GENERIC DETECTION AND CLASSIFICATION OF POLYMORPHIC MALWARE USING NEURAL PATTERN RECOGNITION

Rubén Santamarta
ruben@reversemode.com
www.reversemode.com

Dedicated to all who lost their life far away from their families, struggling for an ideal of justice, equality, peace and freedom, with neither classes nor misery.
Thank you for coming, this land will be forever yours.

International Brigades, 1936-2006

Abstract

The obsolete way in which some anti-virus products are generating malware signatures, is provoking that polymorphic malware detection becomes a tedious problem, when actually it is not so hard. This paper describes the basics of a method by which the generic classification of polymorphic malware could be considered as a trivial issue .

1. Introduction

Pattern recognition is continuosly present in our life. In fact, the vast majority of decission-making processes performed by humans is based on some type of object recognition. For example, if you want to buy an apple, you must know how an apple is, distinguishing it from a lemon or an orange. The colour and the shape of the fruit are important features in which your decision will be based on.

The goal of pattern recognition is to clarify these mechanisms, and others more complicated, of decision-making processes and to automate these functions using computers.

The rigth way to do this, requires to develop mathematical models of the objects based on their features or attributes. It also involves operations on abstract representations of what is meant by our common sense idea of similarity or proximity among objects.

Our goal is to develop a system for discriminating between specific polymorphic malware and other types of polymorphic malware or goodware. For this purpose, we will use executables generated by *Morphine* [1], a widely used Polymorphic Packer/Crypter developed by *Holy-Father* and *Ratter*. *HackerDefender* [1] Rootkit developer and a member of the famous virus writers group *29a* [12] respectively.

Morphine should not be considered as malware by itself. However, it cannot be discussed that a lot of malware uses it since Morphine is extrictily developed just to bypass anti-virus engines. Due to this fact, the presence of Morphine files spreading through networks or located on a server, should trigger the alarms of administrator and IDS.

2. First Approach

Previously, we have introduced the concept of object recognition based on features or attributes. Now is the time to extract these features from the objects we want to recognize, in this case, Morphine files.

In theory, polymorphic files will be different from one generation to another. Enough different to bypass static signatures. Nevertheless, we should see this fact with perspective, then asking ourselves

- Are they so different between each one?
- Different yes, but with respect to what?

The second question brings us the key to solve the problem. Yes, executables generated by Morphine are different, since the cipher-key is different, the polymorphic stub is different,..but not so different.

The Morphine stub, beggining at the executable entry point, is generated by the Morphine polymorphic engine. No static bytes can be located so antivirus static-signatures are useless and then, Morphine detection and classification proccess resorts toward emulation engines or “poor and anything but secure” static-signatures based on Import Table(PEiD), StackSize...

Currently, Polymorphism seeing it from the perspective of pattern recognition, may be reduced to Oligomorphism, however under the point of view of some “pseudo-heuristic” methods used by anti-virus products, it will continue forever and ever being Polymorphism.

Forget for a while everything you know about inherent features of polymorphism like register swapping... Erasing from our mind these bit-layer changes and making an exercise of abstraction, we come across instructions, and finally just assembly Mnemonics.

OCR systems extract features from the pixels that conform a character, pixels are then the main image information unit so it seems logic that we should extract features from the Mnemonics, the most important information unit inside executables.

It is the time to begin the development of our own Neural Pattern Recognition System.

3. Pattern acquisition and Feature extraction

In order to obtain an statistical data set from the objects analized, we developed an static disassembler. This tool uses the disassembler algorithm known as *Recursive Traversal [2].*
Before continuing the description of the tool, we have to introduce how we have handled the branches in the execution flow.

Morphine uses the following instructions for branch redirection.

- Call
- Jmp
- Jecxz/Loop (Usually together)

A bunch of conditional jumps are also used, however we do not take care of them since no junk code is generated between conditional jumps and the offset toward it jumps. In addition, significant changes in the flow are always performed by the three previously exposed instructions.

Knowing this, the tool was reforced with an algorithm which follows these branches, also avoiding loops.

The algorithm stops when:

1. Reaches 150 instructions analized
2. Falls in a loop.

The goal of this tool is to collect information about the Mnemonics analized: Frequency, Relative Frequency, Correlation...

```
[0x0040167e]       push    eax
[0x0040167f]       fnop
[0x00401681]       pop     eax
[0x00401682]       mov     ebp,13D
[0x00401687]       xchg    ebp,edx
[0x00401689]       stc
[0x0040168a]       jmp     004016B0
[0x004016b0]       push    -6B0
[0x004016b5]       pop     ecx
[0x004016b6]       push    ebx
[0x004016b7]       call    004016D7
                   -> Branch detected [ 0x004016d7 ]->[ 0x004016b0 ]
[0x004016d7]       pop     ebx
[0x004016d8]       pop     ebx
[0x004016d9]       push    ebp
[0x004016da]       call    00401704
                   -> Branch detected [ 0x00401704 ]->[ 0x004016b0 ]
                   -> Branch detected [ 0x00401704 ]->[ 0x004016d7 ]
[0x00401704]       add     esp,4
```

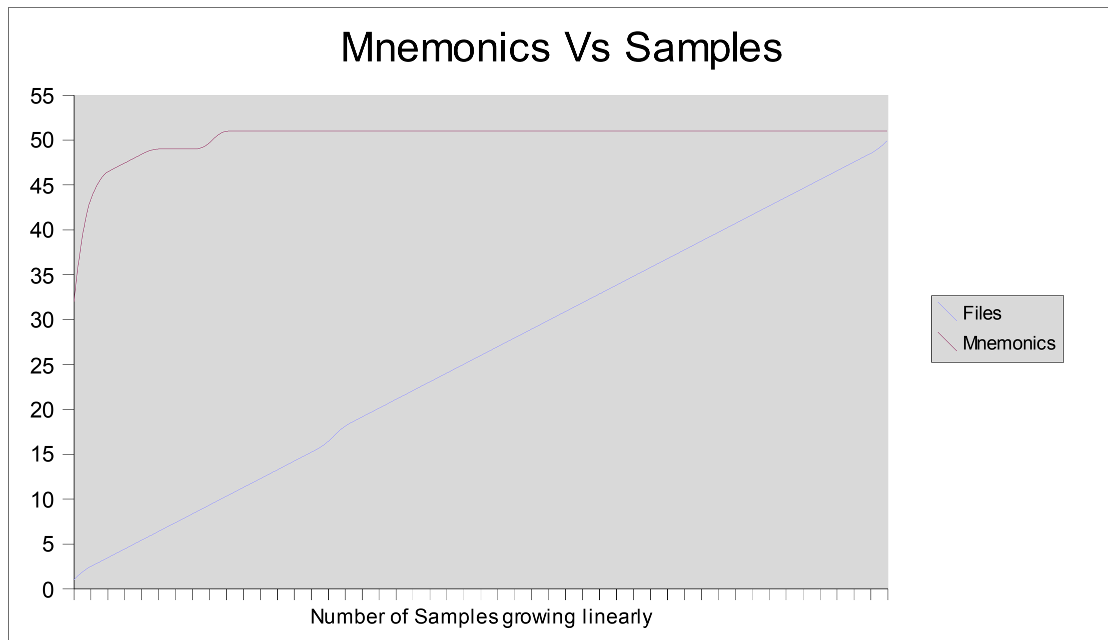
Fig 1.Morphine Static-Disassembler Tool running

Summing up :

1. Find sample Entry Point (Stub beggining)
2. Disassembler following branches and avoiding loops

3. Collect and update information about Mnemonics detected within the sample
4. Analize Next Sample

Plotting the data obtained, we will see how outcomes confirm our previous theory of Polymorphism≅ Oligomorphism



Graph1. Principal Mnemonics detected.

51 different Mnemonics with a significant Frequency were detected. As we can see in the Graph 1, this number does not vary although the number of samples grows.

The Mnemonics detected are the following

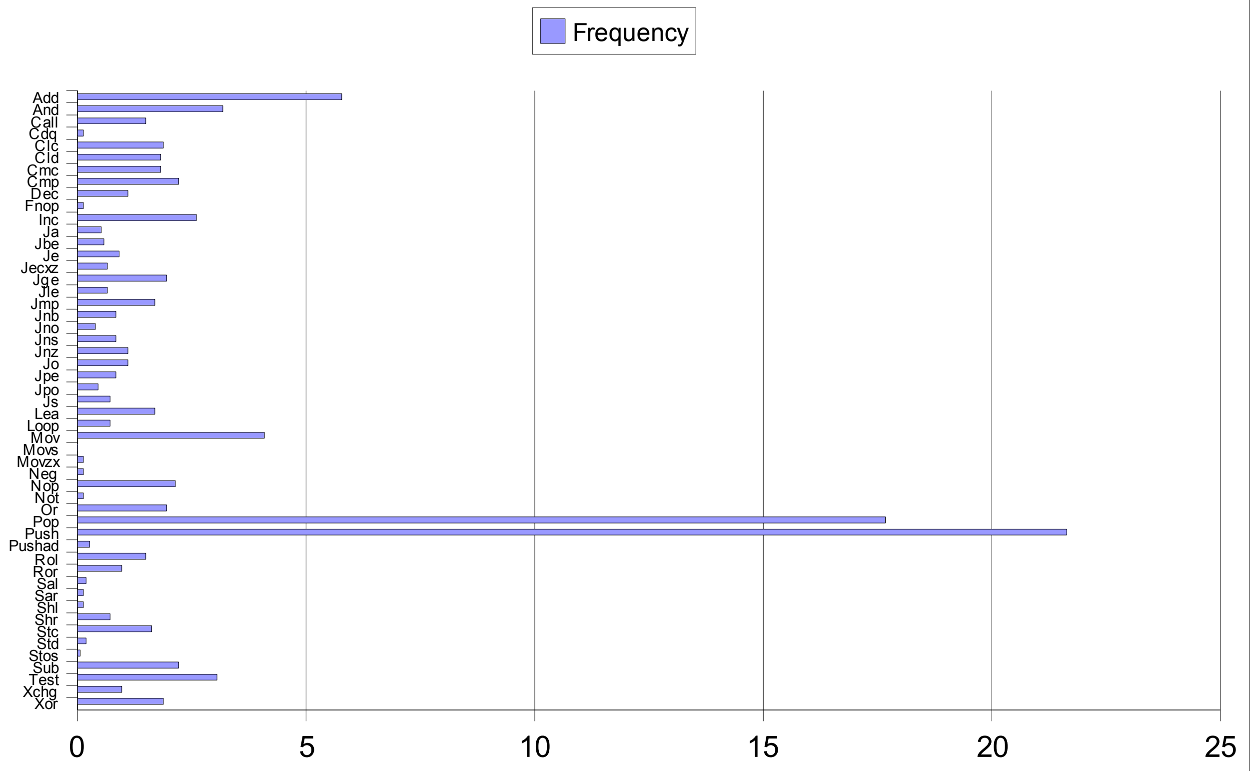
"add"	"and"	"call"	"cdq"	"clc"	"cld"	"cmc"	"cmp"	"dec"	"fnop"	"inc"	"ja"	"jbe"
"je"	"jecxz"	"jge"	"jle"	"jmp"	"jnb"	"jno"	"jns"	"jnz"	"jo"	"jpe"	"jpo"	"js"
"lea"	"loop"	"mov"	"movs"	"movzx"	"neg"	"nop"	"not"	"or"	"pop"	"push"	"pushad"	
"rol"	"ror"	"sal"	"sar"	"shl"	"shr"	"stc"	"std"	"stos"	"sub"	"test"	"xchg"	"xor"

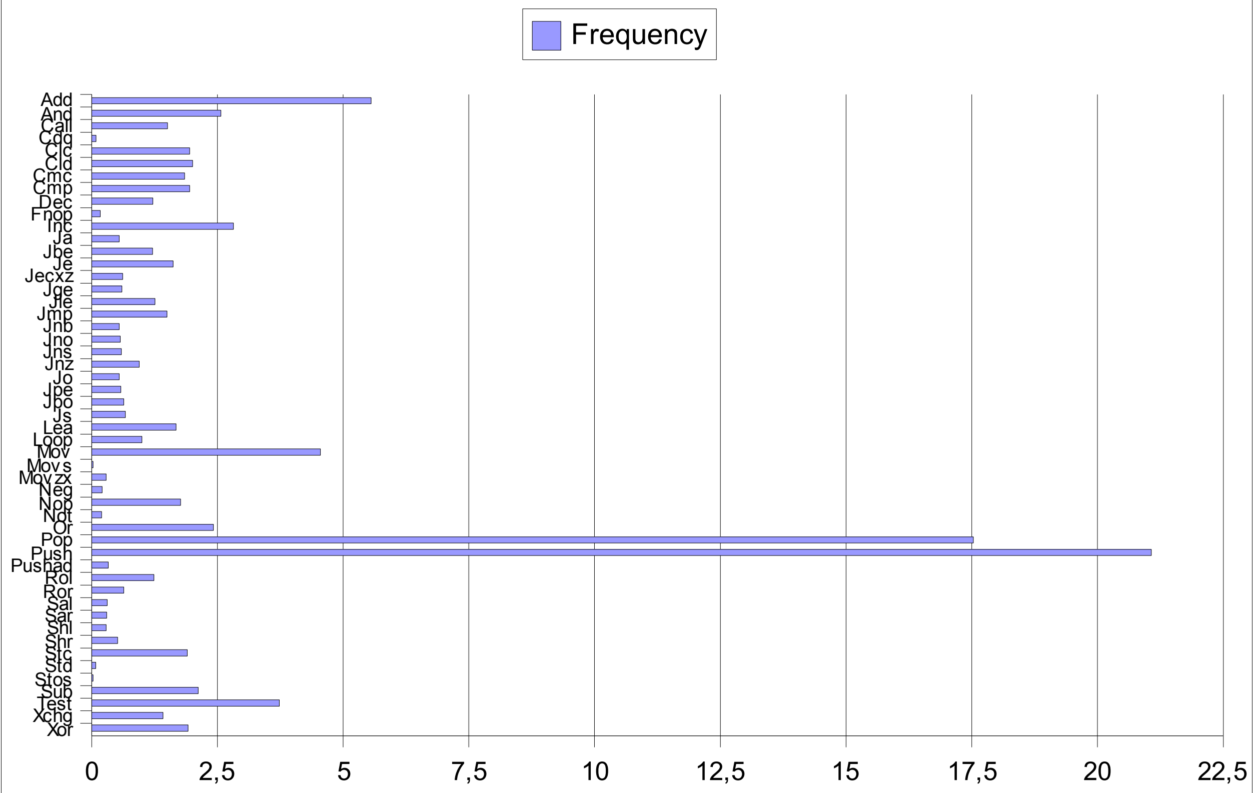
Two curious outcomes were obtained:

1. **Amount and type Mnemonics detected .**
   We cannot forget that we are analyzing 150 instructions as maximum. Goodware, will obtain a much lower amount of Mnemonics analyzing the same number of instructions. In addition, Mnemonics like “Fnop”, “cmc”, “cdq”... are not usually present in the first instructions, beggining at the entry point.

2. **Relative Frequency** [1]
   As we can see in the following histograms, the frequencies obtained are very similar even using a huge threshold between populations.

[1]. The Relative Frequency is calculated over the total number of instructions analized for a given population.


Morphine Polymorphic Engine - Mnemonics Frequencies
Population: 10 Samples
Frequency
Add
And
Call
Cdq
Clc
Cld
Cmc
Cmp
Dec
Fnop
Inc
Ja
Jbe
Je
Jecxz
Jge
Jle
Jmp
Jnb
Jno
Jns
Jnz
Jo
Jpe
Jpo
Js
Lea
Loop
Mov
Movs
Movzx
Neg
Nop
Not
Or
Pop
Push
Pushad
Rol
Ror
Sal
Sar
Shl
Shr
Stc
Std
Stos
Sub
Test
Xchg
Xor
0
5
10
15
20
25


Morphine Polymorphic Engine - Mnemonics Frequencies
Population: 5000 Samples
Frequency
Add
And
Call
Cdq
Clc
Cld
Cmc
Cmp
Dec
Fnop
Inc
Ja
Jbe
Je
Jecxz
Jge
Jle
Jmp
Jnb
Jno
Jns
Jnz
Jo
Jpe
Jpo
Js
Lea
Loop
Mov
Movs
Movzx
Neg
Nop
Not
Or
Pop
Push
Pushad
Rol
Ror
Sal
Sar
Shl
Shr
Stc
Std
Stos
Sub
Test
Xchg
Xor
0
2,5
5
7,5
10
12,5
15
17,5
20
22,5

PCA (Principal Components Analysis) methods are beyond the scope of this paper. However, using the results obtained, we could perform an initial reduction of the dimensional space of the Feature Vector, because of the high correlation encountered between Push-Pop mnemonics and others. I.e this correlation could have been theorically predicted just remembering some habitual features of the polymorphism, I mean pseudo-nop blocks .

1.	Inc esi	2.	Push esi
	Inc edi		Push eax
	Dec Esi		Pop eax
	Dec Edi		Pop esi

4. Design of the Classifier and Pre-Processing

Let us we have defined our n-dimensional Feature Vector as $\mathbf{X}=\{x_1,x_2,\ldots x_n\}$ where n=51, that is, the number of Mnemonics. Thus, each pattern will be represented by its Feature Vector.
Before training our neural classifier, we are going to do a pre-processing into the Feature Vector.
Being F_n the Absolut Frequency of the n-Mnemonic

$$\forall P \in \Omega \begin{cases} F_n > 0 \Rightarrow x_n = 1 : 1 \leq n \leq 51 \, , x_n \in \mathbf{X} \\ F_n = 0 \Rightarrow x_n = -1 : 1 \leq n \leq 51 \, , x_n \in \mathbf{X} \end{cases}$$

We will discriminate between Morphine files and others, so we have two classes: w_1 and w_2.
Our training pattern vectors would be as follows:

$$P=\{(X_1,y_1),(X_2,y_2),\ldots(X_n,y_n)\}$$

Being y the desired output for the n-training pattern. Obviously, the learning will be supervised since we can generate training patterns for both w_1 and w_2 classes.

Why Neural Pattern Recognition ?

MLP are universal aproximators, in addition has been probed to solve problems successful using limited data set. Thus, let us imagine a polymorphic worm spreading through internet, in the early stage of the massive infection, the initial amount of captured samples will be very limited so in order to release as soon as possible an “intelligent signature” which would protect users, neither statistical nor structural pattern recognition could be used.

Designing the Neural Network Topology

The dimension of the Feature Vector will be the number of input neurons in the first layer.
It is hard to calculate the number of hidden units, cascade training algorithm can help us to improve our task, however some authors[3] have proved some empirical rules. One of them is

that the number of hidden units is 10 times the number of output units plus one. More or less, this rule has been successfuly applied to our problem. So, we have a hidden layer with 12 units and finally an output unit because we are facing a binary classification. Both Input and Hidden layer has Bias unit.

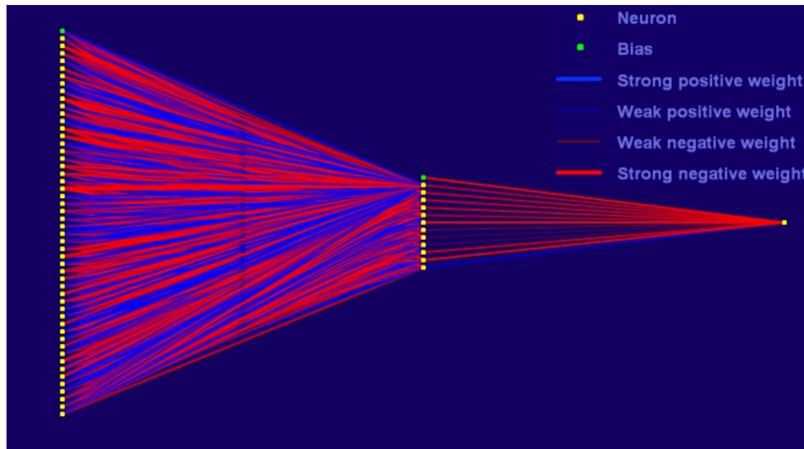


Fig 2. Topology of the neural classifier.

5. Learning algorithm and Activation Functions

Neural Networks are beyond the scope of this paper,however in order to allow the reader to take a full understand of the this part, some tips will be available

BackPropagation is the most widely used learning algorithm, however we will use *Resilient BackPropagation*, *RPROP* [4] from now on, which improves some bad behaviour of the classical BackProp, in addition the convergence will be faster.
RPROP does not use the magnitude of the derivate to update weights, indeed it uses the sign to decide it. In addition, the weights are increased or decreased by two defined factors : η^+ and η^-
Usually :

- $\eta^+ = 1.2$ limited by Δ_{max}
- $\eta^- = 0.5$ limited by Δ_{min}

The RPROP formula for the calculation of weitghs would be as follows:

$$\left(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) > 0 \right) \Rightarrow \left\{ \begin{array}{l} \Delta w_{ij}(t) = \Delta w_{ij}(t-1) + \eta^+ \\ w_{ij}(t) = w_{ij}(t-1) + \Delta w_{ij} \end{array} \right\}$$

$$\left(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) < 0 \right) \Rightarrow \left\{ \begin{array}{l} \Delta w_{ij}(t) = \Delta w_{ij}(t-1) + \eta^- \\ w_{ij}(t) = w_{ij}(t-1) + \Delta w_{ij} \end{array} \right\}$$

$$\left(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) = 0 \right) \Rightarrow \left\{ \begin{array}{l} \Delta w_{ij}(t) = \Delta w_{ij}(t-1) \\ w_{ij}(t) = w_{ij}(t-1) + \Delta w_{ij} \end{array} \right\}$$

Tip

Remembering the basics of derivates and its role in functions analysis. The meaning could be “a measure of a change” and the role, in this case, is to “inform” about the evolution of the Error Function(*E*), which we want to minimize.

In c pseudo-code, the algorithm would be as follows: [4]

```
#define minimum(A,B) ( (A) < (B) ? (A) : (B) )
#define maximum(A,B) ( (A) > (B) ? (A) : (B) )

  sign (A)
  {
       if(A>0 ) return 1
       if(A<0) return -1
       return A
  }
```

For all weights and biases{

$\quad$ **if** $\left(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) > 0\right)$ **then** $\{$

$\qquad \triangle_{ij}(t) = \textbf{minimum}\ (\triangle_{ij}(t-1) * \eta^{+}, \triangle_{max})$

$\qquad \triangle w_{ij}(t) = -\ \textbf{sign}\ \left(\frac{\partial E}{\partial w_{ij}}(t)\right) * \triangle_{ij}(t)$

$\qquad w_{ij}(t+1) = w_{ij}(t) + \triangle w_{ij}(t)$

$\quad \}$

$\quad$ **else if** $\left(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) < 0\right)$ **then** $\{$

$\qquad \triangle_{ij}(t) = \textbf{maximum}\ (\triangle_{ij}(t-1) * \eta^{-}, \triangle_{min})$

$\qquad w_{ij}(t+1) = w_{ij}(t) - \triangle w_{ij}(t-1)$

$\qquad \frac{\partial E}{\partial w_{ij}}(t) = 0$

$\quad \}$

$\quad$ **else if** $\left(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) = 0\right)$ **then** $\{$

$\qquad \triangle w_{ij}(t) = -\ \textbf{sign}\ \left(\frac{\partial E}{\partial w_{ij}}(t)\right) * \triangle_{ij}(t)$

$\qquad w_{ij}(t+1) = w_{ij}(t) + \triangle w_{ij}(t)$

$\quad \}$

$\}$

As activation function, the smoothest, Symmetric Sigmoid for both Hidden and Output Layer. We choose it since output values as well as input are -1 or 1. Sigmoid Symmetric maps $(-\infty,\infty) \rightarrow [-1,1]$

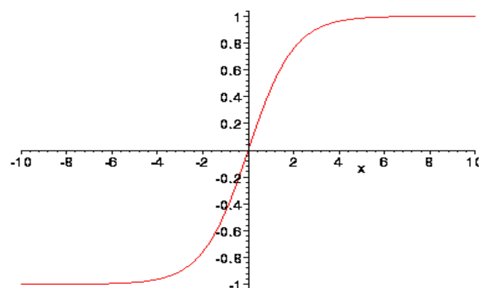


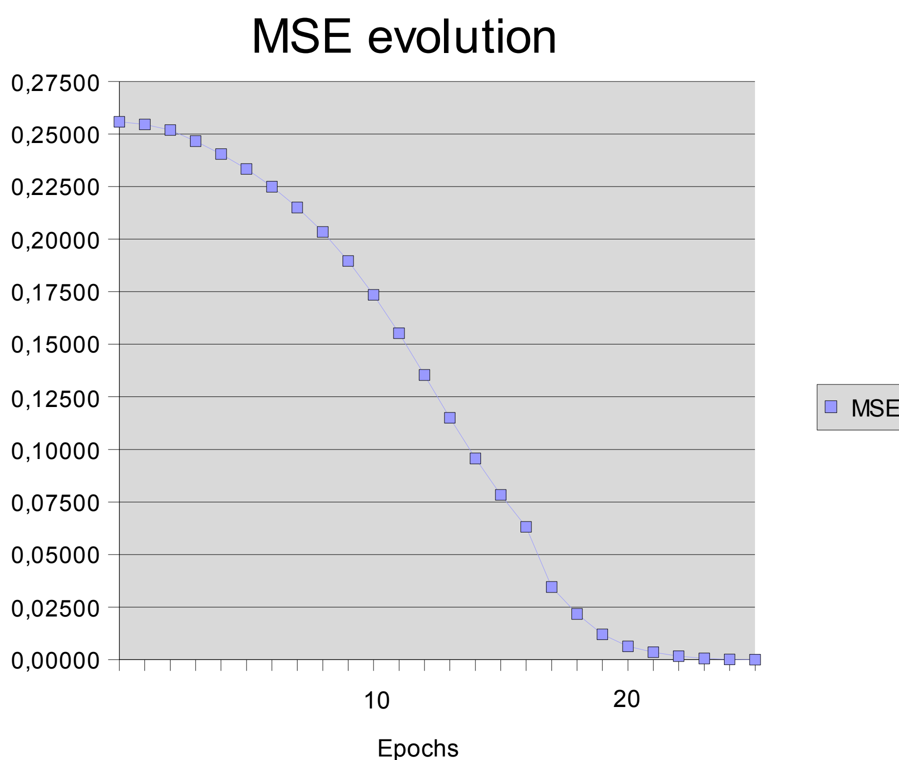
Fig.3 Symmetric Sigmoid Function

A negative output should be considered as a Non-Morphine file while positive output means that the pattern analyzed belongs to Morphine.

6. Training and Experimental Results Obtained

In order to use *Early Stopping [5]* we built up a data set of patterns for training, testing and validation. This method allow us, not only to reach the best learning rate,avoiding overfitting/underfitting, also we can detect poor data set divisions which could lead to overtraining.

The best number of training patterns is about 350 samples, including negative patterns. We got the following results.



Graph 4. MSE Evolution with a Desired Error of 0.0001

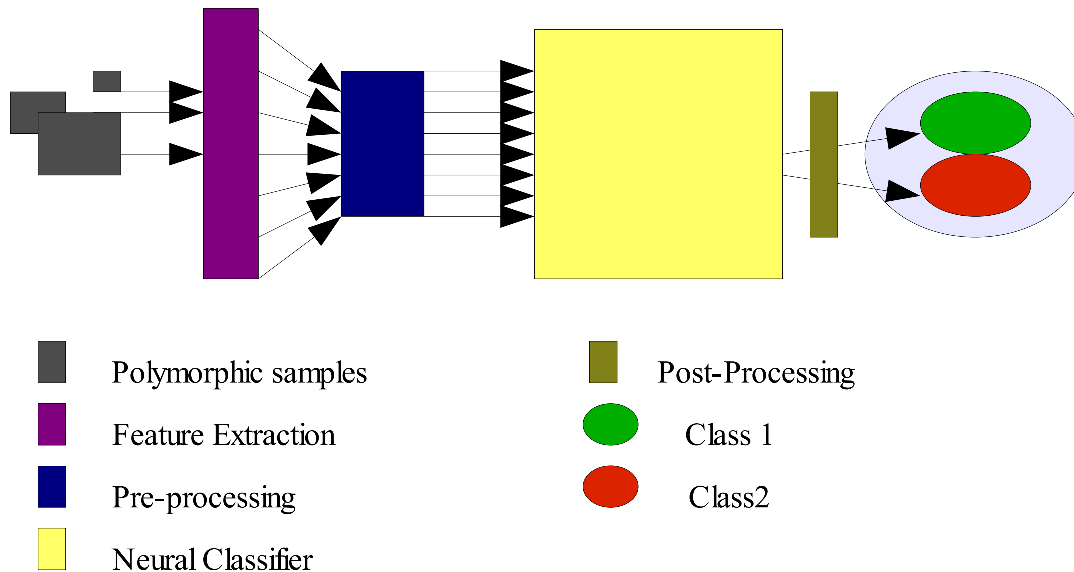
	Training	*Testing*	*Validation*
Samples	350	2000	5000

Samples	*Class*	*Misclassified*	*Error (%)*
5000	Morphine files	3	0,0006
200	Goodware mixed with Upolyx polymorphic samples	0	0

The percent of success   in both two classes is   99,9% and 100% respectively.

7. Conclusion

We should bargain for that only the basics of this method have been explained. However, experimental results obtained show a high level of accuracy. Improving the system with better pre-processing , adding improved feature extraction, extending classes, or developing powerful post-processing would lead to a complete framework resulting in a intelligent infraestructure which could stop polymorphic worms, shellcodes...



This method could be applied in several fields, including IDS, maybe an snort plugin, Antivirus, home-made antivirus ;)

8. References

[1] The Hacker Defender Project
<www.hxdef.org> June 13, 2006

[2] Disassembly Challenges
<http://www.usenix.org/events/usenix03/tech/full_papers/prasad/prasad_html/node5.html>
June 13, 2006

[3] Somaie A.A., Badr A., Salah, T.: Aircraft image recognition using back-propagation. Radar, CIE International Conference on Proceedings, Oct 15-18, 498-501

[4] Riedmiller M., Braun H.: A direct adaptive method for faster backpropagation learning: the RPROP algorithm. Neural Networks, IEEE International Conference on. Vol. 1, 28 March – 1 April (1993) 586-591

[5] Wikipedia: What is Early Stopping?
<http://en.wikipedia.org/wiki/Early_stopping> June 13, 2006

[6] J.P. Marques de Sá: “Pattern Recognition – Concepts, Methods And Applications”
Springer, 2001.

[7] Michael A.Arbib: “The HandBook of Brain Theory and Neural Networks”
The MIT Press, 2003.

[8] Nikola K. Kasabov: “Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering”
The MIT Press, 1998.

[9] Sergios Theodoridis, Konstantinos Koutroumbas: “Pattern Recognition” 2th Edition.
Elsevier, 2003

[10] David J.C. MacKay : Information Theory, Inference, and Learning Algorithms.
2002.

[11] FANN, Fast Artificial Neural Network Library
<http://leenissen.dk/fann/>, June 13, 2006

[12] 29a Labs.
<http://vx.netlux.org/29a/>, June 13, 2006