S A, Somayaji A, et al. A sense of self for Unix processes. In: Proceedings of 1996 IEEE Symposium on Security and Privacy, Oakland, 1996. 120–128
9 Kim J, Bentley P. Towards an artificial immune system for network intrusion detection: An investigation of clonal selection with a negative selection operator. In: Congress on Evolutionary Computation, Seoul, 2001. 1244–1252
10 Lee H, Kim W, Hong M. Artificial immune system against viral attack. In: ICCS 2004, Kraków 2004. 499–506
11 Edge K S, Lamont G B, Raines R A. A retrovirus inspired algorithm for virus detection & optimization. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, Seattle, 2006. 103–110
12 Li Z, Liang Y W, Wu Z J, et al. Immunity based virus detection with process call arguments and user feedback. In: Bio-Inspired Models of Network, Information and Computing Systems, Budapest, 2007. 57–64
13 Balachandran S, Dasgupta D, Nino F, et al. A general framework for evolving multi-shaped detectors in negative selection. In: Proceedings of IEEE Symposium Series on Computational Intelligence, Honloulu, 2007. 401–408
14 Li T. Dynamic detection for computer virus based on immune system. Sci China Ser F-Inf Sci, 2008, 51: 1475–1486
15 Wang W, Zhang P T, Tan Y, et al. A hierarchical artificial immune model for virus detection. In: International Conference on Computational Intelligence and Security, Beijing, 2009. 1–5