Published in IET Information Security
Received on 12th June 2010
Revised on 22nd February 2011
doi: 10.1049/iet-ifs.2010.0136

ISSN 1751-8709

# Eigenviruses for metamorphic virus recognition

*M.E. Saleh*[1] *A.B. Mohamed*[2] *A.A. Nabi*[3]
[1]*Integrated Simulators Complex, Arab Academy for Science and Technology, Alexandria, Egypt*
[2]*Computer Engineering Department, Arab Academy for Science and Technology, Alexandria, Egypt*
[3]*Informatics Institute, Scientific Research and Technology Applications, Mubarak, P.O. Box: 21934, Alexandria, Egypt*
*E-mail: msaleh@cs.utsa.edu*

**Abstract:** Metamorphic virus recognition is the most challenging task for antivirus software, because such viruses are the hardest to detect as they change their appearance and structure on each new infection. In this study, the authors present an effective system for metamorphic virus recognition based on statistical machine learning techniques. The authors approach has successfully scored high detection rate for tested metamorphic virus classes and very low false-positive errors. The system is also able to learn new patterns of viruses for future recognition. The authors conclude the results of their simulation with results analysis and future enhancements in the system to detect other virus classes.

## 1 Introduction

Computer malware analysis and detection is considered an important field in computer security. Every year thousands of new malware arise and cost the world billions of dollars [1]. Since the first virus appeared in the wild, millions of different viruses emerged and attacked computers. Computer viruses have evolved a lot along past decades. When viruses were first found, they were executed exactly as they were written in the executable file. Soon, virus researchers could distinguish each virus with a unique pattern of bytes that resembles its signature. Therefore those viruses could be easily detected on the basis of their signature.

Signature detection is very effective for virus detection, in which antivirus software search for a unique constant pattern or a sequence of bytes in the virus body [2]. Consequently, virus writers had to develop their code to evade detection so that self-encrypting viruses emerged.

Self-encrypting viruses use a decryptor at the beginning of the file to decrypt the virus body on execution, and each generation of the file uses a different key generated when the virus is executed. This makes signature detection impossible as the virus body changes on each infection. However, the problem is not hard as it seems since the decryptor itself always has to be unencrypted, and then virus researchers can extract the signature from it as long as the decryptor is long and unique enough [3].

Virus writers fought back by oligomorphic type of viruses, in which the virus carries some different decryptors with it, and changing the decryptor on each infection. The first known oligomorphic virus is called Whale and another famous one is Memorial that has 96 different decryptors [4]. A virus is said to be oligomorphic if it is capable of mutating its decryptor only slightly [2]. This makes antivirus researchers unable to extract a constant pattern from the decryptor to be the signature, yet they did not give up; another approach of detection was used, it is the emulator. By using an emulator, the antivirus scanner can emulate code execution and after full decryption of the body, a signature can then be extracted [2].

A further step taken by attackers is the creation of polymorphic viruses. As biological viruses can be mutated in new infections, viruses' writers took the idea and made their virus decryptor mutate in every new infection. They attached a special module called mutation engine that is responsible for mutating the decryptor to another form, yet maintains the same behaviour. In this way, polymorphic viruses can mutate their decryptors to billions of different forms, which make it virtually impossible to be detected using string signature [3].

An intuitively predictable step was taken after that, instead of mutating the decryptor only, virus writers mutated the entire virus body and thus encryption will not be needed any more to evade signature detection, and that was the beginning of metamorphic viruses era [5]. Metamorphic viruses use many techniques to mutate and obfuscate their code while maintaining the same function in each generation. We will explain some of these techniques in the following subsections.

### 1.1 Instruction reordering

Code obfuscation techniques used by metamorphic viruses include instruction reordering in which the virus divides its code into blocks of certain size, and then the mutation engine reorders these blocks by inserting jump instructions between the blocks while maintaining the programme result. This technique is also called code transportation [6].

### 1.2 Garbage code insertion

Another technique is garbage code insertion, trash insertion or dead code insertion. In this method, the mutation engine inserts unneeded instructions in random locations in the code, which makes the code look very different in each generation. Examples of trash instructions are NOP that does absolutely nothing ('No Operation'), 'mov R1, R1', 'push R1' followed by 'pop R1', 'shl R1, 0' and many other combinations. Thus, by inserting these trash instructions in random locations in the virus, the virus has no constant body that can be detected using signature scanning [7].

### 1.3 Register swapping

Register swapping technique, as it sounds, concerns changing the register operands of an instruction but not changing the instruction itself. An example of this type of virus is W95/RegSwap virus. Although the resulting morphed virus looks different from the previous version, the variability is not very high and the virus can still be detected by using a half-byte wildcard in signature string scanning [8]. Register swapping technique is also known as register renaming or register exchange.

### 1.4 Instruction substitution

In this technique, the virus is able to replace some of its instructions with equivalent ones, while keeping the semantics of the instructions the same. This technique was used in MetaPHOR mutation engine that appeared in 2002 and was also used in W95/Zmist virus [9].

### 1.5 Instruction transposition

Transposition of instructions is permutation of some instructions and changing their execution order. Nevertheless, instruction transposition cannot be done with any group of instructions. They have to be unrelated; in other words, they are not dependent on each other.

For example, the instructions 'mov eax, edx' and 'add ecx, 5' have no dependency and therefore can be transposed safely [10]. W95/Zmist virus that appeared in 2001 used this technique in its metamorphic engine.

Owing to these morphing techniques used by this type of virus, the detection of such viruses is extremely difficult and different from usual detection techniques. Since once the virus analysts find an appropriate unique pattern of bytes in the virus body, the virus changes itself to be very different from the previous generations. Therefore other techniques must be sought to detect such viruses.

In the next section, we will show some techniques used to detect metamorphic viruses; subsequently, we will explain the proposed approach used in this paper for virus detection, then we will depict how we experimented our approach and show its results, after which we will conduct an analysis for the result and evaluation of the technique.

## 2 Related work

Although many techniques have been proposed for metamorphic virus detection, few of them reached commercial products because of their computational feasibility and acceptable range of false positives. In this section, we will review some of the commercially used techniques and some other experimentally proposed ones.

One of the commonly used techniques in commercial antivirus applications is geometric detection [2]. Geometric detection technique detects the changes in the infected file structure. For example, when W95/Zmist virus infects a file, it increases its virtual size of the data section to be at least 32 KB larger than the physical size, so that such files can be suspicious of being infected by W95/Zmist. However, this method is prone to being false positive as some safe run-time compressed files have the same symptoms [4].

Another method used commercially is wildcard and half-byte scanning, which is typically used for viruses that use register-swapping techniques mentioned in Section 1.3. Fig. 1 shows two generations of W95/Regswap [8]; the bold bytes of opcode are constants between both generations, so that wildcard scanning can be used. The following signature can be used to detect the example in Fig. 1:

??040000008B??0C00000081C0880000008B??89????????
????2B??

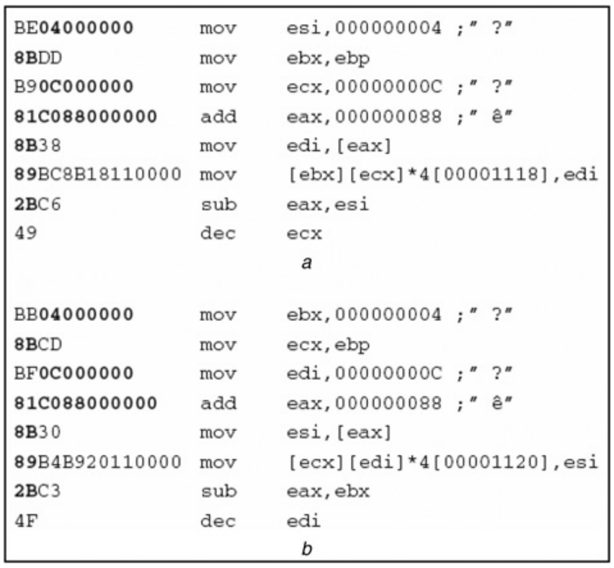
where '?' denotes variable half-byte.

Some of the non-common opcodes between both generations have half-byte similarity, so that half-byte wildcard can be combined with byte wildcard to produce a more accurate detection string as follows:

B?040000008B??0C00000081C0880000008B??89????????
?????2BC?

In 2006, a static analysis heuristic detection method by an arbitrary length of control flow graphs was presented by [11], assuming that the virus does not change its control flow during propagation, but if it does, the authors proposed applying nodes alignment for detection [11]. Another method is the zeroing transformation method that is used to reverse the effect of some obfuscation techniques performed by the mutation engine. The resulting form of the programme after applying zeroing transformation on it is called zero form. Their method showed considerable decrease in the number of variants of subject programmes considered in their test [12]. However, the zeroing transformation method does not work against expression rewriting at a low level. For example, the statements in Fig. 2*a* are equivalent to Fig. 2*b* but zeroing transformation cannot recognise that [13].

Hidden Markov model (HMM) is another method for metamorphic virus detection proposed by Wong and Stamp



```
BE04000000     mov    esi,000000004 ;" ?"
8BDD           mov    ebx,ebp
B90C000000     mov    ecx,00000000C ;" ?"
81C088000000   add    eax,000000088 ;" ê"
8B38           mov    edi,[eax]
89BC8B18110000 mov    [ebx][ecx]*4[00001118],edi
2BC6           sub    eax,esi
49             dec    ecx
```

*a*

```
BB04000000     mov    ebx,000000004 ;" ?"
8BCD           mov    ecx,ebp
BF0C000000     mov    edi,00000000C ;" ?"
81C088000000   add    eax,000000088 ;" ê"
8B30           mov    esi,[eax]
89B4B920110000 mov    [ecx][edi]*4[00001120],esi
2BC3           sub    eax,ebx
4F             dec    edi
```

*b*

**Fig. 1** *Two Regswap infection code fragments [8]*

*a* Version *c* of the code
*b* Version *c+1*, a mutated version

```
OR      EAX, 81818181
OR      EAX, 42424242
SUB     EAX, C3C3C3C3
OR      EAX, 3C3C3C3C
ADD     EAX, C3C3C3C4
              a

MOV     EAX, 00000000
              b
```

**Fig. 2** *Two equivalent code fragments [13]*

*a* A rewritten equivalent version of (b)
*b* Single instruction that set EAX to zero

[14]. HMM, is a statistical model that was first presented by Baum and then it was used in pattern and speech recognition [15], after which it began to be used in biological sequences and DNA analysis [16]. HMM was used by Wong and Stamp to detect metamorphic viruses [14]. They showed good results in detecting some metamorphic viruses that have relatively high similarity among generations. However, the authors did not show the results if a well-known hard metamorphic virus such as W95/Zmist, W95/Zperm or W95/Bistro is tested. Besides, it suffers from unacceptable rate of false-positive when testing normal non-virus files against the system [17].

Metamorphic virus detection is still an open problem in computer virology science; there is no high-performance method for detecting a wide range of this type of virus [17, 18], yet some commercial techniques are doing a fairly good job till now.

## 3 Proposed approach

In this paper, we present a detection approach based on a well-known face recognition technique called Eigenfaces [19]. Eigenfaces is a technique that is used widely and effectively for face recognition. Eigenfaces approach states that every face is a linear combination of other basic set of faces called 'Eigenfaces'. The same person could have two different images owing to change in age or light conditions or pose of face; in this case, the Eigenfaces differ in some basic faces, but not all of them. The method measures the extent of similarity and difference among the faces that would decide whether they can be mapped to a known face in the database or not.

Eigenfaces approach takes advantage of principal component analysis that is used extensively in information theory. Eigenfaces approach treats the problem of face recognition as a 2 dimensional (2D) recognition problem as faces are normally upright, and ignores the geometric details of the face, which makes it relatively computationally easy and simple. The approach functions by first acquiring a set of face images, and then determines the vectors or axes that span across the significant variations among the face images; those vectors are called eigenvectors, and the space defined by these vectors is called eigenspace. Since those eigenvectors when drawn give face-like images, they are called Eigenfaces.

The set of images are then projected – or in other words represented in terms of eigenvectors – into the eigenspace or feature space, and then the system characterises each face by the weighted sum of the Eigenfaces features. Therefore to determine whether a new face belongs to one of the initial set or not, the new input image is projected into the eigenspace of the set of image and a distance classifier is computed between the new image's weight against each weight in the initial set. If the distance is below some threshold that was determined previously, then the image belongs to its closest class of face image, otherwise, the image does not belong to that class.

Our approach uses Eigenfaces approach with some modifications. As different images of the same person have some similarities between themselves, different copies of a metamorphic virus also have common similarities. Principal components analysis, is a statistical tool used in the Eigenfaces method, which is used to quantify these similarities.

In the following sections of the paper we will refer to 'Eigenviruses' as the basic set of binaries that construct the virus which corresponds to Eigenfaces that when linearly combined constructs the face. 'Training set' is the database of viruses by which our system is trained to recognise. However, 'test set', is the set of input viruses' replicates to be recognised. The term 'replicate' refers to a file that is a copy of a virus, and the term 'virus' – in this context – is a general term that identifies the type of one or more replicate files and the term 'virus class' is the set of replicates that belong to a virus. Therefore it is the mission of the technique to map an input replicate to its virus class, as Eigenfaces approach maps the input image to its face class.

The system functions by first acquiring a set of replicate files from different viruses, with more than one file from each virus. This set will be the training set of our model, and then we determine the vectors or axes that span across the significant variations among the replicate files, and those vectors are called eigenvectors and they construct a space called eigenspace. Since those eigenvectors when linearly combined together with certain weights they give one of the original virus replicates according to the weight, then they can be called 'Eigenviruses'. The set of the original replicates are then projected into that eigenspace or feature space constructed by the eigenviruses by finding the weights of each replicate. Thus, the system characterises each virus replicate by the weighted sum of eigenviruses. Then to determine if a new virus belongs to one of the initial set or not, the new input virus replicate is projected into the eigenspace of the set of the initial virus replicates and a distance classifier is computed between the new replicate's weight vector and each weight vector in the initial set. If the distance between the input file and the replicate vector in the initial set is below some threshold that was determined previously, then they belong to the same virus, otherwise the replicate does not belong to that virus.

### *3.1 Preparations*

In order to apply the Eigenfaces approach on binary files, some preparations and modifications had to be made to the approach. First, it is important to remove any data in the file that is not directly relevant to the virus body. In order to do this, we removed the portable executable (PE) header of the subject file as it is not important in virus recognition in our approach, and then we extracted the malicious code of the executable file to be tested against our dataset and saved it into a file. By using infection flags, certain sections of PE files could only be subject to test, which reduces the complexity of the test. We also emphasise that the proposed approach focuses on recognition and classification tasks of a malicious pattern and it is not responsible for locating and extracting the malicious pattern from a file.

Since the approach requires that all inputs have to be in the same length, the input code is padded with zeros to a certain length specified when building the training set. This length is called 'Eigenvirus length'. The Eigenvirus length is decided based on the largest virus in the training set. Based on this length, every other input file or virus replicates in the training set that has shorter length must be padded to the eigenvirus length. If a test input file is larger than the eigenvirus length, then the virus is chopped from the end to be equal to that length.

Unlike the original approach of Eigenfaces, we did not remove the mean vector of the samples. Removing the mean face of face samples in the Eigenfaces approach seems intuitive as all faces of different people have obvious common features, so that removing common features makes the remaining features more descriptive for the face. However, when working with binary virus files, this is not the case. As different viruses can not only look very different, but can also look similar to normal applications, subtracting the mean vector was not applied here. In addition, the original Eigenfaces approach considered one space threshold for the entire eigenspace, while we compute $M$ space thresholds for the $M$ virus classes in the training set. This will be further explained in the next section as well as in Section 5.

## 3.2 Model description

In this section, we will describe the algorithm used to project the set of virus replicates into the eigenspace, as well as how a new input replicate can be recognised as belonging to a virus class in the initial set.

To construct the training set, the following steps are carried out:

1. Acquire an initial set of virus replicate files. $M$ training replicate files.
2. Determine the largest file size; let us say of size $N$ bytes. Then pad the other files with zeros to be all of size $N$.
3. Represent each virus replicate as a column vector $\boldsymbol{\Phi}$. Therefore $\boldsymbol{\Phi}$ is an $N \times 1$ vector.
4. Incorporate all individual virus replicate vectors into one $N \times M$ matrix $\boldsymbol{A}$. $\boldsymbol{A} = [\boldsymbol{\Phi}_1, \boldsymbol{\Phi}_2, \dots, \boldsymbol{\Phi}_M]$.
5. Find the eigenvectors $\boldsymbol{u}$ of the covariance matrix $\boldsymbol{C}$, where $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$. However, since $\boldsymbol{C}$ would be $N \times N$ which is computationally infeasible to get its eigenvectors for large viruses, and also $\boldsymbol{C}$ is not needed in any further computations, we should obtain eigenvectors of $\boldsymbol{C}$ without computing the value of $\boldsymbol{C}$ itself.

Suppose a matrix $\boldsymbol{L} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, where $\boldsymbol{L}$ is an $M \times M$ matrix and $\boldsymbol{v}_i$ is an eigenvector of $\boldsymbol{L}$. So

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}v_i = \lambda_i\, \boldsymbol{v}_i$$

where $\lambda_i$ is the eigenvalue, by multiplying both sides by $\boldsymbol{A}$ it yields

$$\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{v}_i = \lambda_i\, \boldsymbol{A}\boldsymbol{v}_i$$

However, $\boldsymbol{C} = \boldsymbol{A}\,\boldsymbol{A}^{\mathrm{T}}$, and so $\boldsymbol{A}\boldsymbol{v}_i$ is an eigenvector of $\boldsymbol{C}$. As a result, if $\boldsymbol{v}$ is the set of $M$ eigenvectors of $\boldsymbol{L}$, then $\boldsymbol{A}\boldsymbol{v}$ is the set of eigenvectors of $\boldsymbol{C}$.

Hence, $\boldsymbol{u} = \boldsymbol{A}\boldsymbol{v}$, then we can use $\boldsymbol{v}$ to calculate the $M$ largest eigenvectors of $\boldsymbol{C}$, where $M \ll N$ as $M$ is the number of training virus replicates.

6. Sort the eigenvectors according to their associated eigenvalues. The higher the eigenvalue, the more important is the eigenvector in describing the features.
7. We can then choose a number of eigenvectors $M'$ with high eigenvalues to describe the eigenspace, since not all eigenvectors represent important features of the space.
8. When projecting each virus replicate $\boldsymbol{\Phi}_i$ into the eigenspace, each replicate can be represented as a linear combination of eigenvectors and weights.

$$\boldsymbol{\Phi}_i = \sum_{j=1}^{M'} \omega_j \mu_j, \quad \text{where} \quad M' \le M$$

The weights of each replicate $i$ can be calculated as

$$\omega_j = \mu_j^{\mathrm{T}}\, \boldsymbol{\Phi}_i, \quad j = 1, 2, \dots, M'$$

The weights of the replicate can be combined into a vector $\boldsymbol{\Omega}$, where

$$\boldsymbol{\Omega}_i^{\mathrm{T}} = [\omega_1, \omega_2, \dots, \omega_{M'}]$$

The previous steps were necessary to initialise the system, after which the following steps are used to recognise a new input file:

1. Project the input file $\boldsymbol{\Phi}$ into the eigenspace and determine its weights.

$$\omega_j = \mu_j^{\mathrm{T}}\, \boldsymbol{\Phi}, \quad j = 1, 2, \dots, M'$$
$$\boldsymbol{\Omega}^{\mathrm{T}} = [\omega_1, \omega_2, \dots, \omega_{M'}]$$

2. Determine how much the input file is close to a certain virus by measuring the Euclidean distance from its weight vector to the nearest virus replicate weight vector in the training set. This distance is called 'virus class distance' $\varepsilon$.

$$\varepsilon_k^2 = \|\boldsymbol{\Omega} = \boldsymbol{\Omega}_k\|^2$$

3. $\varepsilon$ should be less than a threshold $\theta$, which is determined heuristically.
4. If we consider all the $M$ eigenvectors to construct the eigenspace, then when a virus replicate is projected in the eigenspace, it can then be reconstructed perfectly as we did not ignore any of its features. However, since we chose $M'$ eigenvectors where $M' < M$, accurate reconstruction of the virus replicate cannot be achieved. So, there will be a difference between the original input replicate vector $\boldsymbol{\Phi}$ and $\boldsymbol{\Phi}_{\boldsymbol{v}} = \Sigma_{i=1}^{M'} \omega_i \mu_j$, where $\boldsymbol{\Phi}_{\boldsymbol{v}}$ is the restoration of the eigenspace projected file vector.

This difference is called 'virus space distance', and can be measured as

$$\alpha^2 = \|\boldsymbol{\Phi} - \boldsymbol{\Phi}_{\boldsymbol{v}}\|^2$$

Space distance measures how much the projected file lost from its features. In other words, it measures how much the eigenviruses represent the virus features. The lower the number, the fewer the loss, the more features are represented by the chosen eigenviruses. Since the chosen eigenviruses quantify the common features of all projected viruses in the space, the space distance can vary according to the virus. For each virus $i$, the space distance $\alpha$ of a newly belonging projected file should be below a threshold $\beta_i$.

There are four possibilities for the input file to be

1. Near from a virus class and near from the virus space of that class: In this case, the input file is recognised as belonging to that virus class.
2. Near from a virus class and far from virus space of that class: This happens when the input file does not belong to any class in the space. However, when projected into the eigenspace, it loses many of its original features that make it looks like one of the candidate virus classes.
3. Far from a virus class and near from virus space of that class: In this case, the input file also does not belong to any virus classes in the space. However, it shares some features with existing classes. False negatives might occur in this case.
4. Far from the virus class and far from the virus space of that class: This case takes place when the input file does not belong to any class in the training set and does not share features with them as well.

## 4 Experiment

In this section, we will describe our simulation that was undertaken to evaluate the proposed approach.

### 4.1 Data used and preparations

We chose four viruses to run our test. They are as follows:

*4.1.1 G2 construction kit:* G2 is a virus construction kit developed by 'dark angel' the same author of 'phalcon/skism mass-produced code generator' which is an earlier virus generator. G2 produces a COM and 16-bit EXE infectors. The kit has a configuration file that can be set to have the desired virus features, and then the kit produces an assembly code according to the configurations. G2 can produce a different virus every time, even though the values in the configuration file remain unchanged. The kit mainly uses equivalent instruction substitution to achieve obfuscation. In our test, we used version 1.0 that was released in January 1993 [20].

*4.1.2 Next-generation virus creation kit:* Next-generation virus creation kit (NGVCK) is a virus creation kit written in Visual Basic that each time it runs it creates a virus code. Each virus created from the NGVCK kit does the same thing. However, every virus has almost a completely different structure, which makes scanning the generated virus with the same scan string almost impossible [21, 22]. NGVCK uses garbage instruction insertion, code reordering and register replacement techniques to obfuscate the generated virus code. NGVCK infects 32-bit executables and have multiple encryption methods; it also provides an anti-debugging code inside the generated viruses. We used NGVCK v0.30 as it is a stable version that was released in June 2001. In the process of generating NGVCK sample files, we maintained the same configuration for all generated files.

*4.1.3 Zperm virus:* Zperm was developed by the notorious virus writer 'Z0mbie' in the year 2000. Zperm virus was one of the first 32-bit viruses for Windows platforms. The virus mainly uses permutation engine to change its order of instructions constantly in each infection, including changing its permutation engine as well [23]. Zperm does not produce constant virus body anywhere as self-encrypting viruses do; instead, it permutates itself by adding and removing jump instructions and garbage instructions to produce a highly different version of the virus. So, detecting the virus cannot be done using scan string [24].

*4.1.4 MetaPHOR virus:* MetaPHOR is a very hard metamorphic virus that was developed by 'Mental Driller' in 2002. In fact, MetaPHOR was a challenge to antivirus researchers when it emerged. It is highly obfuscated and difficult to understand [9]. The virus uses various metamorphism techniques to produce a highly different new form of the virus on each infection. The virus consists of a decryptor and a body. Although the decryptor has a size of 4 KB, the virus body has a size of more than 100 KB. In our test, we used only the decryptor of the virus to test against the system as it is not encrypted and much smaller than the body, and thus we can minimise the training set size. MetaPHOR is also called W32/Simile and W32/Etap.

We generated two sets for our test, a training set and a test set. The training set contains samples virus replicates of each virus; the system uses these samples to learn about each virus. The number of samples needed for each virus differs according to the virus; the more metamorphism is used, the more samples are needed. For constructing the training set, we needed 1, 6, 8 and 15 samples of G2, NGVCK, Zperm and MetaPHOR, respectively, and so our training set has 30 virus files. On the other hand, the test set contains replicates of each virus to test against the system after the learning process is complete. Test set contains 250 different replicates of each virus and so the total number in the test set is 1000 different replicates.

After constructing the training set, we had 30 eigenviruses that describe the features of the training set. We chose only three eigenviruses that have the highest eigenvalues to construct the eigenspace, yet with only three eigenviruses the system showed very good results. The number of eigenviruses needed to construct a descriptive space that holds most features of virus classes is done heuristically.

To measure the false-positive errors in our system, in which a benign programme is classified as malicious, we acquired 250 programmes from Cygwin [25] utilities to test against the system.

We extracted the CODE or text sections (which represent the executable sections in most files in general) from all the files we examined, and tested the sections against the system.

### 4.2 Results

In our simulation, we constructed the training set and then we tested each file in the test set against the system, and determined the closest file in the training set to the input file. If there were both from the same virus, then it means correct detection, otherwise, the input file cannot be correctly classified. Table 1 shows the samples needed in the training set for each virus and the result of testing each virus replicates against the system.

Space and class thresholds of each class are determined based on the result of testing the known test set. In our test, we chose the maximum values of space and class distances

**Table 1** Results of test set against the system

| Virus | Samples needed | Correctly matched | Percentage, % |
|---|---|---|---|
| G2 | 1 | 250 | 100 |
| NGVCK | 6 | 250 | 100 |
| Zperm | 8 | 250 | 100 |
| MetaPHOR | 15 | 250 | 100 |

among the correctly matched viruses, So that these values correspond to the boundary of the virus class, by which we can classify an unknown input. A point worth noting is that, the threshold values are determined heuristically, that is, it is not necessary to choose the highest values among the correctly matched samples, but a larger value mostly guaranteeing that all other samples from the class will lie within the class boundary can also be chosen. Table 2 shows the threshold of each of the four families in our training set.

When constructing the eigenspace, replicates from the same class are distributed near each other according to the similarities between them. Standard deviation of a group of replicates of the same virus can be a good similarity measure for files from this virus class. Table 3 shows the standard deviation of each class in the training set across each dimension. The values in Table 3 are rounded up to the nearest integer.

As we chose three eigenviruses to construct the eigenspace, we could then plot the viruses in a 3D space and see how they are distributed. Fig. 3 shows virus replicates' sample files in the training set distributed in the 3D eigenspace where each axis in the space is an eigenvirus. Fig. 4 shows the test set samples distributed in the eigenspace.

### 4.3 Testing benign files

In order to test a normal file against the system, we chose 250 binary utilities of Cygwin package [25]. If the projected file in eigenspace has a distance more than the space or class threshold to its nearest class, the file is correctly classified as not belonging to the space. Otherwise, the file is misclassified as one of the virus classes and then a false-positive error is produced. In our test, ten input Cygwin file was misclassified as a virus. After projecting the input files to the eigenspace, 240 files had a distance more than the threshold specified for each class. This means that we had 96% correct identifications of the sample normal file and 4% false positive for the subject samples.

**Table 2** Class and space thresholds for each virus

| Virus | Class threshold | Space threshold |
|---|---|---|
| G2 | 51 | 2984 |
| NGVCK | 262 | 4400 |
| Zperm | 2748 | 14 050 |
| MetaPHOR | 446 | 5805 |

**Table 3** Standard deviation of each virus class across 3D space

| Virus | *X* | *Y* | *Z* |
|---|---|---|---|
| G2 | 0 | 0 | 0 |
| NGVCK | 126 | 243 | 55 |
| Zperm | 1227 | 2217 | 5755 |
| MetaPHOR | 233 | 227 | 35 |

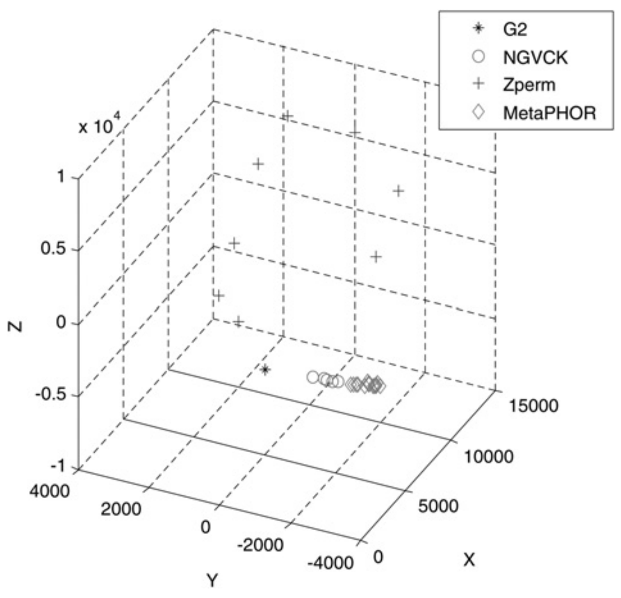


**Fig. 3** *Training set virus replicates in a 3D eigenspace*

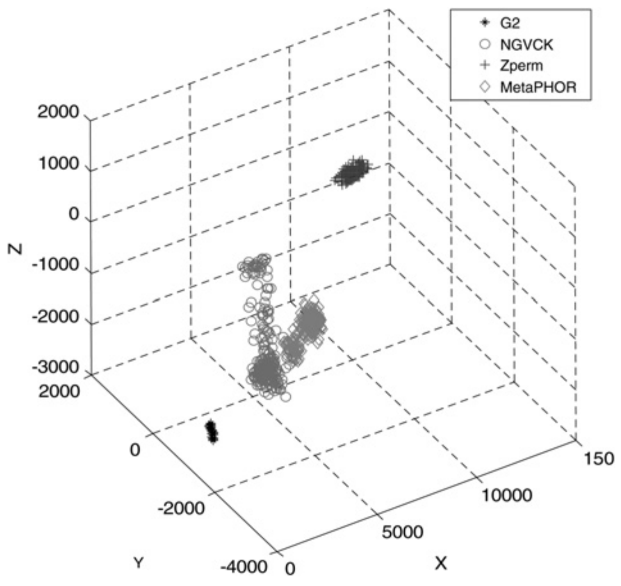


**Fig. 4** *Test set virus replicates in a 3D eigenspace*

## 5 Analysis and evaluation

Space distance is an indicator of how much the input file belongs to the eigenspace. The lower the number, the more feature description is done by the chosen eigenviruses. As Table 2 shows, G2 has the lowest class and space thresholds, which means that its samples are not scattered across the eigenspace, rather it is somehow confined in a small space and the three chosen eigenviruses could describe most of its features, whereas Zperm has the highest threshold because of its high variability and dissimilarities among its replicates.

Standard deviation is also a very good measure of the similarity among replicates of the same virus, as standard deviation measures the dispersion of replicates from their

**Table 4** Testing a random virus replicate from each class against commercial AVs

| | G2 | NGVCK | Zperm | MetaPHOR |
|---|---|---|---|---|
| Symantec | detected | heuristic | heuristic | detected |
| Kaspersky | detected | – | detected | detected |
| ESET | detected | – | detected | detected |

mean point. By examining Table 3 we can note that G2 has zero standard deviation as we needed just one sample, followed by NGVCK and MetaPHOR and finally Zperm. Zperm uses code reordering extensively in a way that is not used by the other three viruses, and that is the reason why its replicates have high variability.

Dispersion of Zperm in the eigenspace with such comparatively high standard deviation can lead to false-positive errors with some benign files, as a projected benign file can lie anywhere in the eigenspace. The less the standard deviation of each virus class in the space, the less likely a false-positive error would occur. Normalisation techniques can greatly help in reducing the variability of the subject input file if it is used to preprocess the file before the test [26]. In case of Zperm, we believe that the use of a depermutator to preprocess the file would greatly reduce its standard deviation.

There is a trade-off between database (training set) size and accurate recognition results. Since the training set represents the knowledge base by which the system can learn about viruses, so the more training data, the more features extracted from them, the more accurate results achieved. However, more space and time complexity arise. The number of replicates of each virus needed for accurate recognition differs according to the virus. For viruses that have high similarity among their replicates, few samples are needed to construct a good model, whereas it is the opposite for hard metamorphic viruses. The size of the initial database of the system can be determined heuristically by increasing the number of replicates in each class to reach acceptable results. In addition, the system can have a continuous learning process. When the system successfully recognises a new input file as belonging to one of the classes in the database, the new file can be incorporated into the database so that more features can be extracted and learned, then more accurate results would be given afterwards.

To give a hint about system performance, the approach was implemented using MATLAB 7.0 R14 and ran on Windows XP SP2 and Intel Dual-core 2.60 GHz processor with 2 GB RAM. It took the system about 21 s to scan the 1000 test files.

In order to compare the results of the system with existing detection systems, we tested a random replicate from each of the four virus classes used in this paper against three well-known antivirus products. Table 4 shows the result of this test [27–30] (DISCLAIMER: The comparison does not reflect the products' general virus detection capabilities or stands as a benchmarking report. It only demonstrates how such a viral replicate can be found by the products.).

The antivirus engines are chosen from the top of antivirus software for the year 2009 according to AV comparative report [31]. The versions used for Symantec, Kaspersky and ESET are 20101.2.0.161, 7.0.0.125 and 5634, respectively.

It can be noted from the results in Table 4 that G2 is easy to be detected so that all the products used in the test succeeded to recognise. On the other hand, NGVCK evaded all the products except for Symantec Norton that recognised the sample as a heuristic virus, that is, the software suspects the file, but it does not recognise its name and in this case, the antivirus product will be unable to repair the infected file. In case of Zperm virus, it was successfully detected by most of the products, whereas Symantec Norton recognised it as 'Bloodhound.W32.1' which is a heuristic type of viruses and so that it cannot be repaired [32]. In case of MetaPHOR, the replicate was detected by all products.

## 6 Conclusions

Metamorphic viruses are the hardest to detect because of their ongoing change in structure while keeping the logical sequence the same on each infection. We developed a system for metamorphic virus recognition based on a statistical machine learning technique. Although when 250 benign files were tested against the system we had 4% false-positive errors, the approach successfully identified 100% of the test set files which consists of 1000 metamorphic virus replicates.

The proposed approach starts with a small training set that contains some replicates of each virus, and then determines the most important features in these replicates. The system represents these features by what is called eigenviruses. Eigenviruses are vectors that span across the most important features in the sample files. By representing these sample files in terms of the eigenviruses, the recognition task is then a pattern recognition problem and can be solved using clustering techniques. For the four sample virus classes we chose in our test, Euclidean distance was used as a distance measure between classes. Although the distance measure technique is very simple, it showed very good results with the chosen test set.

With a high number of virus classes used, Euclidean distance measure may not give good results. Other effective techniques can be used such as using Mahalanobis distance as a distance measure or using support vector machine to cluster the groups of virus class.

In addition, to reduce the potential errors that may occur with larger training set, some malware normalisation techniques can be used, such as using depermutator to help reorganise viruses that use code reordering obfuscation techniques. With such preprocessing for the input test file, we believe that the system will have great performance when used as a product.

## 7 Acknowledgments

M.E. Saleh thanks Peter Szor, the chief antivirus researcher at Symantec Corporation and Peter Ferrie senior antivirus researcher at Microsoft Corporation for their cooperation and responds to the questions regarding techniques used by commercial antivirus software and effectiveness of published experimental detection methods.

## 8 References

1 Symantec Corporation: Internet Security Threat Report, April 2010, vol. XV, p. 21

2 Szor, P.: 'The art of computer virus research and defense' (Addison-Wesley, 2005)

3 Nachenberg, C.: 'Computer virus – coevolution', *Commun. ACM*, 1997, **50**, (1), pp. 46–51

4 Szor, P., Ferrie, P.: 'Hunting for metamorphic'. Virus Bulletin Conf., September 2001, pp. 123–144

5 Jordan, M.: 'Dealing with metamorphism', *Virus Bull.*, 2002, **October**, pp. 4–6

6 Vinod, P.: 'Survey on malware detection methods'. The 3rd Hackers' Workshop, Hack.in, 2009

7 Borello, J.-M., Mé, L.: 'Code obfuscation techniques for metamorphic viruses', *J. Comput. Virol.*, 2008, **4**, (3)

8 Finones, R.G., Fernandez, R.T.: 'Solving the metamorphic puzzle', *Virus Bull.*, 2006, **March**, pp. 14–19

9 Szor, P., Ferrie, P., Perriot, F.: 'Striking similarities', *Virus Bull.*, 2002, **May**, pp. 4–6

10 Desai, P. Department of Computer Science, San Jose State University: 'Towards an undetectable computer virus'. M.S. Report, 2008, p. 10

11 Al Daoud, E., Al-Shbail, A., Al-Smadi, A.: 'Detecting metamorphic viruses by using arbitrary length of control flow graphs and nodes alignment', *UbiCC J.*, 2009, **4**, (3), pp. 628–633

12 Lakhotia, A., Mohammed, M.: 'Imposing order on program statements to assist anti-virus scanners'. Proc. 11th Working Conf. on Reverse Engineering (WCRE'04), 2004, pp. 161–170

13 Ferrie P. Senior Anti-virus Researcher, Microsoft Corporation, USA, Private Communication, April 2010

14 Wong, W., Stamp, M.: 'Hunting for metamorphic engines', *J. Comput. Virol.*, 2006, **2**, (3), pp. 211–219

15 Rabiner, L.R.: 'A tutorial on hidden Markov models and selected applications in speech recognition', *Proc. IEEE*, 1989, **77**, (2), pp. 257–286

16 Durbin, R., Eddy, S.R., Krogh, A., Mitchison, G.: 'Biological sequence analysis: probabilistic models of proteins and nucleic acids' (Cambridge University Press, 1999), p. 46

17 Szor P. Chief antivirus researcher, Symantec Corporation, CA, USA, Private Communication, April 2010

18 Filiol, E., Helenius, M., Zanero, S.: 'Open problems in computer virology', *J. Comput. Virol.*, 2006, **1**, (3–4)

19 Turk, M., Pentland, A.: 'Eigenfaces for recognition', *J. Cogn. Neurosci.*, 1991, **3**, pp. 71–86

20 G2 Virus Generator, Available at http://vx.netlux.org/vx.php?id=tg00

21 Wang, J.: 'Computer network security: theory and practice' (Springer, 2009), p. 287

22 Jakobsson, M., Myers, S.: 'Phishing and countermeasures: understanding the increasing problem of electronic identity theft' (Wiley-Interscience, 2006), p. 114

23 W95.Zperm.A, Symantec Security Response, Available at http://www.symantec.com/security_response/writeup.jsp?docid=2000-121812-0711-99

24 Szor, P.: 'The new 32-bit Medusa', *Virus Bull.*, 2000, **December**, pp. 8–10

25 Cygwin, Available at http://cygwin.com

26 Christodorescu, M., Kinder, J., Jha, S., Katzenbeisser, S., Veith, H.: 'Malware normalization'. Technical Report # 1539, Department of Computer Sciences, University of Wisconsin, Madison, 2005

27 Virus Total, Testing a G2 replicate, Available at http://www.virustotal.com/file-scan/report.html?id=d072f99476dfd3164233701f83c1d435017f09225c6b78d3bd8667399cf6d089-1290261829

28 Virus Total, Testing a NGVCK replicate, Available at http://www.virustotal.com/file-scan/report.html?id=05a7d0d18b6a56e8734b837befe5a99a0e77160f27a96c051b94435801f28c9a-1290261516

29 Virus Total, Testing a Zperm replicate, Available at http://www.virustotal.com/file-scan/report.html?id=123c620d363d3da8e3d4ad873a1068c652be23a3b27952d4bdd439d2803e1f06-1290261264

30 Virus Total, Testing a MetaPHOR replicate, Available at http://www.virustotal.com/file-scan/report.html?id=ec08c95a7511ca10865f21d757bda7e9d61662a56b226a13f885551f070a2e0a-1290260269

31 AV Comparatives: 'Summary Report 2009'. 2009, Available at http://www.av-comparatives.org/images/stories/test/summary/summary2009.pdf

32 Symantec, Bloodhound.W32.1, Available at http://www.symantec.com/security_response/writeup.jsp?docid=2002-070318-2244-99