*The Plymouth Student Scientist, 2018, **11** (1), 223-237*

Binary visualisation for malware detection

Irina Baptista

Project Advisor: Stavros Shiaeles , School of Computing, Electronics and Mathematics, Plymouth University, Drake Circus, Plymouth, PL4 8AA

Abstract

It is becoming increasingly harder to protect devices against security threats; as malware is steadily evolving defence mechanisms are struggling to persevere. This study introduces a concept intended at supporting security systems using Self-Organizing Incremental Neural Network (SOINN) and binary visualization. The system converts a file to its visual representation and sends the data for classification to SOINN. Tests were done to evaluate its performance and obtain an accuracy rate, which rounds the 80% figures at the moment, and false positive and negative rates. Bytes prevalence were also analysed with malware samples having a higher amount of null bytes compared with software samples, which may be a result of hiding malicious data or functionality. The patterns created by the samples were examined; malware samples had more clustering and created different patterns across the images whereas software samples presented mostly static and constant images although exceptions were noted in both categories.

Introduction

With the constant evolution of malware techniques and the increasing number of devices infected the arms race between malware writers and security analysts is escalating. The majority of antimalware mechanisms adopt a signature-based approach (Moser, et al., 2008) and need to be aware of new malware and their source code in order to update its database with the malicious signature. This process relies on manually analysing a binary file, without executing it, allowing the analyst to understand in detail the behaviour of a sample, such as function calls, detect packers, find strings, – for instance if a program accesses a URL then a string will be created for the URL – and observe operations using the disassembled version, however reverse engineering code to obtain the malware signature is extremely time-consuming and challenging for security defenders (Jilcott, 2015), thus the need for automated analysis.

One method to automate the analysis process consists in running the binary in a virtual environment, where its interactions with the system, for e.g. API calls and registry changes, will determine if the program is malicious or not. This process provides a level of security to the system; as the malware is analysed in an isolated environment if any change is made by the binary it will only be reflected in the virtual machine image which can be replaced with a newer one after the analysis. However this automation method, also designated as dynamic analysis, has its shortcomings being them the insufficient information regarding the program's functionality in some scenarios (Conti, et al., 2008), affecting the system performance as it is necessary to run the malware for an extend amount of time with limited resources or even the use of detection routines within the code (Katangur, et al., 2013). These routines detect whether the binary is being executed in a virtual environment, in which case the malicious actions will not be performed.

In addition to that, malware writers are constantly exploring advanced attacking techniques such as polymorphism and metamorphism, and this has been proved successful while evading signature-based systems (Christodorescu & Jha 2003, Moser, et al., 2008). Polymorphic malware constantly mutates its appearance to avoid detection by anti-malware programs; the change may be done by modifying attributes such as filename or randomly generating a new key to encrypt the code but without affecting the functionality of the malicious code. Metamorphic malware is another form of polymorphism with the difference of auto-mutating its code and generating a somewhat different instance of itself with each execution becoming harder to detect but also harder to create.

In face of those issues research is now focusing on machine learning. Yoo, 2004 uses Self-Organizing Maps (SOM) to flag unusual patterns in executables files particular to malware with the pattern position reflecting the position of the malicious code in the file. Chouchane, et al., 2008 base their research on that a good number of malicious programs are variations of previously identified malware and apply Hidden Markov Models to verify if a program may be a variant from some previous software, the framework predicts the effects of morphing actions on a particular property of a program being evaluated. Dahl, et al., 2013 presents a large-scale malware classification system using neural networks; the system utilizes random projections to decrease the input dimensionality. Gavrilut, et al., 2009 proposes a machine learning framework to discern malware and clean files. Xu, et al., 2017 centre their research in behaviour analysis, specifically the fingerprints left on the program memory access, and employ machine learning to classify malicious behaviour. Firdausi, et al., 2010 propose a method to tackle the ineffectiveness of static malware analysis against the rapid spread of malware using automatic analysis and data mining techniques. The solution compares 5 machine learning algorithms with the best performance being achieved by J48 with an accuracy rate of 96.8%.

The purpose of this study is to present and evaluate a malware detection system based on machine learning to address the problems of the current security mechanisms, mainly detection of zero day exploits. The system employs binary visualization and the Self-Organizing Incremental Neural Network (SOINN) to distinguish malware from software. Binary visualization was used in the pre-processing part of the system to identify potential anomalies in a file. Malin, et al. (2012) explain that by visualising the contents of a file it is possible to gather information about the data distribution on that file, identify data obfuscation and group similar files based on their binary patterns. This technique converts the binary values of a file to its two-dimensional representation. However, to maintain the proximity between points the two-dimensional plane was constructed with a Hilbert space-filling curve. Considering a two-dimensional square grid $N \times N$ where $N \equiv 2^n$ and $n \geq 0$, a space-filling curve (SPC) is a continuous curve that passes through N^2 cells in the dimensional space without ever intercepting itself. Some algorithms need to perform calculations between neighbouring points, therefore preserving the data locality becomes essential when traversing the data (Gotsman & Lindenbaum, 1996), and in these contexts SPC are extremely useful. Among them the Hilbert curve is believed to achieve the best locality preservation (Jagadish 1990, Moon, et al. 2001, Lawder & King, 2001) and was used for clustering in this research.

With regards machine learning SOINN, an unsupervised neural network used to classify data without previous knowledge of its network structure (Shen & Hasegawa, 2006), was used as the analysis and detection mechanism. It has no predefined topology and adjusts its structure to account for new input and delete noise becoming a simple yet powerful algorithm which has been adapted and used in various fields, including computer vision, anomaly detection, pattern classification and many others (Qiu, et al., 2016).

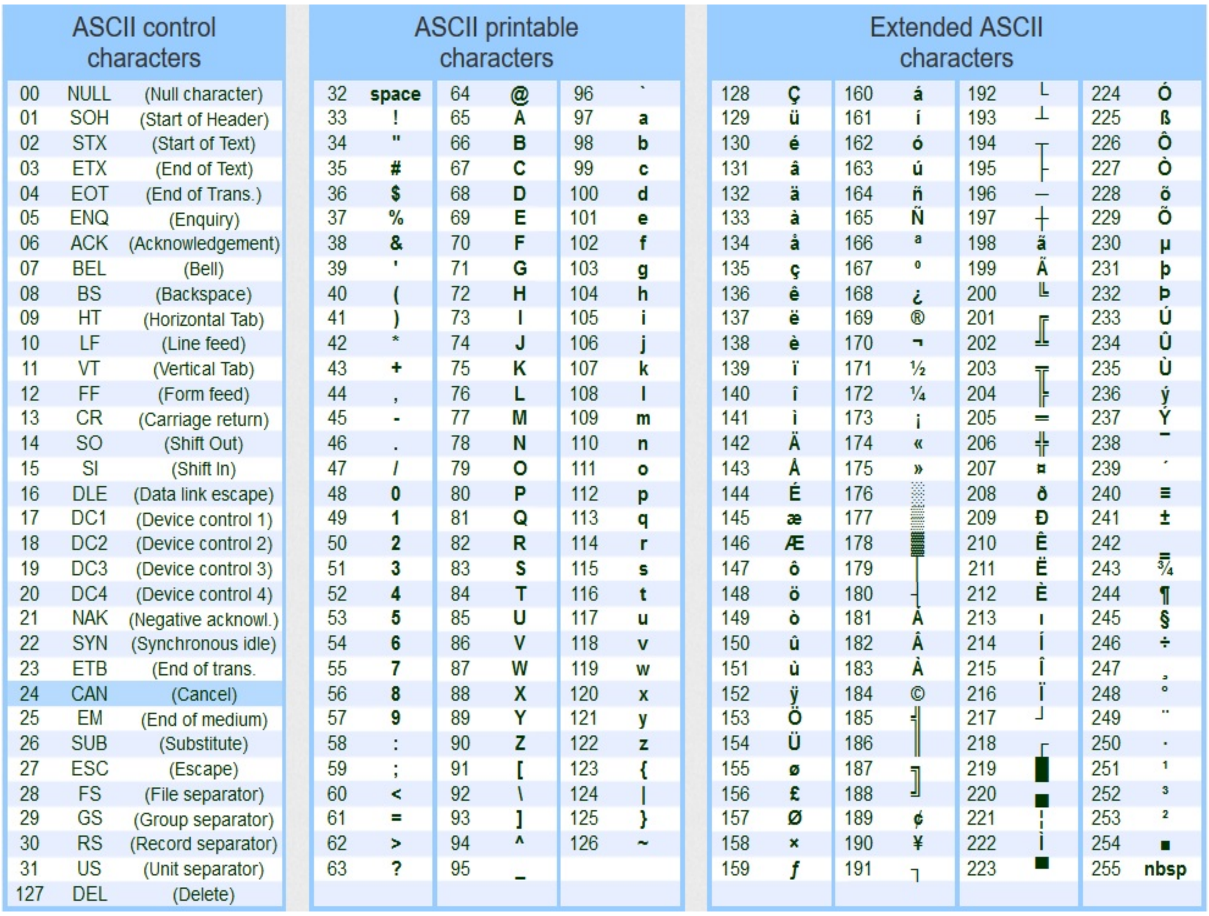
Methodology

Binary file visualisation using Binvis.io

The system was constructed based on the online tool binvis.io (Cortesi, 2016), which uses images and the RBG plot to visualise binaries from files. The files were firstly converted to a string containing their binary code as each character of the string would later be used to create the image. The characters were picked one by one and converted to their appropriate colour. The process of colour selection was done by comparing the values in the ASCII table (figure 1) to their equivalent in the colour scheme provided in binvis.io (Cortesi, 2016) (figure 3).

The values in the colour schemes (figure 3) have been divided in 5 groups:

- Black = 0 or 0x00
- Green = Low bytes
- Blue = ASCII text
- Red = High bytes (Extended ASCII)
- White = 255 or 0xFF



ASCII control characters			ASCII printable characters						Extended ASCII characters							
00	NULL	(Null character)	32	space	64	@	96	`	128	Ç	160	á	192	└	224	Ó
01	SOH	(Start of Header)	33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
02	STX	(Start of Text)	34	"	66	B	98	b	130	é	162	ó	194	┬	226	Ô
03	ETX	(End of Text)	35	#	67	C	99	c	131	â	163	ú	195	├	227	Ò
04	EOT	(End of Trans.)	36	$	68	D	100	d	132	ä	164	ñ	196	─	228	õ
05	ENQ	(Enquiry)	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	Õ
06	ACK	(Acknowledgement)	38	&	70	F	102	f	134	å	166	ª	198	ã	230	µ
07	BEL	(Bell)	39	'	71	G	103	g	135	ç	167	º	199	Ã	231	þ
08	BS	(Backspace)	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Þ
09	HT	(Horizontal Tab)	41	)	73	I	105	i	137	ë	169	®	201	╔	233	Ú
10	LF	(Line feed)	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Û
11	VT	(Vertical Tab)	43	+	75	K	107	k	139	ï	171	½	203	╦	235	Ù
12	FF	(Form feed)	44	,	76	L	108	l	140	î	172	¼	204	╠	236	ý
13	CR	(Carriage return)	45	-	77	M	109	m	141	ì	173	¡	205	═	237	Ý
14	SO	(Shift Out)	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	¯
15	SI	(Shift In)	47	/	79	O	111	o	143	Å	175	»	207	¤	239	´
16	DLE	(Data link escape)	48	0	80	P	112	p	144	É	176	░	208	ð	240	≡
17	DC1	(Device control 1)	49	1	81	Q	113	q	145	æ	177	▒	209	Ð	241	±
18	DC2	(Device control 2)	50	2	82	R	114	r	146	Æ	178	▓	210	Ê	242	‗
19	DC3	(Device control 3)	51	3	83	S	115	s	147	ô	179	│	211	Ë	243	¾
20	DC4	(Device control 4)	52	4	84	T	116	t	148	ö	180	┤	212	È	244	¶
21	NAK	(Negative acknowl.)	53	5	85	U	117	u	149	ò	181	Á	213	ı	245	§
22	SYN	(Synchronous idle)	54	6	86	V	118	v	150	û	182	Â	214	Í	246	÷
23	ETB	(End of trans.	55	7	87	W	119	w	151	ù	183	À	215	Î	247	¸
24	CAN	(Cancel)	56	8	88	X	120	x	152	ÿ	184	©	216	Ï	248	°
25	EM	(End of medium)	57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	¨
26	SUB	(Substitute)	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
27	ESC	(Escape)	59	;	91	[	123	{	155	ø	187	╗	219	█	251	¹
28	FS	(File separator)	60	<	92	\	124	|	156	£	188	╝	220	▄	252	³
29	GS	(Group separator)	61	=	93	]	125	}	157	Ø	189	¢	221	¦	253	²
30	RS	(Record separator)	62	>	94	^	126	~	158	×	190	¥	222	Ì	254	■
31	US	(Unit separator)	63	?	95	_			159	ƒ	191	┐	223	▀	255	nbsp
127	DEL	(Delete)														

Figure 1: ASCII Table *(ASCII, no date)*

With this information, it is possible to assume what data different files may transport, i.e. the resulting image from a text file will mainly be composed by blue pixels whereas a compressed file will mostly likely display random colours.

The data clustering was performed using the Hilbert curve to ensure data that is close together remains grouped once the image processing finishes. Figure 2 presents the same file clustered using Hilbert curves and simple curves.

The byte colour is then draw on the image according to the coordinates given by the Hilbert curve. Since space-filling curves are made of squares, to obtain a rectangular shape the image height is subdivided in four creating squares stacked on top of each other. The squares are represented by their capacity which is the amount of data they can hold. To account for files that are larger than the capacity of the 4 squares previously introduced it is necessary to skip characters along the string. The flowchart used to create a binary image can be seen on figure 4.

The final output is an image representing the features, data type of a file.

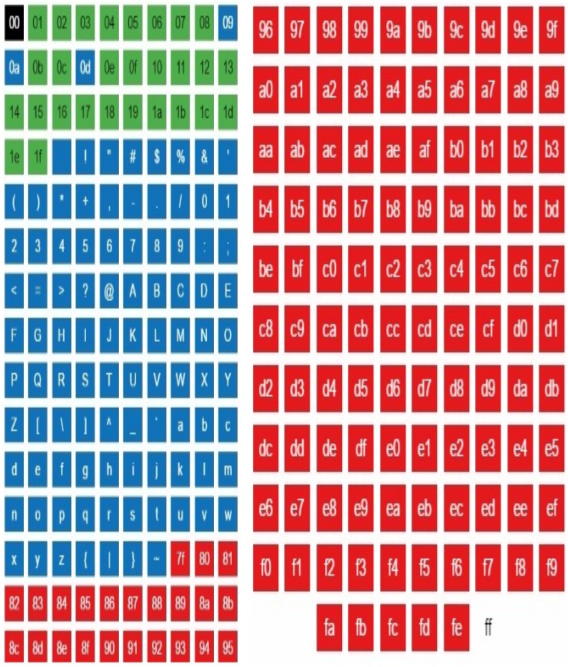

00 01 02 03 04 05 06 07 08 09
0a 0b 0c 0d 0e 0f 10 11 12 13
14 15 16 17 18 19 1a 1b 1c 1d
1e 1f ! " # $ % & '
( ) * + , - . / 0 1
2 3 4 5 6 7 8 9 : ;
< = > ? @ A B C D E
F G H I J K L M N O
P Q R S T U V W X Y
Z [\ ] ^ _ ` a b c
d e f g h i j k l m
n o p q r s t u v w
x y z { | } ~ 7f 80 81
82 83 84 85 86 87 88 89 8a 8b
8c 8d 8e 8f 90 91 92 93 94 95
96 97 98 99 9a 9b 9c 9d 9e 9f
a0 a1 a2 a3 a4 a5 a6 a7 a8 a9
aa ab ac ad ae af b0 b1 b2 b3
b4 b5 b6 b7 b8 b9 ba bb bc bd
be bf c0 c1 c2 c3 c4 c5 c6 c7
c8 c9 ca cb cc cd ce cf d0 d1
d2 d3 d4 d5 d6 d7 d8 d9 da db
dc dd de df e0 e1 e2 e3 e4 e5
e6 e7 e8 e9 ea eb ec ed ee ef
f0 f1 f2 f3 f4 f5 f6 f7 f8 f9
fa fb fc fd fe ff

Figure 3: Binvis.io colour scheme. Tab (09), newline (0a) and carriage return (0d) are considered to be text *(Cortesi, 2016)*.

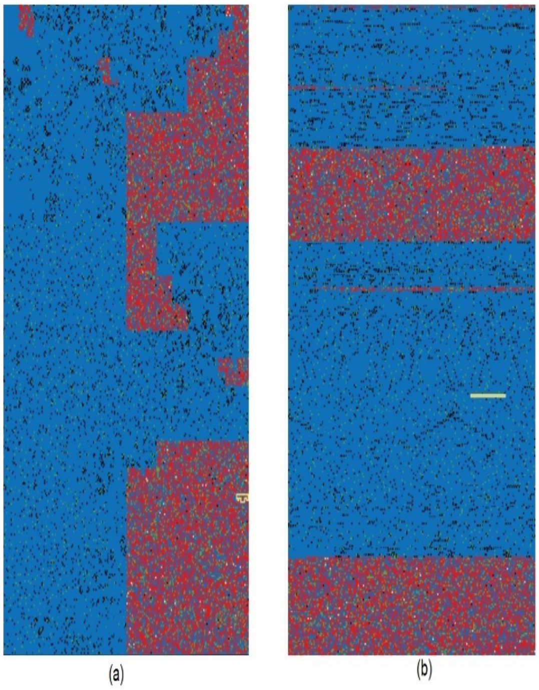


Figure 2: Hilbert curve clustering (a). The second clustering mechanism was the simple scan (b) which is more intuitive when navigating through data. The Hilbert curve on the other hand provides structural features which could remain unseen or difficult to evaluate otherwise.

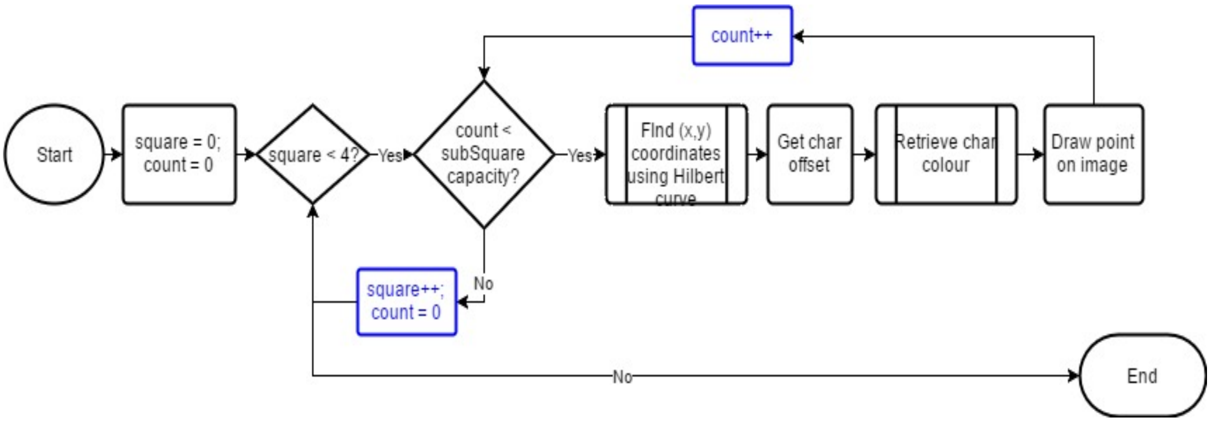


Figure 4: Binvis.io flowchart

Image pre-processing

Before sending the image representation of a file to SOINN it was necessary to normalise the data. Normalisation or pre-processing reduced redundancy, highlighted features and improved performance.

Firstly, it was necessary that all images had the same resolution. Differences on input could influence and decrease the reliability of the algorithm as an image would be transporting more data than another.

The scale or resizing of the images was done using a variation of the nearest neighbour algorithm. For an input sample, the output or class was computed by finding the closest match/sample in the training data. The process was similar when scaling an image. If an image needed to be enlarged, empty spaces were appended to the picture (Tech-Algorithm.com, 2007) and coloured using the nearest neighbour method. The algorithm finds which pixel is nearest to an empty space and fills this space with the pixel colour. Figure 5 illustrates a binary representation of a file before (5a) and after (5b) being resized using the nearest neighbour algorithm.

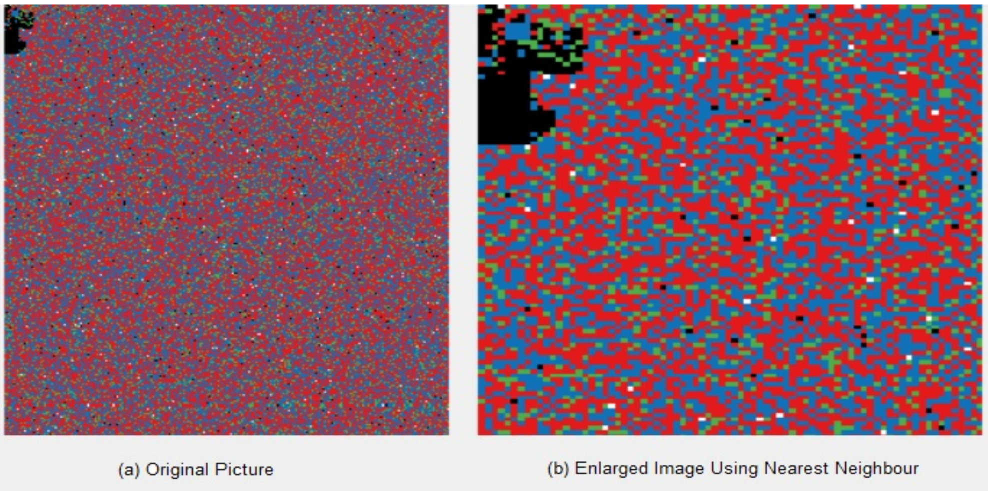


Figure 5: Using the nearest neighbour algorithm to rescale an image

The second part of the pre-processing stage is feature extraction. By sending only the most relevant features to the algorithm it was possible to decrease redundancy and the time necessary to train it increasing the accuracy of the samples, as knowing only individual pixels in the image would not provide sufficient information for SOINN.

To extract those features, the image was divided (Park, et al., 2006) in 4 parts separating the top, bottom and upper and lower middle. The sum of each colour in the RGB space was then calculated and used to create and histogram of colours (seen on figure 6). This highlighted patterns on data and reduced the size (Sun, et al. 2014, Zheng, et al. 2012, Gray & Tao, 2008) of the vector input in SOINN and consequently the time necessary for training.

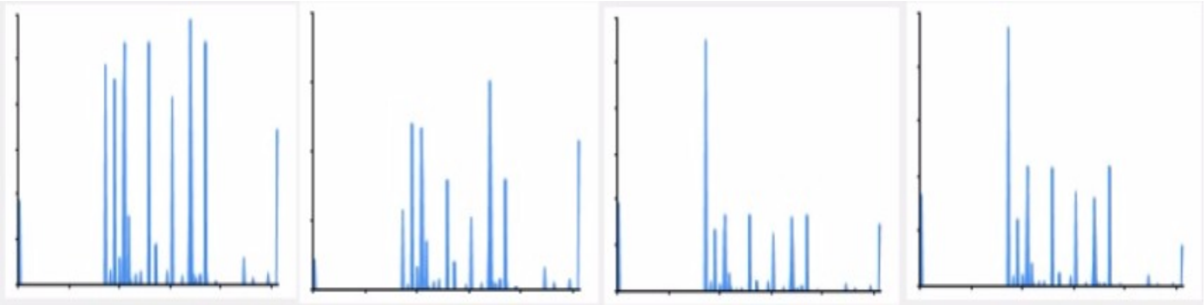


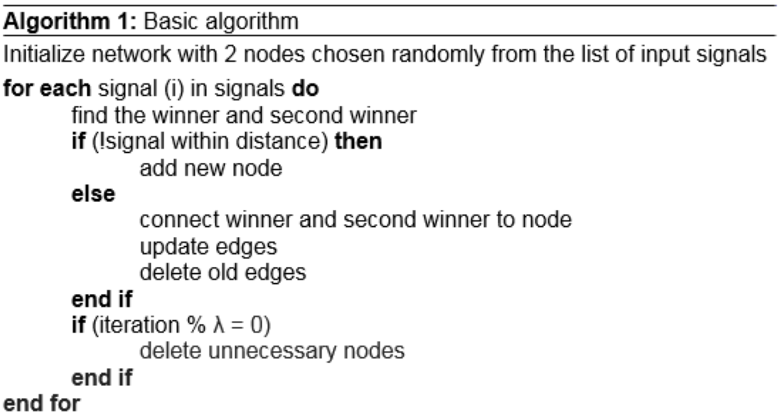
Figure 6: Four histograms depicting an image to account for the top, bottom and middle parts. The X axis represent the RGB space, which is composed by 256 colours, whilst the Y axis the number of times a pixel is portrayed on the image. Each image is then represented by a 1024 feature vector

Self-Organizing Incremental Neural Network (SOINN)

The feature vector obtained in the pre-processing stage is sent to SOINN for network training, which is significantly fast taking less than 2 seconds to compute 200 sample vectors. SOINN is composed from nodes and edges; nodes are vectors once inside the network and the edges represent the connections between nodes, creating the network structure.

Initially, the algorithm receives two random signals (vectors) for input which when converted to nodes become the basis of the network topology. The remaining signals, sent to the algorithm in various iterations, are built around those nodes. Each time a signal arrives to the algorithm the 2 closest nodes referred to as winner and second winner are calculated. If the new signal is within a predefined distance it is connect to the winners, otherwise it will become a new node.

To allow for noise removal, both nodes and edges have a set time in the network in which they need to be removed. However, while the edges are removed without exception, nodes are only removed if considered insignificant to the network, specifically if it has an insufficient number of connections. The following algorithm (figure 7) is based on (Shen & Hasegawa, 2006) explanation of the SOINN network structure.



```
Algorithm 1: Basic algorithm
Initialize network with 2 nodes chosen randomly from the list of input signals
for each signal (i) in signals do
        find the winner and second winner
        if (!signal within distance) then
                add new node
        else
                connect winner and second winner to node
                update edges
                delete old edges
        end if
        if (iteration % λ = 0)
                delete unnecessary nodes
        end if
end for
```

Figure 7: Basic algorithm for the Self-Organizing Incremental Neural Network

Distance calculation

To calculate the distance between two nodes the Euclidean Distance was used, with each node representing a vector of features. The distance is given by calculating the mean value between the input signal and the existing nodes (Lee, et al., 2012).

The distance is achieved with the formula:

$$d_e = \sqrt{\sum_{i}^{n} (n_i - s_i)^2}$$

Where i is the position of the value in vector of features, n indicates the node and s is the input signal.

Network training

A set with 180 samples was used during training. The set contained a mixture of malicious and normal executable files, 78 of which belonged to the later. The malicious data contained 37 trojans, 17 ransomwares, 23 mixed malware types, – backdoor, botnet and others – and 25 unclassified samples. Figure 8 shows the malicious samples distribution on the training set.

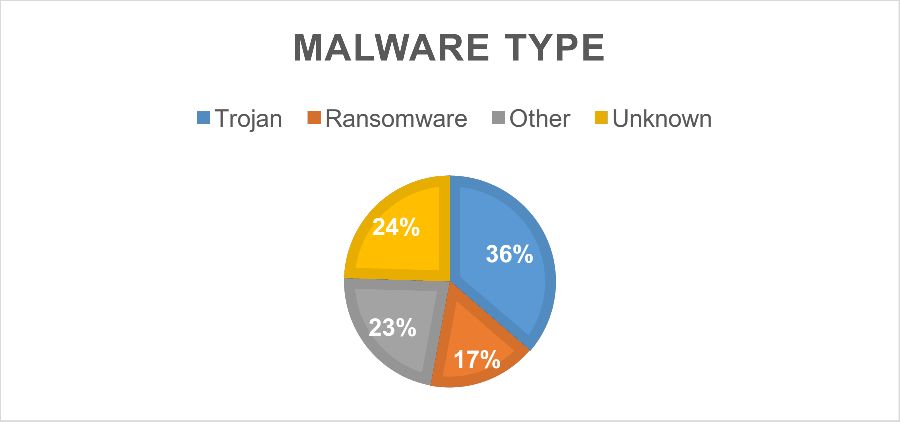


Figure 8: Malware samples distribution according to type

The algorithm was trained for 1000 iterations in a stationary environment (Shen & Hasegawa, 2006) therefore during each iteration a sample was randomly chosen from a list containing the feature vectors from the images and then sent to SOINN.

As it was necessary to keep track of the network accuracy and compare results during testing a label was inserted into each sample. However, the label was for testing purposes only and had no influence on the network overall structure or classification mechanism.

Data filtering

Different files were collected from online and other sources and divided according to their extensions. During training and testing only files possessing an ".exe" extension were used. This was done to limit the scope of the network to a specified group of files.

Influencing variables

Tests were conducted to determine which values would provide higher classification accuracy. Each test was performed 10 times and then the average figures regarding time, accuracy and SOINN output – number of nodes, edges and classes – were calculated.

Image resolution

Two parts of the algorithm are responsible for its accuracy: image manipulation and SOINN structure, therefore the tests were subdivided into two categories respectively. In the first group the width and height of the images were altered and their influence on the network was analysed. The width of the images was increased using the equation 2^x where $x \geq 0$ since the algorithm uses Hilbert curves (Rose 2001, Jagadish 1997, Lawder 1999). On the other hand, the height was simply the width multiplied by 4.

No significant changes in the accuracy rate were discovered by changing the image resolution as seen on figure 9 where the accuracy varies between 81 and 83.

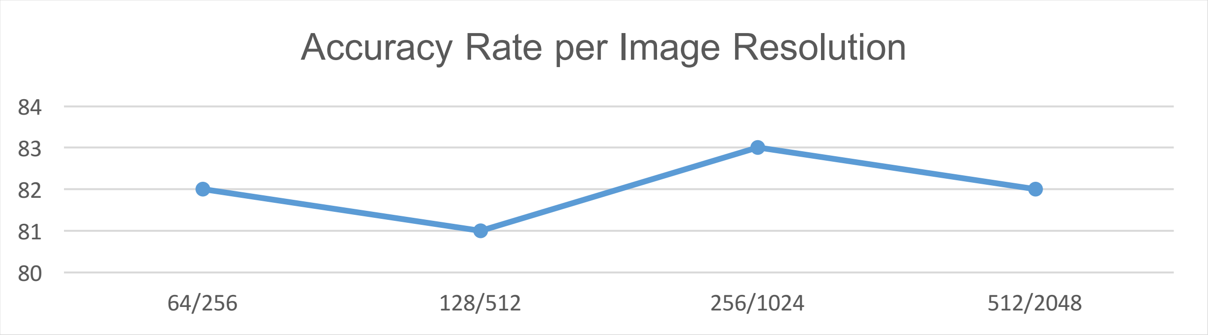


Figure 9: Image resolution influence in the accuracy rate. The (x) axis represents the resolution used in the form width/height. The final values chosen for the image resolution were 256/1024 according to the graph.

The performance, however, was affected by the resolution to a higher degree than the accuracy. Smaller images needed less time to process than larger images (table 1).

Table 1: Effect of the image resolution effect in the algorithm perfomance

Width	Height	Node Age	Edge Age	Processing Time
64	256	300	200	< 5 seconds
128	512	300	200	< 15 seconds
256	1024	300	200	< 45 seconds
512	2048	300	200	≈ 3 minutes

Node and edge removal

Contrarily to the first group, increasing the time to remove a node or edge from the network topology provided better classification results without affecting the system performance.

This time it was possible to see an improvement on the accuracy from the 63% using 50 and 25 for node and edge respectively reaching a peak of about 90% with the values 300 and 200 (figure 10).

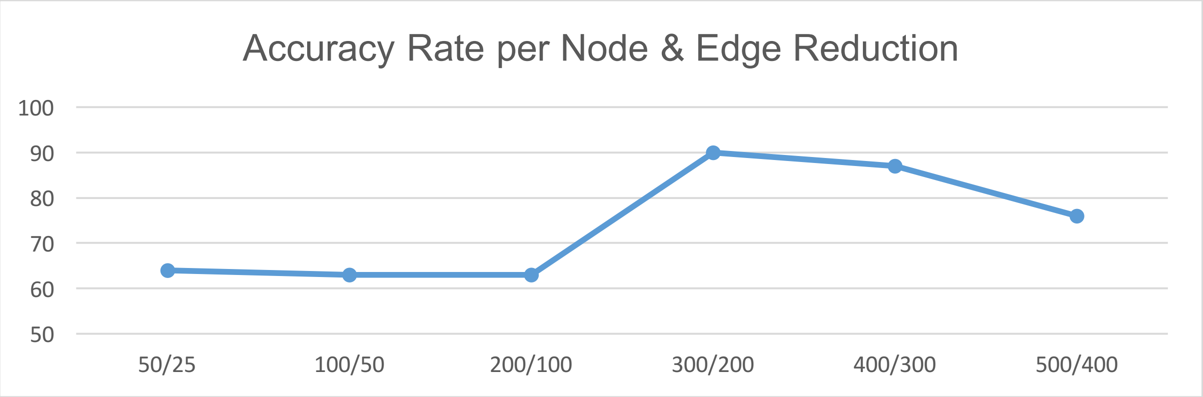


Figure 10: Node and edge age influence in the accuracy rate. The (x) axis represents the specified “time” before removal from the SOINN network in the form node/edge. In the end, the values 300/200 were used.

Computing the result

During testing the nearest neighbour distance of the file being tested is calculated against all nodes in the network (Shen & Hasegawa 2006, Shen, et al., 2007); the node within smaller distance of the test file is the winner. The category of the file, for instance malicious or not, is obtained by returning the class ID of the winner node (Yamasaki, et al., 2010). The flowchart at figure 11 describes the steps taken to classify the file.

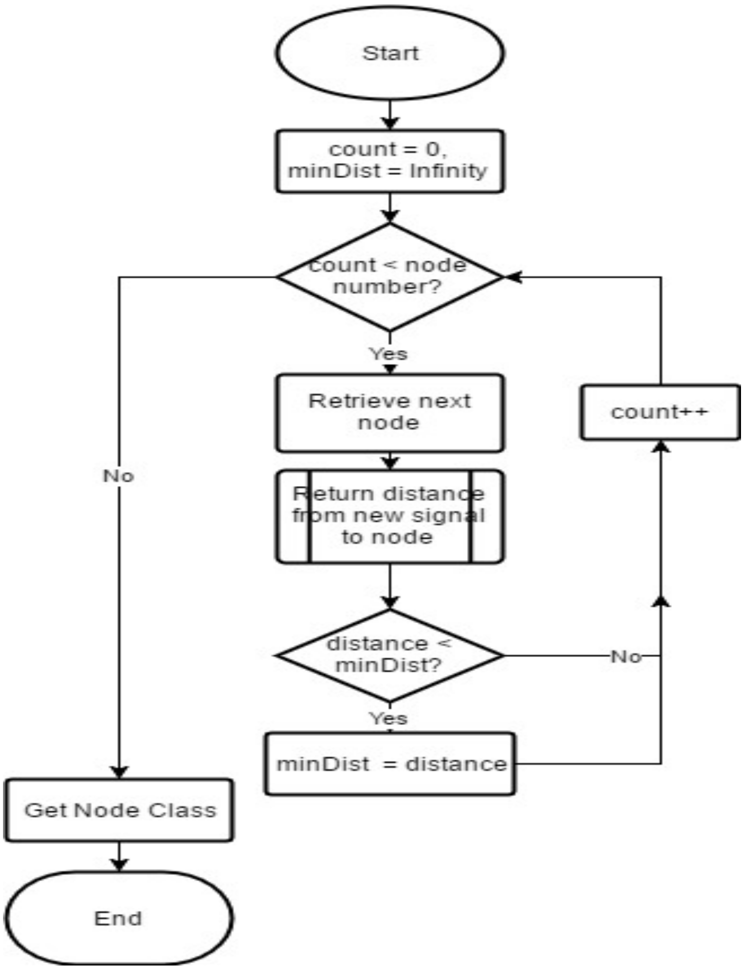


Figure 11: Classification of a file flowchart.

The file category, is returned in the “Get Node Class” call and this value is used to classify the file as whether it is safe or not.

Results

False positive and false negative rate

The accuracy rate varied between 76% and 89% in a set of 10 tests. The algorithm has a higher probability of raising false negatives than positives (figure 12).

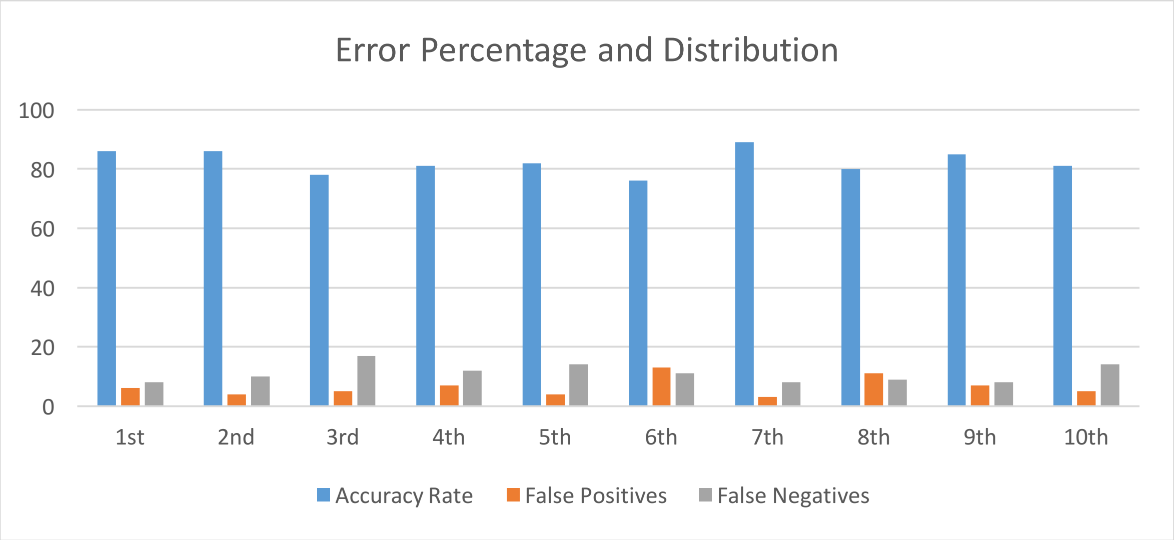


Figure 12: % of false positives and false negatives raised. The (x) axis indicate the test number

Bytes prevalence on malware and software images

Malware samples are prone to have distinct black (null terminator characters) (black areas) and grey areas (control characters) whereas software samples have a more even distribution of colours across the image with a minor proportion of null characters. Software samples also have higher amounts of red pixels (special symbols and characters from the extended ASCII) although more evenly distributed. The relation between colours distribution and classification is seen on figure 13.

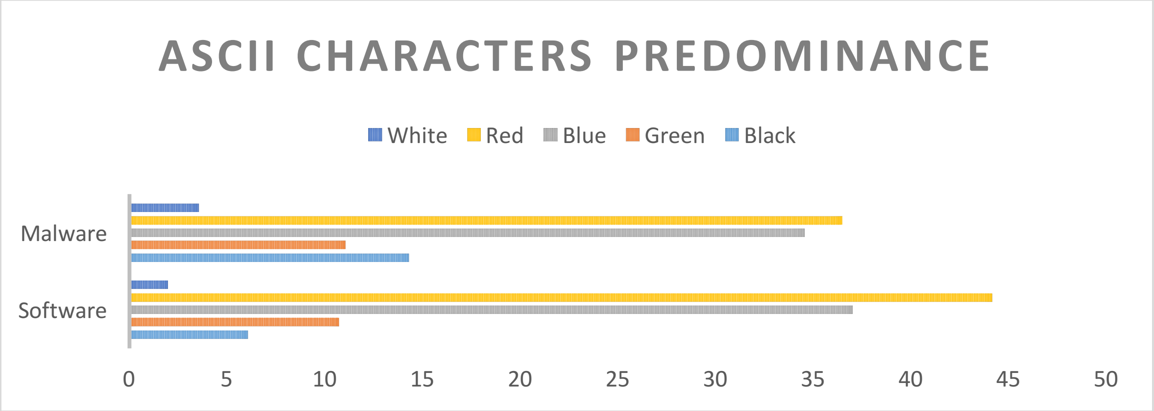


Figure 13: Average colour frequency and influence on the classification between malware and software

Discussion

The study suggests that certain patterns in the binary data of a file can assist in malware detection although similar characteristics exist between it and normal data.

One of the main findings of the research was that malware samples contained a higher amount of null values which was seen by the volume of black pixels in the images. Null values or zero-byte hexadecimals are mostly used to indicate the end of a string (Sikorski & Honig 2012, Ligh, et al. 2011, Huseby 2005), for padding, to hide data in the registry (Ligh, et al., 2011) or instructions (Sovarel, et al., 2005), may indicate the creation of an unknown file (Dahl, et al., 2013) or in a web-related context as a circumvention mechanism (Baloch, 2016) known as null byte injection. Fonseca, et al. (2010) explain that by placing a null byte in between two concatenated strings, the first being a call to a compromised file, an attacker can bypass the concatenation and execute the malicious file. This method is used by hackers to create botnets (Evron, et al., 2007). Ovid (2000) and CAPEC Content Team (2017) demonstrate the use of null bytes to attack and access data. Everett (2006) shows the exploitation of a vulnerability using a null terminator to successfully gain unauthorised access to a server.

Another pattern to note was that some malware samples were mostly covered with green pixels or had a higher occurrence of them when comparing to other samples, this is uncommon with software samples. Green pixels indicate the use (and abuse) of control characters. As it happens with null bytes, attackers make use of control characters to exploit vulnerabilities (Dadkhah, et al., 2014) or hide malicious data (Wressnegger, et al., 2016).

Software samples had a more even distribution of colours across the image with a higher predominance of red followed closely by blue pixels. The patterns recall the image of noise with almost no clustering seen on the picture except for in the top or bottom half of it where the majority of distinctive features occur for this category. There are, of course, exceptions to this rule which may be the cause of false negatives seen in the classification phase. On the other hand, images of malware appear to be more dynamic with different patterns or clusters seen across the samples.

At times the mixture and closeness of pixels with different colours creates the effect of a new colour and this is seen mostly with malware samples were purple, orange and different shades of green were reproduced.

Limitations

The main limitation of the study was the lack of malicious and normal samples and this led to restricting the data to executable files only. Most sources are not willing to provide malware binaries without a cost involved and gathering reliable samples individually from different sources is very time consuming.

A further limitation was regarding the sample details since for some it was not possible to determine the family and type certain malware samples were a part of. Such was the result of collecting samples from different sources as these do not store data in the same detailed format.

Conclusion

The fight against malware has reached a crucial point where the current defence systems have become insufficient; familiar threats represent minor risk to the systems however newly created binaries are harder to detect creating a vulnerability. Thousands of new malicious binaries are registered every day (Bayer, et al., 2010) and malware analysts struggle to handle the analysis of such volume.

This study proposed a new focus for security systems were the detection mechanism would be done with the aid of machine learning. The main findings revealed analysts can have an insight into the malware structure and discover obfuscated code through binary visualisation and this can indeed give them a new perspective to malware, malwares presented a higher amount of null values in comparison to software.

Furthermore, machine learning is efficient in recognising malware, but more importantly it is capable of detecting zero day exploits.

Further investigation is recommended on improving the feature extraction such that it transports increased valuable data to machine learning. Modifying the algorithm to its improved version Enhanced Self-Organizing Incremental Neural Network (ESOINN) would decrease the number of overlapped classes (Shen, et al., 2007) and increase the accuracy rate.

References

ASCII, [no date]. *American Standard Code for Information Interchange.* [Online] Available at: http://www.theasciicode.com.ar/ [Accessed 03 05 2017].

Baloch, R., 2016. *Bypassing Mobile Browser Security for Fun and Profit*.

Bayer, U., Kirda, E. & Kruegel, C., 2010. *Improving the efficiency of dynamic malware analysis.* Sierre, Switzerland.

Chouchane, M. R., Walenstein, A. & Lakhotia, A., 2008. *Using Markov chains to filter machine-morphed variants of malicious programs.* Fairfax, VI, USA, pp. 77-84.

Christodorescu, M. & Jha, S., 2003. *Static Analysis of Executables to Detect Malicious Pattern*.

Conti, G., Dean, E., Sinda, M. & Sangster, B., 2008. *Visual Reverse Engineering of Binary and Data Files.* Heidelberg, Berlin, Germany.

Cortesi, A., 2016. *binvis.io - Vsual Analysis of Binary Files.* [Online] Available at: http://binvis.io/#/

Dadkhah, M., Dadkhah, A. & Deng, J., 2014. Malware Contamined Website Detection by Scanning Page Links. *Information Technology & Electrical Engineering (ITEE) Journal,* 3(6).

Dahl, G. E., Stokes, J. W. & Deng, L., 2013. *Large-scale malware classification using random projections and neural networks.* Vancouver, BC, pp. 3422-3426.

Dahl, G., Stokes, J. & Deng, L., 2013. *Large-scale malware classification using random projections and neural networks.* Vancouver, BC, pp. 3422-3426.

Everett, A., 2006. *Unauthenticated Authentication: Null Bytes and the Affect on Web-based Applications which Use LDAP*.

Evron, G., Damari, K. & Rathaus, N., 2007. *Web server botnets and hosting farms as attack platforms,* Virus Bulletin.

Firdausi, I., Lim, C., Erwin, A. & Nugroho, A. S., 2010. *Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection.* Jakarta, pp. 201-203.

Fonseca, J., Vieira, M. & Madeira, H., 2010. *The Web Attacker Perspective - A Field Study.* San Jose, CA, pp. 299-308.

Gavrilut, D., Cimpoesu, M., Anton, D. & Ciortuz, L., 2009. *Malware Detection Using Machine Learning.* Mragowo, Poland, pp. 735-741.

Gotsman, C. & Lindenbaum, M., 1996. On the Metric Properties of Discrete Space-Filling Curves. *IEEE Transactions on Image Processing,* 5(5), pp. 794-797.

Gray, D. & Tao, H., 2008. *Viewpoint Invariant Pedestrian Recognition with an Ensemble of Localized Features.* Berlin, Heidelberg.

Huseby, S., 2005. *Common Security Problems in the Code of Dynamic Web Applications*.

Jagadish, H., 1997. Analysis of the Hilbert curve for representing two-dimensional space. *Information Processing Letters,* 62(1), pp. 17-22.

Jagadish, H. V., 1990. *Linear Clustering of Objects with Multiple Attributes.* Atlantic City, New Jersey, USA, pp. 332-342.

Jilcott, S., 2015. *Scalable malware forensics using phylogenetic analysis.* Waltham, Massachusetts, Technologies for Homeland Security (HST), 2015 IEEE International Symposium on, pp. 1-6.

Katangur, A., Chaitankar, V., Kar, D. & Akkaladevi, S., 2013. *Dynamic Analysis of Malicious Code and Response.* Athen.

Lawder, J., 1999. *The Application of Space-filling Curves to the Storage and Retreival of Multi-Dimensional Data,* London.

Lawder, J. K. & King, P. J. H., 2001. *Querying multi-dimensional data indexed using the Hilbert space-filling curve.* New York, USA: ACM SIGMOD Record.

Lee, L. H., Wan, C. H., Rajkumar, R. & Isa, D., 2012. An enhanced Support Vector Machine classification framework by using Euclidean distance function for text document categorization. *Applied Intelligence,* 37(1), pp. 80-99.

Ligh, M., Adair, S., Hartstein, B. & Richard, M., 2011. Chapter 10: Malware Forensics. In: *Malware Analyst's Cookbook and DVD.* Indianapolis, Indiana: Wiley Publishing, Inc., pp. 388-393.

Malin, C., Casey, E. & Aquilina, J., 2012. Chapter 6: Analysis of a Malware Specimen. In: *Malware Forensics Field Guide For Windows Systems: Digital Forensics Field Guides.* Massachusetts, USA: Elsevier, pp. 444-446.

Moon, B., Jagadish, H. V., Faloutsos, C. & Saltz, J. H., 2001. Analysis of the clustering properties of the Hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering ,* pp. 124-141.

Moser, A., Kruegel, C. & Kirda, E., 2008. *Limits of Static Analysis for Malware Detection.* IEEE Computer Society.

Ovid, 2000. *CGI Security and the null byte problem*.

Park, U. et al., 2006. *ViSE: Visual Search Engine Using Multiple Networked Cameras,* pp. 1204-1207.

Qiu, T., Shen, F. & Zhao, J., 2016. Review of Self-Organizing Incremental Neural Network. *Journal of Software,* 27(9), pp. 2230-2247.

Rose, N., 2001. *Hilbert-type space-filling curves*.

Shen, F. & Hasegawa, O., 2006. An Incremental Network for On-line Unsupervised Classification and Topology Learning. *Neural Networks,* 19(1), pp. 90-106.

Shen, F., Ogura, T. & Hasegawa, O., 2007. An enhanced self-organizing incremental neural network for online unsupervised learning. *Neural Networks,* 20(8), pp. 893-903.

Sikorski, M. & Honig, A., 2012. Chapter 1: Basic Static Techniques. In: *Practical Malware Analysis The Hands-On Guide to Dissecting Malicious Software.* San Francisco: no starch press, pp. 9-29.

Sovarel, A., Evans, D. & Paul, N., 2005. *Where's the FEEB? The Effectiveness of Instruction Set Randomization.* Baltimore, pp. 10-10.

Sun, Y., Liu, H. & Sun, Q., 2014. *Online learning on incremental distance metric for person re-identification.* Bali, s.n., pp. 1421-1426.

Team, C. C., 2017. *CAPEC-52: Embedding NULL Bytes*.

Tech-Algorithm.com, 2007. *Nearest Neighbor Image Scaling.* [Online]
Available at: http://tech-algorithm.com/articles/nearest-neighbor-image-scaling/
[Accessed 05 05 2017].

Wressnegger, C., Freeman, K., Yamaguchi, F. & K., R., 2016. *From Malware Signatures to Anti-Virus Assisted Attacks,* Braunschweig, Germany.

Xu, Z., Ray, S., Subramanyan, P. & Malik, S., 2017. *Malware detection using machine learning based analysis of virtual memory access patterns.* Lausanne, Switzerland.

Yamasaki, K., Makibuchi, N., Shen, F. & Hasegawa, O., 2010. *How to use the SOINN software: User's guide (version 1.0).* Berlin, Heidelberg.

Yoo, I., 2004. *Visualizing Windows Executable Viruses using Self-Organizing Maps.* Washington DC, USA, pp. 82-89.

Zheng, W., Gong, S. & Xiang, T., 2012. Reidentification by Relative Distance Comparison. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 35(3), pp. 653-668.