Hindawi
Security and Communication Networks
Volume 2018, Article ID 2063089, 10 pages
https://doi.org/10.1155/2018/2063089

Research Article

A Novel Immune-Inspired Shellcode Detection Algorithm Based on Hyperellipsoid Detectors

Tianliang Lu,[1,2] **Lu Zhang**,[1] **and Yixian Fu**[1]

[1]*Institute of Information Technology and Network Security, People's Public Security University of China, Beijing, China*
[2]*Collaborative Innovation Center of Security and Law for Cyberspace, Beijing, China*

Correspondence should be addressed to Tianliang Lu; ltl135@126.com

Received 21 October 2017; Accepted 31 January 2018; Published 28 February 2018

Academic Editor: Paolo D'Arco

Copyright © 2018 Tianliang Lu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Shellcodes are machine language codes injected into target programs in the form of network packets or malformed files. Shellcodes can trigger buffer overflow vulnerability and execute malicious instructions. Signature matching technology used by antivirus software or intrusion detection system has low detection rate for unknown or polymorphic shellcodes; to solve such problem, an immune-inspired shellcode detection algorithm was proposed, named ISDA. Static analysis and dynamic analysis were both applied. The shellcodes were disassembled to assembly instructions during static analysis and, for dynamic analysis, the API function sequences of shellcodes were obtained by simulation execution to get the behavioral features of polymorphic shellcodes. The extracted features of shellcodes were encoded to antigens based on *n*-gram model. Immature detectors become mature after immune tolerance based on negative selection algorithm. To improve nonself space coverage rate, the immune detectors were encoded to hyperellipsoids. To generate better antibody offspring, the detectors were optimized through clonal selection algorithm with genetic mutation. Finally, shellcode samples were collected and tested, and result shows that the proposed method has higher detection accuracy for both nonencoded and polymorphic shellcodes.

1. Introduction

Shellcode is a small piece of executable binary code written by the attackers. To exploit the buffer overflow vulnerability, shellcode is injected to target programs in the form of network traffic or malformed files, overwriting the program return address to gain executable permission. Usually shellcode opens a shell that can be connected by the attackers and execute commands, so-called shellcode. Most well known worms (e.g., Blaster, Stuxnet, CodeRed, and a newly emerging ransomware WannaCry) have been propagated by means of shellcode injection.

Writing a shellcode has high level technology requirement for attackers who need to be proficient in assembly instructions, function call, and process memory stack structure. But with the advent of penetration testing tool, such as Metasploit [1], shellcode can be conveniently customized based on user needs and can be encoded to avoid illegal characters or signature detection. To defeat signature-based detection, polymorphic shellcode that contains a decryption routine and encrypted payload has been developed. During the execution, the decryption routine will decrypt the payload into original shellcode. The original shellcode can be encoded to many polymorphic instances with different encryption algorithms, so it will bring great difficulty to signature-based detection technology.

To detect shellcodes, especially recognizing polymorphic instances accurately, a shellcode detection algorithm based on artificial immune system is proposed. The main contributions of this paper are as follows:

(i) Inspired by biological immune system to eliminate bacteria and viruses, the immune theory is applied to solve the shellcode detection problem.

(ii) In order to detect polymorphic shellcode, both static disassembly analysis and dynamic simulation analysis are applied, and assembly instruction sequence and API function sequence are obtained.

(iii) Hyperellipsoid detectors encoding method is adopted to improve the nonself space coverage rate, and the performance of detectors is optimized through a genetic mutation.

The rest of this paper is organized as follows. Section 2 gives an overview of the related previous work. Section 3 describes the static and dynamic features extraction of shellcode. Section 4 explains the proposed artificial immune based shellcode detection algorithm in detail. Section 5 evaluates our approach through experiments. Section 6 concludes the paper.

2. Related Work

2.1. Shellcode Detection. Shellcode detection is an efficient way to prevent code-injection based vulnerability attacks. In recent years, both static and dynamic methods have been proposed to identify shellcodes.

A traditional approach for shellcodes static detection is signature matching, where the signature is a set of strings or regular expressions that are extracted from the acquired shellcode samples. As the most effective way to defeat known shellcodes, signature-based method is employed in firewalls, antivirus software, and other security products. But it is less useful against new samples and can be easily bypassed by encoding techniques, such as polymorphism and metamorphism. Alphanumeric shellcode automatic generation methods [2] were proposed to bypass character restrictions and antishellcode engines, and encoding has achieved more than 20% size reduction for many shellcodes. Verma et al. [3] present a novel approach to detect alphanumeric shellcodes based on string similarity. Zhao and Ahn [4] propose a novel feature extraction method called instruction sequence abstraction and facilitate a Markov-chain-based model for shellcode detection.

The static detection method could not effectively handle polymorphic shellcodes. To solve this problem, the dynamic detection method based on emulation is proposed. The behavioral features (e.g., API calls and memory reads and writes) are obtained and can be used to detect shellcodes. Libemu [5] is a library that can be used for x86 emulation and shellcode detection. Embedded in IDS or Honeypot systems, Libemu can detect and execute shellcodes by using the GetPC heuristics. Polychronakis et al. [6] present a heuristic detection method for the presence of previously unknown polymorphic shellcodes. Their approach relies on a NIDS-embedded CPU emulator that executes every potential instruction sequence in the inspected traffic, aiming to identify the execution behaviors of polymorphic shellcodes. Luo et al. [7] improve the GetPC location mechanism and IA-32 instruction recognition method. Subsequently implement a dual-mode virtual machine to detect polymorphic shellcodes by transferring states of the finite automaton between control-flow model and data-flow model. Gu et al. [8] propose a new shellcode detection methodology in which snapshots of the process's virtual memory are taken before input data are consumed, and the snapshots can also be used to examine the system calls that shellcodes invoke.

In conclusion, static detection method has good detection performance for nonencoded shellcodes but cannot effectively detect polymorphic or metamorphic shellcodes. Dynamic analysis can spot polymorphic shellcodes' malicious behaviors accurately by emulation, but it also has some drawbacks. Since the starting position of the shellcodes in network flow is unknown, the computation complexity for positioning execution entrance through heuristic GetPC or searching NOPSled is very high. Dynamic simulation of shellcodes with hard encoding will fail because of lack of execution context. Specific simulation environment can be bypassed through the jump over hook technology.

2.2. Artificial Immune System. The biological immune system is a complex system consisting of organs, cells, and molecules. It can identify the self, exclude the nonself (or antigen, e.g., virus, bacteria), for maintaining security and stability in the biological environment. The immune system has drawn significant attention as a potential source of inspiration for novel approaches to solve complex computational problems. Its highly distributed, adaptive, and self-organizing nature, together with its learning, memory, feature extraction, and pattern recognition features offer rich metaphors for its artificial counterpart [9]. AIS has been applied in many research areas; because of the natural similarity of responding to nonself, AIS can solve many computer security problems [10, 11], such as malware detection [12], intrusion detection [13], and spam detection [14]. Four major AIS algorithms have been constantly developed and gained popularity: (1) negative selection algorithm (NSA); (2) artificial immune network; (3) clonal selection algorithm; and (4) the danger theory and dendritic cell algorithm. The negative selection algorithm and clonal selection algorithm are introduced in our paper for generating and optimizing the shellcode detectors.

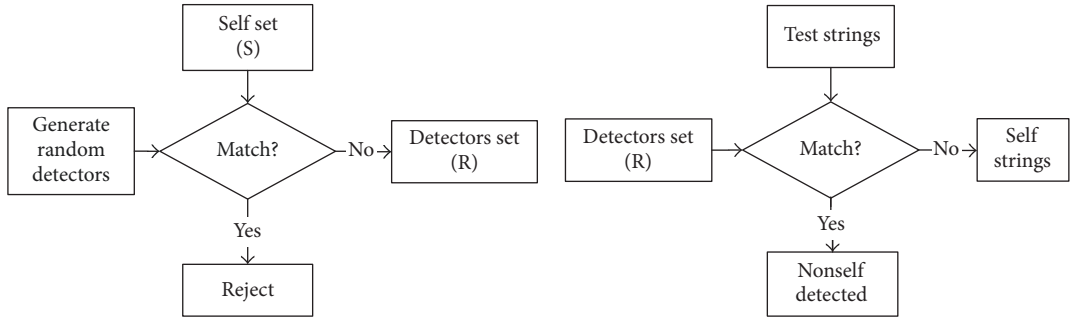
2.2.1. Negative Selection Algorithm. Discrimination between self and nonself is one of the major mechanisms in the immune system. Negative selection algorithm is a computational imitation of self/nonself discrimination. It is inspired by the T-cell maturing process that happens in the thymus. T-cells of enormous diversity are first assembled with a genetic rearrangement process and those that recognize self cells are eliminated, and the rest are deployed into the immune system to recognize outside pathogens. The negative selection algorithm runs as follows [15].

The algorithm has two phases:

(i) Generate detectors set, shown in Figure 1(a). Each detector is a random generated string that does not match any of the self set.

(ii) Nonself detection: suspicious strings are tested, and if a string matches any of the detectors then it is identified as nonself. If no detector matches the string, then it is considered self string, as shown in Figure 1(b).

In the original version of NSA, elements of the shape space and detectors were represented using binary strings,



(a) Generation detectors set (b) Nonself detection

Figure 1: Negative selection algorithm.

and the matching rules include r-contiguous bit, r-chunks, and Hamming distance. Gonzalez and Dasgupta [16] introduce a real-valued representation, called real-valued negative selection (RNS). In RNS, a detector (antibody) is defined by an n-dimensional vector that corresponds to the center and by a real value that represents its radius. Ji and Dasgupta [17] improve the hypersphere representation by using variable radius detectors called V-detectors, which allows the model to cover more nonself space with fewer detectors at no cost in computational complexity bounds. Shapiro et al. [18] introduce hyperellipsoids which are much more flexible, mostly due to they can be stretched and reoriented. Fewer hyperellipsoids than hyperspheres are needed to achieve similar coverage of nonself space.

2.2.2. Clonal Selection Algorithm. The clone selection theory is used to explain the basic response of the adaptive immune system to an antigenic stimulus. The B cells are capable of recognizing that an antigenic stimulus will become activated and begin to proliferate. The clones then undergo somatic hypermutation and produce antibodies that are specific to the invading antigen. After proliferation, B cells differentiate into plasma cells or long-lived B memory cells. The memory cells ensure that both the speed and accuracy of the immune response become successively stronger after each infection.

Inspired by the clonal selection process, de Castro and von Zuben [19] propose an optimization algorithm, known as CLONALG, which exploits the cloning, mutation, and selection mechanisms of clonal selection. The clonal selection algorithm can solve many engineering problems, such as pattern recognition, multimodal optimization, and travelling salesperson problems (TSP). In our paper, the clonal selection algorithm is introduced to optimize shellcode detectors, and detector offspring with better recognition ability is generated after the clone and mutation process.

3. Shellcode Feature Extraction

The process of static and dynamic features extraction for shellcode is shown in Figure 2.

Extracting the features of collected samples (both shellcode and benign samples) includes assembly instruction sequences based on the static disassemble technology and API function sequences based on the dynamic simulation extraction technology. These two kinds of sequences are then split to sequences segments based on the n-gram model, forming the feature set. The feature vector of specific sample is calculated by TF-IDF (term frequency-inverse document frequency), a numerical statistic that is intended to reflect how important a word is to a document in a collection.

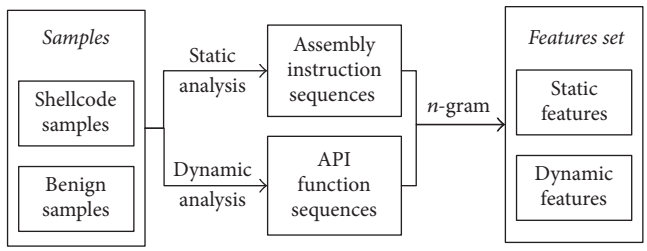


Figure 2: Static and dynamic features extraction.

```
6a08    push 0x8
59      pop ecx
50      push eax
e2fd    loop 0x000000ae
40      inc eax
50      push eax
```

Figure 3: The disassembly instructions of a shellcode.

3.1. Static Analysis Technology. The disassembly technology is widely used in many areas such as virus analysis and software decryption. The assembly instructions contain key information about shellcode malicious operations. To convert the binary shellcodes to assembly instructions based on the disassembly technology, there exist two basic disassembly algorithms: linear sweep disassembly and recursive descent disassembly. IDA Pro, W32Dasm, and C32Asm are the most widely used disassembly software. Figure 3 shows the disassembly result of a selected shellcode sample, and the assembly mnemonic sequence is as follows: push, pop, push, loop, inc, and push.

```
40104b  LoadLibraryA(urlmon)
40107a  GetTempPathA(len = 104, buf = 12fce4)
4010b2  URLDownloadToFileA(http://blahblah.com/evil.exe0,
                C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\dEbW.exe)
4010bd  WinExec(C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\dEbW.exe)
4010cb  ExitProcess(1146894915)
```

Figure 4: The API function call of shellcode.

3.2. Dynamic Analysis Technology. The dynamic analysis technology is the most effective way to detect polymorphic shellcodes. The polymorphic or metamorphic shellcodes will eventually call system API functions to accomplish specific operations when they are executed. The main operations accomplished by shellcodes include opening a shell, downloading and executing files, and connecting to the remote C&C (command and control) server. Using the hook technology, the executing process of shellcodes when calling API functions can be monitored, and the function name, input parameters and return values, and other information can be obtained. Spector and scdbg [20] are the typical shellcode analysis tools for monitoring API function call. Scdbg hooks up more than 200 API functions, through simulation execution to detect shellcode behaviors, including network operations, file operations, thread operations and dynamic library loading, and other sensitive behaviors.

Figure 4 shows a shellcode's API function sequence including input parameters that are obtained through simulation execution using scdbg. It is obvious that the purpose of the above shellcode is to download an executable file from the specific URL and save it to the temporary folder; then execute the program named "dEbW.exe." The API function sequence of the shellcode is as follows: LoadLibraryA, GetTempPathA, URLDownloadToFileA, WinExec, and ExitProcess.

3.3. Feature Extraction Based on n-Gram Model. An n-gram is a contiguous sequence of n items from a given sequence of text. Widely used in statistical natural language processing, the n-gram model is used to extract features from both assembly mnemonic sequences and API function sequences. In our paper, the assembly mnemonics and API function names are treated as items or words of n-gram. The training corpus is composed of assembly mnemonic sequences and API function sequences extracted from large amount of shellcode and benign samples through static and dynamic analysis.

We extract the static n-gram features from shellcode disassembly instructions, which represent the local semantics of assembly mnemonic sequences. Let $n = 3$, and the static n-gram features extracted from the shellcode in Figure 3 are as follows: (push pop push), (pop push loop), (push loop inc), and (loop inc push), as shown in Figure 5.

Similarly, we extract the dynamic n-gram features from shellcode API function sequences, which represent the local semantics of the ordered operation when executing the shellcodes. Let $n = 3$, and the dynamic n-gram features extracted from the shellcode in Figure 4 are as follows: (LoadLibraryA GetTempPathA URLDownloadToFileA), (GetTempPathA URLDownloadToFileA WinExec), and (URLDownloadToFileA WinExec ExitProcess).

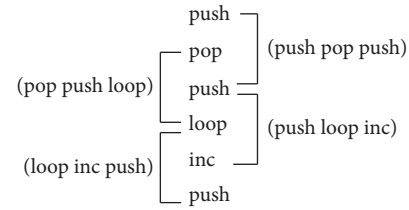


Figure 5: Static n-gram features extraction.

3.4. Feature Selection Based on Information Gain. As a feature selection method, information gain (IG) means how much information a feature adds to the classification system.

Let x be the static or dynamic feature, $y_i \in y$ be one of the k class sample labels (i.e., $k = 2$, shellcode or benign), and $\mathrm{IG}(x)$ be the IG feature weight for feature x.

$$\begin{aligned}\mathrm{IG}(x) &= H(y) - H(y \mid x) \\ &= -\sum_{i=1}^{k} p(y_i) \log(p(y_i)) \\ &\quad + p(x) \sum_{i=1}^{k} p(y_i \mid x) \log(p(y_i \mid x)) \\ &\quad + p(\overline{x}) \sum_{i=1}^{k} p(y_i \mid \overline{x}) \log(p(y_i \mid \overline{x})),\end{aligned} \tag{1}$$

where $H(y)$ is the information entropy of a sample and $H(y \mid x)$ is the conditional entropy of feature x in the sample, which represents the information quantity (i.e., x exists or does not exist in the classification system). Compute the IG value for every feature, and the larger the IG value is, the more information the feature contributes to classify shellcodes and benign samples. The feature set $F = \{f_1, f_2, \ldots, f_N\}$ is composed of the top N features based on IG values.

3.5. Feature Vector Encoding. Based on real value encoding method, each sample will be encoded to a feature vector $v = (v_1, v_2, \ldots, v_N)$, where v_i represents the weight of feature f_i in feature set F. v_i is calculated by TF-IDF. According to the concept of TF-IDF, if the frequency of feature f_i appearing in a shellcode is high but the frequency appearing in the whole sample collection is low, that means the feature f_i can identify shellcode more accurately and should be given higher weight.

Let $\mathrm{TF}_{i,j}$ be the frequency of feature f_i appearing in sample s_j and defined as follows:

$$\mathrm{TF}_{i,j} = \frac{N_{i,j}}{\sum_{k \in K} N_{k,j}}, \tag{2}$$

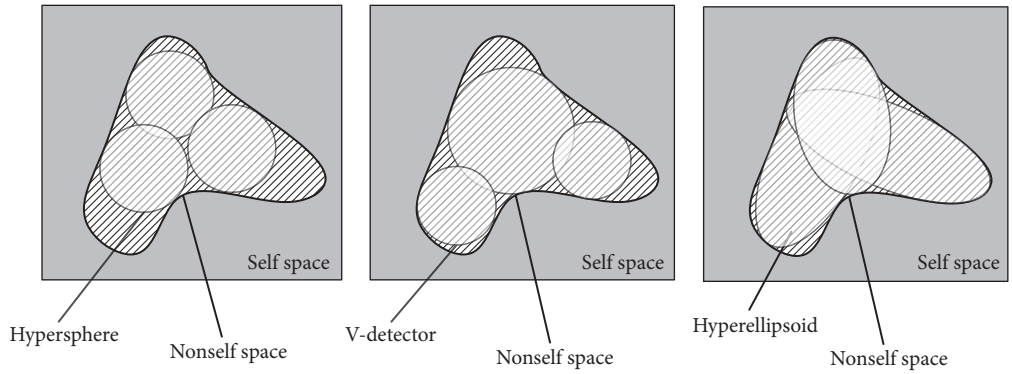


Figure 6: Comparison of nonself space coverage rate.

where $N_{i,j}$ is the number of times that feature f_i appeared in sample s_j, K is feature collections extracted from sample s_j, and k is one element of K.

The IDF of feature f_i is defined as follows:

$$\mathrm{IDF}_i = \log \frac{|D|}{1 + |\{s \in D : f_i \in s\}|}, \tag{3}$$

where $|D|$ is the total number of samples and $|\{s \in D : f_i \in s\}|$ is number of samples where the feature f_i appears.

The TFIDF of feature f_i for sample s_j is defined as follows:

$$\mathrm{TFIDF}_i = \mathrm{TF}_{i,j} \times \mathrm{IDF}_i. \tag{4}$$

Let the ith dimension v_i of feature vector $v = (v_1, v_2, \ldots, v_N)$ equal TFIDF_i. The feature vector v will be used during the process of immune detectors generation and shellcodes identification.

4. Shellcode Detection Based on Immune Theory

4.1. Definition of Hyperellipsoid. There are various geometric shapes for detectors with real-valued representation, such as hyperrectangles, hyperspheres, and hyperellipsoids.

The nonself space coverage of hyperspheres with constant radius, hyperspheres with variable radius (V-detectors), and hyperellipsoids are compared. As shown in Figure 6, the rectangle represents the whole space, the space with gray color represents self space, and the space with slash represents nonself space. It is intuitive to see that, with the same number of detectors (e.g., three), the hyperellipsoids have best coverage effect and the V-detectors come the second. Shapiro et al. [18] point out that, compared with the hyperspheres, only a half of hyperellipsoids are needed to achieve the same coverage effect. In our paper, shellcode detectors are represented using hyperellipsoids, in order to improve the shellcode detection rate.

An n-dimensional ellipsoid is defined as follows:

$$(x - \omega)^T \mathrm{A} (x - \omega) = 1, \tag{5}$$

where A is a real symmetric positive-definite $n \times n$ matrix and ω, an $n \times 1$ matrix, is the center of the ellipsoid.

An n-dimensional ellipsoid is simply a sphere that has been stretched along the n orthogonal semiaxis of the ellipsoid. The volume of an n-dimensional ellipsoid is defined as follows:

$$V = \Omega_n l_1 l_2 \cdots l_n, \tag{6}$$

where Ω_n is the volume of an n-dimensional hypersphere and $l_1 l_2 \cdots l_n$ are the lengths of the n semiaxis of the ellipsoid. The volume Ω_n of an n-dimensional unit hypersphere is calculable as

$$\Omega_n = \frac{\pi^{n/2}}{\Gamma(1 + (1/2)\, n)}. \tag{7}$$

Since A is positive definite, it can be decomposed into the form as follows:

$$\mathrm{A} = V \Lambda V^T, \tag{8}$$

where V is $n \times n$ matrix whose columns are orthonormal eigenvectors of A and Λ is a diagonal matrix whose diagonal entries are the eigenvalues associated with the eigenvectors in V. Substituting for A in (5) results in

$$(x - \omega)^T V \Lambda V^T (x - \omega) = 1. \tag{9}$$

Letting $\Lambda_{i,i}\ (1 \le i \le n)$ be the diagonals of Λ, then the length of the ith semiaxis is defined by

$$l_i = \frac{1}{\sqrt{\Lambda_{i,i}}}. \tag{10}$$

Let p be an n-d point, which represents the feature vector of a shellcode sample, and let d be an n-d ellipsoid detector as defined in (5). The point p is inside of d if and only if the inequality in (11) holds, which means the tested sample is classified as shellcode by detector d.

$$(p - \omega)^T \mathrm{A} (p - \omega) < 1. \tag{11}$$

4.2. Shellcode Detection Model. The shellcode detection model based on immune theory has two phases, as shown in Figure 7: detectors generation and shellcode detection.

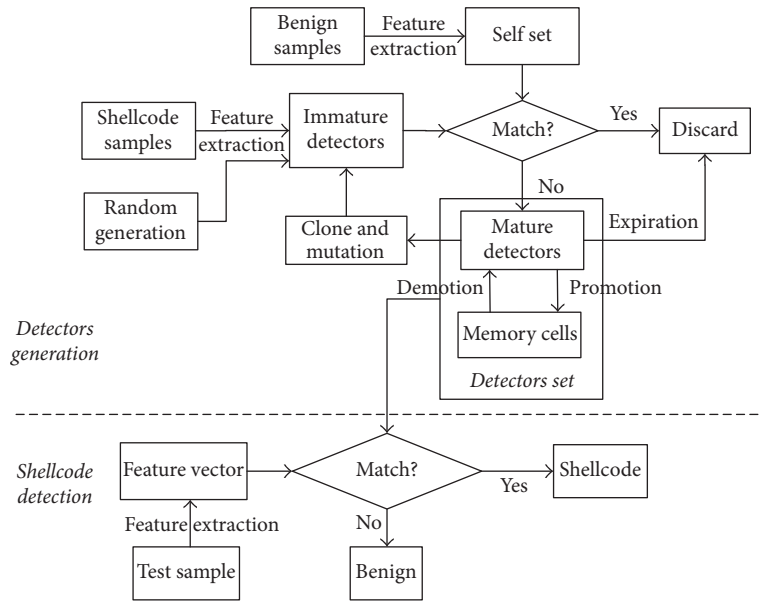


Figure 7: The shellcode detection model based on immune theory.

Detectors Generation. The self set is composed of features extracted from benign samples. Based on negative selection algorithm, the immature detector will be eliminated if it matches any element of self set, and after the immune tolerance it becomes mature detector. Mature detectors with high affinity will be cloned and mutated to generate good performance offspring. If the mature detector matches enough shellcode samples (antigen) during its lifetime, it will be promoted to memory cell and ensure long-term survival.

Shellcode Detection. The features extracted from test sample are encoded to feature vector. The feature vector is matched with memory cells and mature detectors in turn. If the vector falls in the hyperellipsoids of a detector, the test sample is identified as shellcode.

4.3. Detectors Generation. The detector set is $D = \{d_1, d_2, \ldots, d_{Nd}\}$, and Nd is the total number of detectors. The self set is $S = \{s_1, s_2, \ldots, s_{Ns}\}$, and Ns is the total number of self elements.

The immature detectors have three sources: features of shellcodes, random generation, and mutation of cloned detectors. The maturation process of immature detectors based on negative selection algorithm is described as shown in Algorithm 1.

4.4. Detectors Optimizing Base on Clonal Selection Algorithm. The immune system has the ability of learning and memorization. In order to improve the performance of hyperellipsoid detectors, through the adjustment and optimization of its geometry, the nonself space coverage rate can be improved. To reach better shellcode detection effect, the higher-affinity detectors are cloned, and better performance offspring of hyperellipsoid detectors is generated through genetic variation.

Input: S, the self set
(1) generate d, a immature hyperellipsoid detector
(2) repeat for every s_i in $S = \{s_i, i = 1, 2, \ldots, Ns\}$
(3) if $(s_i - \omega)^T \mathrm{A}(s_i - \omega) > 1$, then $i = i + 1$
(4) else d is eliminated, go to (1):
(5) until $i = Ns$, d is matured and $D = D \cup \{d\}$
(6) return D

Algorithm 1

4.4.1. Clone Selection Algorithm. Detectors optimization is a process to cover more nonself space while reducing the overlap with other detectors. The number of cloned offsprings $\mathrm{Num}(d)$ of detector d is proportional to its affinity $a(d)$ and $\mathrm{Num}(d)$ is calculable as

$$\mathrm{Num}\,(d) = \alpha * a\,(d)\,, \tag{12}$$

where α is the clone coefficient. The affinity of detector d is defined as follows:

$$a\,(d) = e^{N(d,A) - c * O(d,D)}, \tag{13}$$

where A is the nonself antigen set and D is detector set. $N(d, A)$ represents the number of antigens covered by detector d. $O(d, D)$ represents the overlap of d with other detectors, and c is the overlap penalty coefficient.

To compute the overlap of a hyperellipsoid detector, we employ a 2^n-way tree. A 2^n-way tree partitions an n-d space into 2^{nk} hyperrectangles, where k is the number of levels of the tree. Each node in the 2^n-way tree is checked whether other detectors besides detector d have covered it, and then the overlap $O(d, D)$ can be calculated.

4.4.2. Genetic Variation of Hyperellipsoid Detectors. Evolutionary algorithms employ mutation as a variation operator to search the solution space. The mutation algorithms need to fulfill two design goals. First, through a finite series of mutations, any valid hyperellipsoid can be mutated into any other valid hyperellipsoid. Second, the mutation is random but should not be too far from the original hyperellipsoid. To get high affinity hyperellipsoid detectors, three kinds of mutation means are used [18]. Let d be a hyperellipsoid detector defined by (5).

Orientation Mutation. The directions of semiaxis determine the orientation of a hyperellipsoid. The columns of V are orthonormal semiaxis of d. Taking a 2-d plane for example, let r_1 and r_2 be the vectors of two semiaxis that are chosen randomly from the n semiaxis of d. To accomplish the rotation, a small angle θ is chosen from a Gaussian distribution with mean $u = 0$ and standard deviation $\sigma = \pi/2$ radians. The randomly chosen semiaxes r_1 and r_2 are both rotated by θ to produce new semiaxis, whose vectors are r_1^m and r_2^m.

$$\begin{aligned} r_1^m &= \cos(\theta)\, r_1 + \sin(\theta)\, r_2 \\ r_2^m &= -\sin(\theta)\, r_1 + \cos(\theta)\, r_2. \end{aligned} \tag{14}$$

It is possible to achieve any orientation through a series of 2-d rotations and the mutation is small by virtue of the Gaussian distribution.

Center Mutation. The center point of d is an n-d vector $\omega = (\omega_1, \omega_2, \ldots, \omega_n)$. Let $\omega^m = (\omega_1^m, \omega_2^m, \ldots, \omega_n^m)$ be the mutated center, and ω^m should be near ω. The center mutation operator chooses ω_i^m from a Gaussian distribution with mean $u = \omega_i$, and the standard deviation σ is a parameter that can be changed. Hence, the center mutation operator can return any n-d vector, and the Gaussian distribution guarantees that the mutated center usually keeps near its original location.

Semiaxis Length Mutation. According to (10), the semiaxis length mutation is achieved when Λ is manipulated. The only constraint on the output of the semiaxis length mutation operator is that Λ^m must be diagonal with positive entries. Let l_i^m be l_i after mutation. l_i^m obeys Gaussian distribution with mean $u = l_i$ and the standard deviation σ is a parameter that can be changed. A validation is performed to ensure that $l_i^m > 0$. The mutation of semiaxis length can return any valid Λ, and again the Gaussian distribution guarantees that the mutated semiaxis length is usually close to the original semiaxis length.

4.5. Memory Detectors. The immune system has the ability to memorize the antigens that had appeared before. When the same antigens invade the body again, the immune system will make a stronger response to eliminate the antigens more quickly. To improve the shellcode detection effect, based on the principle of immune memory, detectors with good detection performance will be promoted to memory cells.

Each mature detector has a maximum lifespan. If a detector can match enough shellcode samples exceeding the threshold in its life cycle, it is promoted to a memory cell, or it is eliminated due to the end of life. There are two main reasons why a mature detector should be eliminated: first, its coverage space is far from the feature vector of most shellcode samples; second, its coverage space has serious overlap with other detectors, and other detectors match the shellcode in advance.

In order to maintain a stable total number of memory cells, a promotion and demotion mechanism is made using the LRU (least recently used) algorithm. Letting C be the maximum capacity of memory cell set, the mature detector newly promoted is added to the memory cell set, and if C is reached, according to LRU, the demoted memory cell will become a new mature detector with an age of 0.

4.6. Shellcode Detection. The shellcode detection process is shown in Figure 7. The assembly mnemonic sequences and API function sequences are extracted from test samples and then split into n-gram features. The feature set is $F = \{f_1, f_2, \ldots, f_N\}$, and, based on the TF-IDF, we calculate the weight of each feature, denoted by v_i. Finally, the feature vector $v = (v_1, v_2, \ldots, v_N)$ is generated. The shellcode detection process is described as shown in Algorithm 2.

The feature vector $v = (v_1, v_2, \ldots, v_N)$ is verified by each detector in D. If $(v - \omega)^T \mathrm{A} (v - \omega) < 1$, the vector v (an n-d point) falls in the coverage space of detector d_i (an n-d hyperellipsoid); therefore, sample s is classified as shellcode. If the sample s cannot be identified by any detector in D, it is then classified as benign.

5. Experimental Design

5.1. Collation of Shellcode and Benign Samples. We collect 600 nonencoded shellcode samples from a variety of sources, such as Metasploit and exploit-db. These samples are executable on IA-32 architecture. We also collect 600 benign samples, including code segments of executable files (e.g., exe and dll) and network package segments (e.g., http protocol), with the random length between 50 and 300 bytes.

To evaluate the detection effect of the proposed algorithm for polymorphic shellcodes, we use a variety of polymorphic engines to encode the shellcodes. These engines include Clet, ADMmuate, Jempiscodes, TAPioN, and engines of Metasploit (using the MSFvenom command line interface), such as Alpha2, Countdown, JmpcCallAdditive, Pex, Call4_dword_xor, and Shikata_ga_nai.

5.2. Algorithm Implementation. To get the assembly instruction sequences, we implement the customized linear sweep disassembly algorithm based on capstone, a lightweight multiplatform, and multiarchitecture disassembly framework. We can get the API function sequences through simulation execution using scdbg, a shellcode analysis application built around the libemu emulation library.

We use TfidfVectorizer in scikit-learn to extract the n-gram features and calculate the feature vectors of shellcodes and benign samples. We use Matlab to implement the artificial immune theory, including the detectors generation based on negative selection, the detectors optimization based on clone and mutation, and the shellcode detection.

Input:
$v = (v_1, v_2, \ldots, v_N)$, the feature vector of test sample s
D, the detector set (both mature detectors and memory cells)
(1) repeat for every d_i in $D = \{d_i, i = 1, 2, \ldots\}$
(2) if $(v - \omega)^T A(v - \omega) < 1$ then sample s is a shellcode
(3) else $i = i + 1$
(4) until $i = |D|$, sample s is a benign sample
(5) return s (shellcode or benign)

Algorithm 2

5.3. Experimental Results and Analysis. The collected shellcode and benign samples are both equally divided into training samples set and test samples set. The immune self set is composed of feature vectors of benign samples in training samples set. The feature vectors of training shellcode samples are one source of immature detectors.

5.3.1. Comparison of Different Detectors Encoding Method. The shellcode detection performances of constant-sized hyperspheres, V-detectors, and hyperellipsoids are compared. Different mutation means are applied to these three kinds of shapes for detectors with real-valued representation. Center mutation is applied to hypersphere detectors with constant radius. Center mutation and radius length mutation are applied to V-detectors. Center mutation, orientation mutation, and semiaxis length mutation are applied to hyperellipsoids detector. These three kinds of real-valued representation detectors have different nonself space coverage effect which is translated to the shellcode detection rate to a large extent. Taking the same genetic variation generation ($T = 200$), we compare the shellcode detection effect of the three collections of detectors with the same numbers Nd.

As shown in Figure 8, hyperellipsoids detectors have the highest detection rate, followed by V-detectors, and the detection rate of constant-sized hypersphere is the worst. With a relatively small number of detectors (e.g., $Nd = 200$ and $Nd = 400$), the difference between the three kinds of detection rate is relatively large. Even when the number of hyperellipsoids detectors is small, they can still effectively cover the nonself space through changing the semiaxis length and the orientation. The nonself space can be sufficiently covered when the number of detectors is enough. But some small holes that are so close to self points can not be covered by constant-sized hypersphere, so even though the detector number is large enough its detection rate is lower than the other two.

5.3.2. Detection Effect for Polymorphic Shellcodes. To generate polymorphic shellcodes, the 300 shellcodes in training set are encoded by 10 polymorphic engines, and we get 3000 polymorphic samples. Half of the polymorphic samples are randomly chosen to the training set to generate detectors, and the rest is used to test the detection effect of our algorithm. We proposed a comprehensive analysis technology for shellcode, including both static disassembly analysis and dynamic simulation analysis, which is more effective for detecting polymorphic shellcodes, as shown in Table 1.

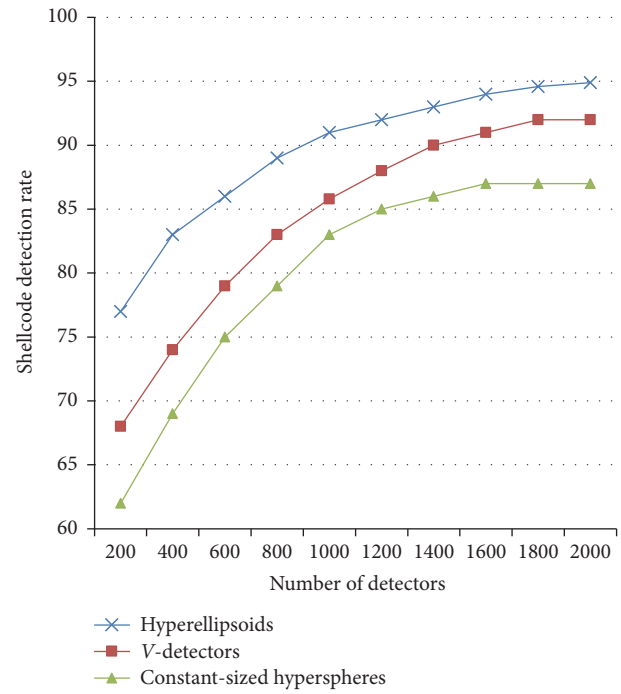


Figure 8: The detection rate of different encoding methods.

Both 1000 detectors were generated for static detection and our algorithm ISDA. As shown in Table 1, with the increase of genetic variation generation, the distribution of detectors is more reasonable; as a result the shellcode detection rate shows an upward trend. Especially, in the early stage of genetic variation, the shellcode detection rate grows fast. As the nonself space coverage trending is stable, the growth speed of the shellcode detection rate slows down. For ordinary shellcodes, both the static detection and ISDA can achieve good performance, and the detection rate of ISDA is a little higher. For polymorphic shellcodes, the features for disassembly instructions are hidden, so the detection rate of static detection is very low, but our algorithm ISDA can maintain a high detection rate.

5.3.3. False Positive Rate Analysis. The benign samples we collected include code segments of executable files and network package segments. We analyze the false positive rate of these two kinds of benign samples. To achieve more valuable results, the k-fold cross validation is used, and in

Table 1: Detection performance for polymorphic shellcodes.

Genetic variation generation	Ordinary shellcodes		Polymorphic shellcodes	
	Static detection	ISDA	Static detection	ISDA
0	78.5	82.1	67.8	82.8
100	82.2	86.0	70.2	86.1
200	85.7	90.9	74.3	89.6
300	88.1	91.8	76.1	90.9
400	89.7	92.9	78.7	92.3
500	90.4	93.6	79.5	92.4

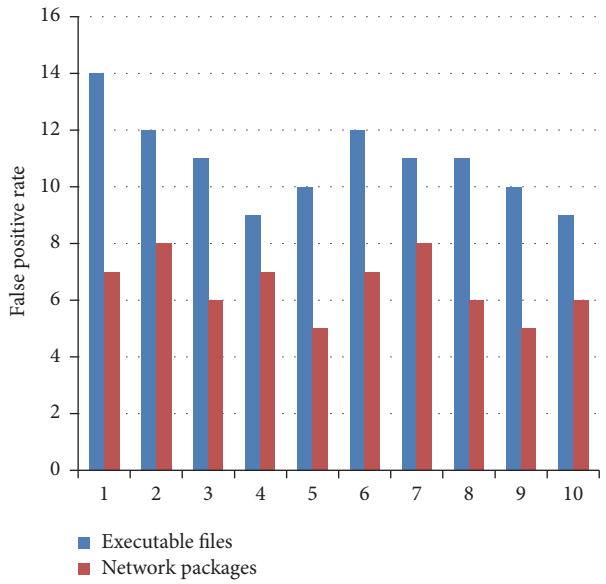


Figure 9: The false positive rate.

our experiment let $k = 10$. The benign samples are randomly partitioned into 10 equal sized subsamples. The shellcode detection algorithm is trained and tested 10 times.

As shown in Figure 9, the false positive rate of code segments is higher than network package segments. The code segments of executable files contain disassembly instructions that may perform like shellcodes, such as calling some API functions for network connection, file operation, and program launch.

Self-modifying code can modify its instructions dynamically at run-time; the eventually execute instructions do not appear in the initial code segment. That is why static signature-based detection method lacks flexibility to tackle polymorphic shellcodes. In this paper, we propose a technique for polymorphic shellcode detection, which combines both static analysis and dynamic analysis. Depending on the execution behavior, we can differentiate between polymorphic shellcodes and benign samples. But our approach also has some drawbacks. It is an offline shellcode detection method; to be more practical it should be embedded to intrusion detection system (IDS) that inspects the network packets for suspicious shellcodes. The dynamic analysis technology can increase the shellcode detection rate, but it is also a bottleneck when handling high-speed network traffic. Detecting such attacks remains an open problem, and it will be our next step research direction.

6. Conclusion

Artificial immune system (AIS) is a computational intelligence system that shows great promise recently. AIS has many good characteristics: self-organization, learning, classification, adaptation, and diversity, and it has been applied to many problem domains, such as anomaly detection, multioptimization problems, fault diagnosis, and network security. In this paper, we introduce an immune-inspired shellcode detection algorithm (ISDA).

Shellcodes are binary code segments with malicious instructions that are usually treated as normal input data by the target program. To improve the nonself space coverage rate, the hyperellipsoid detectors encoding method is adopted. Because the static detection methods can be bypassed using techniques such as self-modifications, both static assembly instruction sequence and dynamic API function sequence are obtained in our paper.

The main purpose of our work is to solve an important network security problem, which is vulnerability exploit through shellcode, especially unseen and polymorphic shellcode. The proposed shellcode detection method is a useful way to identify a zero-day attack by inspecting the payload data. A zero-day vulnerability brings much more risk, because it is unknown to vendors of the target software and no patches or advises are available to mitigate it. The shellcode detection algorithm can be embedded in IDS and firewall to defeat zero-day attack.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Key R&D Program of China (2017YFB0802804), the National Nature Science Foundation of China (61602489), and the CERNET Innovation Project (NGII20160405).

References

[1] Rapid7, "Metasploit: the world's most used penetration testing framework," 2017, https://www.metasploit.com/.

[2] A. Basu, A. Mathuria, and N. Chowdary, *Automatic Generation of Compact Alphanumeric Shellcodes for X86*, vol. 8880 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, 2014.

[3] N. Verma, V. Mishra, and V. P. Singh, "Detection of alphanumeric shellcodes using similarity index," in *Proceedings of the 3rd International Conference on Advances in Computing, Communications and Informatics, ICACCI 2014*, pp. 1573–1577, India, September 2014.

[4] Z. Zhao and G.-J. Ahn, "Using instruction sequence abstraction for shellcode detection and attribution," in *Proceedings of the 1st IEEE International Conference on Communications and Network Security, CNS 2013*, pp. 323–331, USA, October 2013.

[5] D. Lukan, "Shellcode detection and emulation with libemu," 2014, http://resources.infosecinstitute.com/shellcode-detection-emulation-libemu/.

[6] M. Polychronakis, K. G. Anagnostakis, and E. P. Markatos, "Network-level polymorphic shellcode detection using emulation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 4064, pp. 54–73, 2006.

[7] Y. Luo, C. Xia, and Y. Li, "A Polymorphic Shellcode Detection Method Based on Dual-Mode Virtual Machine," *Journal of Computer Research and Development*, vol. 51, no. 8, pp. 1704–1714, 2014.

[8] B. Gu, X. Bai, Z. Yang, A. C. Champion, and D. Xuan, "Malicious shellcode detection with virtual memory snapshots," in *Proceedings of the IEEE INFOCOM 2010*, USA, March 2010.

[9] D. Dasgupta, S. Yu, and F. Nino, "Recent advances in artificial immune systems: models and applications," *Applied Soft Computing*, vol. 11, no. 2, pp. 1574–1587, 2011.

[10] Y. Tan, "Artificial Immune System: Applications in Computer Security," *Artificial Immune System: Applications in Computer Security*, pp. 1–174, 2016.

[11] T. Gong and B. Bhargava, "Immunizing mobile ad hoc networks against collaborative attacks using cooperative immune model," *Security and Communication Networks*, vol. 6, no. 1, pp. 58–68, 2013.

[12] J. Brown, M. Anwar, and G. Dozier, "Detection of Mobile Malware: an Artificial Immunity Approach," in *Proceedings of the 2016 IEEE Symposium on Security and Privacy Workshops, SPW 2016*, pp. 74–80, usa, May 2016.

[13] S. Singh, "Artificial Immune System Based Statistical Model for Intrusion Identification," *International Journal of Computer Science and Mobile Computing*, vol. 4, no. 6, pp. 170–176, 2015.

[14] I. Idris, A. Selamat, and S. Omatu, "Hybrid email spam detection model with negative selection algorithm and differential evolution," *Engineering Applications of Artificial Intelligence*, vol. 28, pp. 97–110, 2014.

[15] S. Forrest, L. Allen, A. S. Perelson, and R. Cherukuri, "Self-nonself discrimination in a computer," in *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 202–212, May 1994.

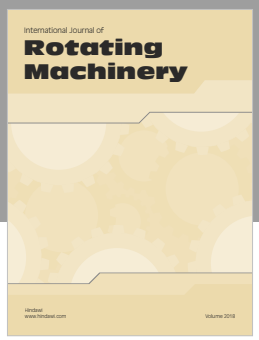
[16] F. A. Gonzalez and D. Dasgupta, "Anomaly detection using real-valued negative selection," *Genetic Programming and Evolvable Machines*, vol. 4, no. 4, pp. 383–403, 2003.

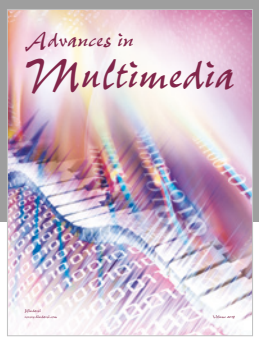
[17] Z. Ji and D. Dasgupta, "Real-valued negative selection algorithm with variable-sized detectors," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 3102, pp. 287–298, 2004.

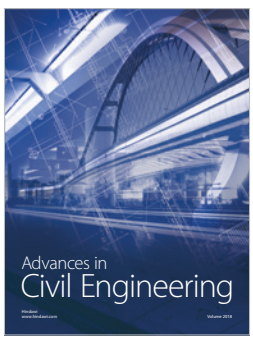
[18] J. M. Shapiro, G. B. Lament, and G. L. Peterson, "An evolutionary algorithm to generate hyper-ellipsoid detectors for negative selection," in *Proceedings of the GECCO 2005 - Genetic and Evolutionary Computation Conference*, pp. 337–344, usa, June 2005.

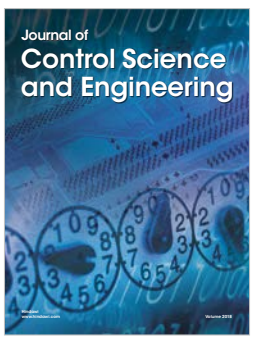
[19] L. N. de Castro and F. J. von Zuben, "Learning and optimization using the clonal selection principle," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, pp. 239–251, 2002.

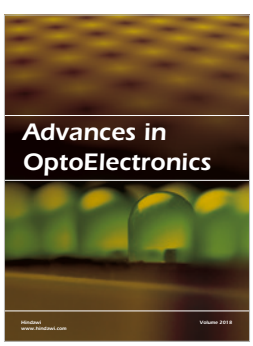
[20] D. Zimmer, "Scdbg is a shellcode analysis application built around the libemu emulation library," 2011, http://sandsprite.com/blogs/index.php?uid=7&pid=152.

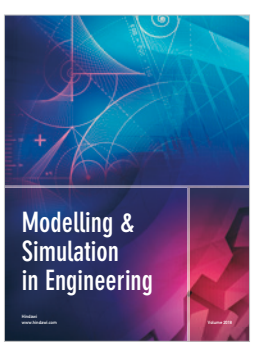


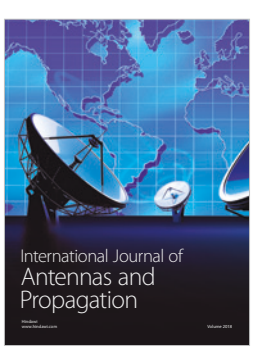


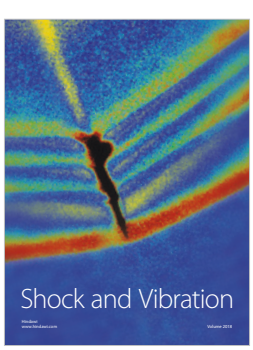


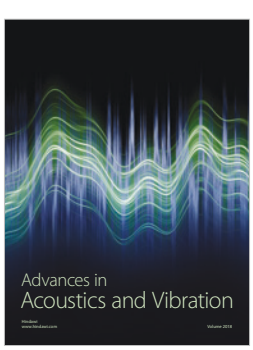












Hindawi

Submit your manuscripts at
www.hindawi.com