# Application of Artificial Intelligence for Detecting Derived Viruses

Omotayo F. Asiru[1], Moses T. Dlamini[1,2] and Jonathan M. Blackledge[1,3,4]
[1]Discipline of Computer Science, School of Mathematics, Statistics & Computer Science, College of Agriculture, Engineering & Science, University of Kwazulu-Natal, Durban, South Africa.
[2]Council of Scientific and Industrial Research (CSIR), Defence, Peace, Safety and Security (DPSS), CCIW, Pretoria, South Africa.
[3] Ministry of Defence, Military Technological College, Muscat, Sultanate of Oman.
[4]Dublin Institute of Technology, Kevin Street, Dublin 8, Ireland.

[1]tayoasi@yahoo.com
[2]TDlamini1@csir.co.za
[3]jonathan.blackledge59@gmail.com

**Abstract:** Computer viruses have become complex and operates in a stealth mode to avoid detection. New viruses are argued to be created each and every day. However, most of these supposedly 'new' viruses are not completely new. Most of the supposedly 'new' viruses are not necessarily created from scratch with completely new (something novel that has never been seen before) mechanisms. For example, most of these viruses just change their form and signatures to avoid detection. But their operation and the way they infect files and systems is still the same. Hence, such viruses cannot be argued to be new. In this paper, the authors refer to such viruses as derived viruses. Just like new viruses, derived viruses are hard to detect with current scanning-detection methods. Therefore, this paper proposes a virus detection system that detects derived viruses better than existing methods. The proposed system integrates a mutating engine together with neural network to improve the detection rate of derived viruses. Experimental results show that the proposed model can detect derived viruses with an average accuracy detection rate of 80% (this include 91% success rate on first generation, 83% success rate on second generation and 65% success rate on third generation). The results further shows that the correlation between the original virus signature and its derivatives decreases further down along its generations.

**Keywords:** Artificial Neural Network, Computer Virus, Mutating Engine, Derived Virus.

## 1. Introduction

A computer virus can simply be defined as a program that infects other programs or systems by modifying them for malicious purposes (Konstantinou, 2008). Computer viruses comes with a wide variety of nefarious activities that ranges from consuming excess of memory space; to showing some funny and peculiar actions; up to performing serious temporary or permanent damage to a computing system (Filiol, 2005). Criminals are now using viruses to hijack and take over computers from unsuspecting users. Hijacked computers are then used to create botnets to spread spam, perform a distributed denial of service and steal identities. Therefore, the effects of computer viruses have moved from just showing an annoying messages on the screen toward having an impact on company reputation and affecting business bottom line.

There is a wide variety of anti-virus programs in existence nowadays which makes an attempt to solve the prevalent virus problem in computing systems. For example, McAfee, Symantec's Norton, Kaspersky are just a few of the market leaders in the anti-virus industry. In spite of all available anti-virus programs in the market today, virus attacks do not seem to slow down. Kaspersky labs antivirus solutions reported a total of 69,277,289 unique malicious viruses in the year 2016 (Kaspersky Lab, 2016). This was against 4,000,000 that were reported by the same product in the year 2015 (Ivanov et al., 2015). This statistics gives 1631.9% increase within a year i.e. between 2015 and 2016. Furthermore, Symantec's internet security threat report shows a 125% increase in the number of zero-day vulnerabilities between 2014 and 2015.

New viruses are being created each day (Daoud, 2009). However, this paper argues that most of these viruses that Daoud (2009) refers to are not completely new. These supposedly "new" viruses are not necessarily created from scratch with completely new (something novel that has never been seen before) mechanisms. A number of these just change their forms and come up with 'new' signatures to avoid detection (Feng & Gupta, 2009).

Hence, such viruses cannot be argued to be completely 'new'. Examples of such are polymorphic and metamorphic viruses. This paper refers to them as derived viruses. This is mainly because the different variants of virus signatures are normally derived from the original signatures.

Just like new viruses, derived viruses are hard to detect with current virus scanning-detection method. This is because current virus scanning-detection methods rely mainly on known signatures to detect viruses. However, virus signatures change rapidly to avoid detection (Silverman, 2001). The main difference between a normal virus and a derived virus is that the latter changes its components to look like a new and different program before it can multiply. This self-modification ability that modern day viruses possess has put a strain on the current virus detection methods (Kumar, 2016).

Nowadays, some virus writers are creating mutation engine for other virus writers to make use of. An example of such is a tool kit called “Dark Avenger's Mutation Engine". This tool allows a virus writer who has a normal virus to use the engine with his virus code, making each infected file appear like a totally different virus code. This is done in such a manner that taking any two files infected by the same virus, their signatures will be totally different and without any correlation.

A derived virus as used in this paper can be defined as any virus that uses any of the modern obfuscation techniques to change its form and how it looks. Viruses normally do this in order to produce different variants that actually look like a new virus. However, most of these exhibit exactly the same behaviour as with the original virus. The obfuscation techniques used by many modern viruses include but not limited to junk or dead code insertion, variable substitution, instruction replacement, instruction re-arrangement and instruction transposition (Rad et al., 2012).

An example of derived virus is a polymorphic virus. A polymorphic virus is a piece of code that is characterized by the following behaviour; encryption, self-multiplication and changing of one or more components of itself so that it remains undetected (Kumar, 2016). A polymorphic virus is not worse than a normal virus in terms of the damage it causes. However, this type of a virus is hard to detect (Silverman, 2001). For example, an original signature of a particular polymorphic virus might be “881600808826000dcd13cd19". This virus signature can produce a variant or derivative by adding a NOP instruction. A NOP (short for No Operation) instruction is an assembly language command or instruction that does nothing. The NOP instruction is used to allow future modification of code without rewriting or recompiling it (Clements, 2014). By adding a NOP instruction to the above polymorphic virus code, the new signature becomes "88160080908826000d9090cd13909090cd19. This same virus can further produce another supposedly 'new' virus by adding a JUMP instruction. The JUMP instruction transfers program execution flow from the current location to a new location (Kjell, 2015). By adding a JUMP instruction to the above polymorphic virus code, the new signature becomes “eb10908826000deb0f9090cd13909090cd198816008088ffebed" (Silverman, 2001). This now look very different from the original signature. Hence, such a virus cannot be easily detected by the current detection technique. Furthermore, the two variants cannot be argued to be new viruses.

Signature scanning is the oldest and most common virus detection technique used by many anti-virus programs today. It has so many disadvantages. For example, it has a poor ability to detect derived and new viruses (Golovko & Bezobrazov, 2015; Hamza & Hussain, 2014). Signature scanning anti-virus works by comparing the signatures of files on a computer system against signatures of viruses stored in the anti-virus database (Mishra, 2010). If the signature of any file matches a signature in the installed anti-virus database, such a file is declared infected and necessary action is taken whether to delete or quarantine it (Kakad et al., 2014). This method of virus detection is effective and gives accurate result but only to known and normal viruses. Another major drawback of signature scanning is the time lag between virus creation and detection. Other virus detection techniques include but not limited to integrity checker, CPU emulator, heuristic scanner and anomaly based detection. Most of the listed virus detection techniques can detect any type of virus but still have high false positive (Mishra, 2010).

Hence, this work aims to add to the body of knowledge by helping improve the rate of detection of derived viruses. The rest of the paper is structured as follows. Section 2 describes related work and section 3 presents and discusses the proposed model. In section 4, some of the results of the experiments are presented and discussed. Section 5 concludes the paper and points out future work.

## 2. Related work

This section reviews existing research efforts that are directed at detecting viruses, more so the derived ones. Several researchers have made plausible attempts to try and accurately detect derived viruses. For example the work of Chen et al. (2012). Chen et al. (2012) proposes a neural network ensemble method. The authors therein (Chen et al., 2012) discuss how machine learning could be used for virus detection. The characteristics of the virus are first extracted for classification and learning through the system calling sequence of a program. Different machine learning approaches are then used to construct an ensemble. The results of the machine learning are combined according to Dempster Shafers evidence theory to form the final output of the system. This research work only discusses how an artificial neural network can be trained. However, it does not show how the output of the trained artificial neural network detects a computer virus. An astounding 97% accuracy was reported for this work when compared with other traditional ways. It must be mentioned though that Chen et al. (2012) did not specify if this include derived or new viruses.

Vinod et al. (2012) proposes a method of detecting variants of metamorphic malware using bioinformatics techniques that are normally used for Protein and DNA matching. In their proposed method, a multiple sequence alignment is used to arrange an opcode sequence of malware. This is to determine similarity among malware samples and also to determine frequent occurring pattern in a malware family. The frequent pattern depicts maliciousness. It is reported that the result of the experiments is comparable with some of the anti-virus software that are commonly in use today.

A memory-based abstraction approach to handle obfuscation in polymorphic virus is proposed by Nguyen et al. (2012). The aim of this research work is to handle obfuscation in polymorphic virus. The methodology involves abstracting the binary code based on their memory states and analyzes the abstracted states to detect useless instructions. This method is proposed because in spite of any obfuscation technique employed, the actual malicious code, once executed produces the same memory state patterns when properly abstracted. However, the major challenge of this approach is the need to develop an abstract form that captures the memory states affected by each instruction when executed. As such, it is stated that the approach is not practically possible because of high complexity.

Kamarudin et al. (2013) conducted an analysis on the effectiveness of signature based anti-virus software in detecting metamorphic viruses. In their work, Kamarudin et al. (2013) created a seed virus using a virus generator. Thereafter, the seed virus is run on their morphine engine which contains some code obfuscation techniques. This process is done to generate a family of metamorphic variants for the seed virus and it was conducted 20 times which gives 20 metamorphic variants of the seed virus with each generated variant different from one another. This explains the reason why detection of metamorphic virus is difficult with signature based anti-virus software. Kamarudin et al. (2013) also performed a similarity test on 4 generations of metamorphic variants and the original seed. It was discovered that the similarity decreases with higher generations. The first generation virus is similar to the virus seed with about 60 percent and the fourth generation virus has an average similarity of 19.7 percent with the virus seed.

Gang & Zhongquan (2014) developed a malicious code detection solution that is based on fuzzy reasoning. This work adopts a comprehensive scheme to get behaviours in such phases as file structure analysis and function calls identification. It also analyses the most common obfuscation technique that insert data after call instruction and provide an algorithm that identifies information calls. A dynamic fuzzy neural network decision system is then used to determine mutating and unknown malicious executable viruses. The system does not show how viruses are detected. However, 95.2% detection accuracy was reported for this work's experiment.

A model to classify files into malware or benign was developed by (Kuriakose & Vinod, 2014). Kuriakose & Vinod (2014) use feature selection methods such as term frequency, inverse document frequency among others to develop the model. Malware and benign portable executables are unpacked and disassembled to get bi-gram

data set. The bi-gram dataset is used for training and testing. The experiment shows that the model can detect more synthetic metamorphic viruses with average detection rate of 92%.

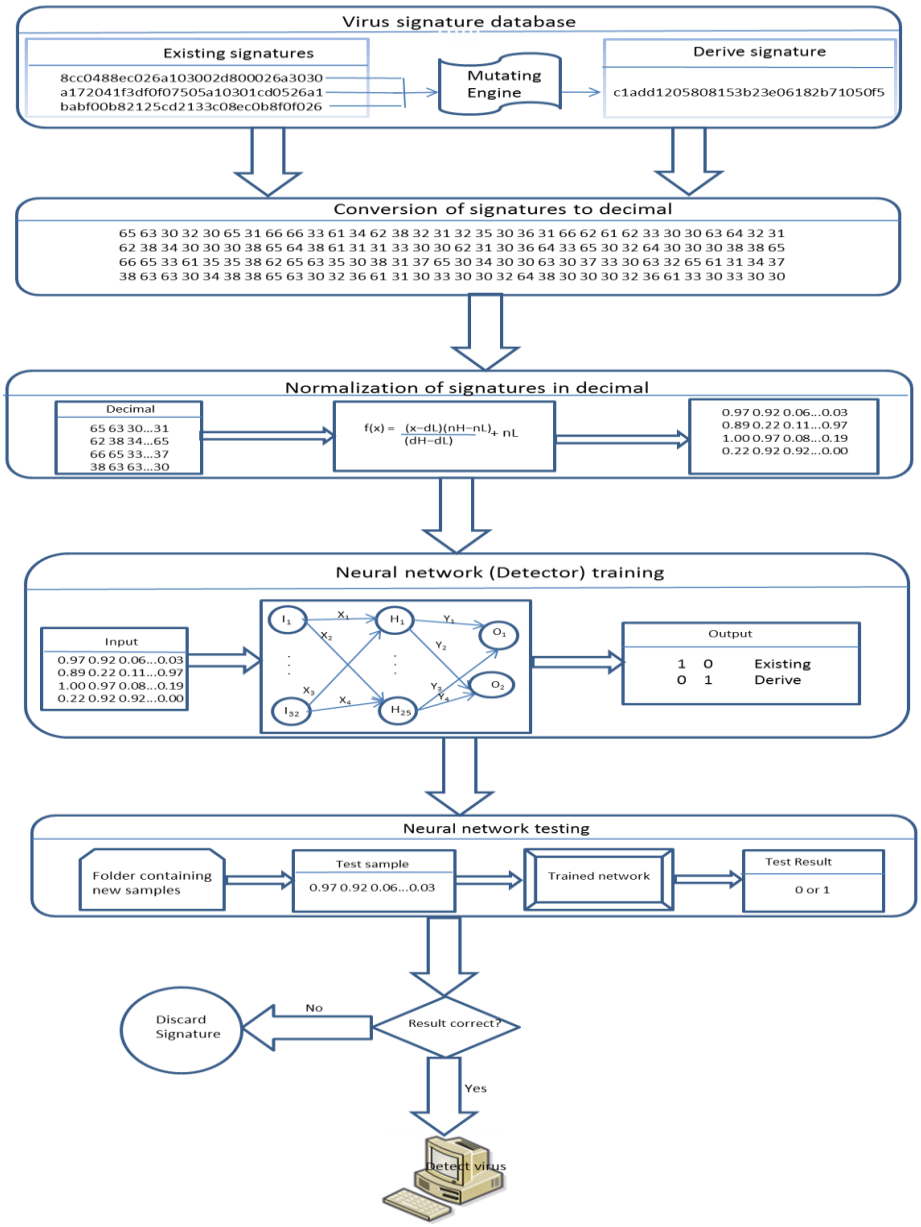
Andree & Nhien-An (2016) uses statistical methods, Naïve-Bayes algorithm and N-grams to find out if there is significant data that could be used to classify malware in control flow change. In order to find a discriminator between a malware and good software, Andree & Nhien-An (2016) first calculate and compare statistical values of the median, variance, variance coefficient and spread for both malware and good software. The second approach used by Andree & Nhien-An (2016) to find a discriminator is a Naïve-Bayes classifier. The classifier is first trained with sample datasets comprising of raw jump length data for both malware and good software. Thereafter, the classifier test unknown software against the data that has been learned by the classifier. The third approach used by the authors is based on the extraction on N-grams of words from a text. The N-grams of sample datasets are extracted and saved to a database for further comparison. Test samples are processed the same way the sample datasets are processed and saved into another database. The categorization test searches for occurrence of the to-test n-grams in the sample datasets and check for similarities. The data found in this work is not usable and its likelihood is too low to make a decision.

The reviewed literature shows that virus detection technique based on signatures is most commonly used technique (Kamarudin et al., 2013) and it has so many disadvantages. These include; human intervention in signature extraction, the need for users to regularly update the signature database (Vinod et al., 2010), time lag between virus creation and detection and most importantly, its inability to detect new and derived virus (Mishra, 2010). Some other approaches that use behavior to detect the presence of a virus have high false positive while some are still far from perfection. Some of the methods proposed in literature involve some error prone pre-processing and some of these are too complex to implement. Another approach that sounds promising might be; to use the concept of artificial immune system to enhance computing devices with their own immune system. However, training a computer on how to differentiate non-viral files from viral files seems to be a big challenge. This is important in a computing system where applications are installed and uninstalled on a regular basis. Some research explored the use of artificial intelligence in detecting computing viruses (Gang & Zhongquan, 2014). However, more research still needs to be conducted in order to fully explore options from the field of artificial intelligence. Furthermore, few research efforts focus on derived and metamorphic virus detection. These are some of the challenges faced by the current virus detection techniques that are proposed in literature. The next section discusses our proposed model and reflects on how it extends existing literature.

# 3. The proposed model for derived virus detection

In view of the current problems of detecting computing derived viruses as identified in the previous section, there is a need for a solution that can detect derived virus. The solution should be dynamic in nature and not dependent on pre-defined virus signatures. The solution should also learn from existing virus signatures and be able to detect derived viruses thereof. Hence, the authors propose a model based on these requirements. The proposed model is as shown in figure 1.

The proposed model comprises of various building blocks depicting different processes. The processes in each block are discussed below.




## 3.1 Virus signature database

This is the first block of the model. This block generates the dataset using the following steps:

### 3.1.1 Existing/known virus signatures

The first step of this model is to obtain existing virus signatures. The existing virus signatures can be obtained from any source such as VX Heaven and nlnetlabs website. However for this particular study the signatures were obtained from nlnetlabs website. The size of a virus signature string varies. This could be from a few bytes to tens of thousands of bytes (Wang, 2008). For the experimentation of this study, the authors consider signatures of 32 bytes. This is because several neural networks require the input to be represented as a fixed-length feature vector (Le et al., 2014). Therefore obtaining a generalized result requires the use of virus signatures with different bytes. This work considers fixed size string virus signatures. Hence, this is one of the limitations of this work.

### 3.1.2 Derive virus signatures

In this step, a mutating engine is designed, built and used to generate the derived virus signatures. The process starts by arranging known virus signatures into groups based on their similarities. Each group can be argued to come from a particular virus family which consisting of related virus signatures. Virus signatures in a family are

randomly combined to produce a derived virus signature for that family. This becomes the first derived virus signature and is then added to the family. Thereafter, another randomization is conducted taking into considerations both the existing signatures and the newly generated derivative. The result is a second derivative signature. This process is repeated three times to generate three new derivatives. Using the mutating engine, this process is repeated across all the groups of virus signature families. From the existing virus signatures taken from nlnetlabs website, the mutating engine was able to generate new virus derivatives of three generations grouped into a number families. The aim here is to simulate a derive virus signature using the existing virus signatures and train the system to be able to identify it later. This is necessary since this research aims to train a neural network to be able to detect derived viruses based on the existing viruses.

## 3.2 Conversion of the virus signatures

The second block of the model converts the virus signatures obtained in stage 1 from their string to decimal forms. This is necessary because most neural networks are designed to accept numbers as input (Heaton ,2011). It is also possible to represent the signatures in other numerical formats like binary and ASCII. However, this might introduce more complexity. For example, binary representation of each virus signature might generate a lot of 0's and 1's which might not be easy to manage. For example, the first four characters of the signature "b840008ed8a11300b106d3e02d00088e" gives "0110001000111000 00110100000110000" in binary. Furthermore, ASCII representation might not be necessary since the signatures do not contain special characters. This is another limitation of this work. It does not consider virus signatures that have special characters. However, our future work will incorporate special characters of virus signatures.

## 3.3 Normalization

The output from the block above is the signatures written in a decimal form. This is normalized to improve the efficiency of the model. For this experiment, all training data inputs are normalized between -1 and 1 using the formula below. This is because a hyperbolic tangent activation function is used to scale the output.

$$F(x) = \frac{(x - dL)\ (nH - nL)}{(dH - dL) + nL}$$

The above equation normalizes a value *x*, where the variable *dH* represents the high value of the data, variable *dL* represents the low value of the data and the variable *n* represents the high and low normalization range desired i.e. -1 and 1.

The training data targeted output, i.e. the classes (existing and derived) are encoded also. "1, 0" denotes an existing signature and "0, 1" denotes a derived virus signature.

## 3.4 Neural network training

For this study, the authors train and use a neural network as the virus detector. Our model uses a supervised multilayer feed-forward neural network. The authors chose a multilayer feed-forward neural network because it can learn a mapping of any complexity. The network learning is based on repeated presentation of training samples with each input matched to its corresponding output. Multilayer perceptrons are feed-forward networks with one or more nodes between the input and output layer. Figure 2 shows a multilayer feed-forward neural network.

The first layer is the input layer which accepts data ($x_1, x_2 \ldots x_n$) into the neural network and it is linked directly to the hidden layer. The hidden layer lies between the input layer and the output layer. The output layer is the last layer. This layer gives the neural network predicted output to the external party where it is needed for further processes or decision. Each node in the input layer has value for every input ($x_1, x_2 \ldots x_n$). The neural network initially assigns weights ($w_{11}, w_{21} \ldots w_{4n}$) to each connection. Each input is multiplied by its corresponding weight and the result of each connection is summed together and passed to the hidden layer nodes ($h_1, h_2, h_3 \ldots h_n$). A hyperbolic tangent activation (∫) function is used to scale the result and the final value becomes the value for the

node the connections entered. This calculation is also performed on the nodes in the hidden layer using the values that entered the hidden layer nodes and the assigned weights ($w2_{11}…w2_{m4}$) and result is sent to the output layer ($y_1$, $y_m$). The neural network checks its predicted output against the desired output; error is calculated by subtracting the desired output from the predicted output. The weights are adjusted based on the error and the iteration continues until the error becomes very small.

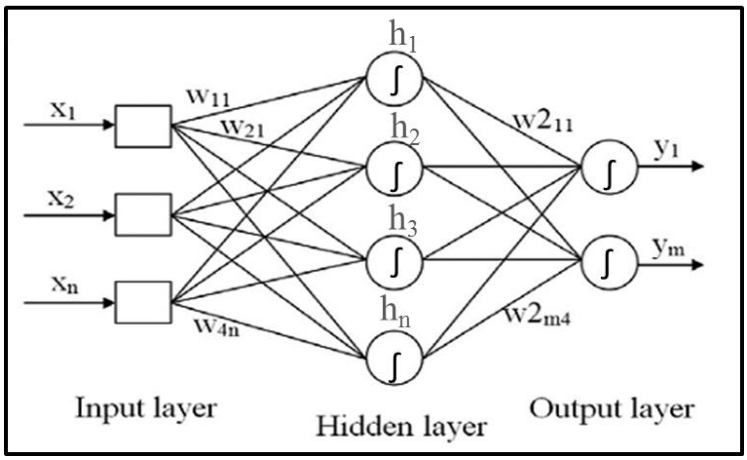



Figure 2: A multilayer neural network (Zainal-Mokhtar & Mohamad-Saleh, 2013)

## 3.5 Neural network testing

The performance of neural networks is determined by their generalization ability. A generalization is the property of trained neural networks to classify an input correctly even if it is not a member of the training set. This is what block four of this model does. New samples that are not part of the training dataset are given to the network for classification. A correct classification of the hidden samples means such a network has learned. Incorrect classification means otherwise.

Finally, the trained neural network is used as a detector within a computer system to detect both existing and derived viruses.

# 4. Experimental results

After training the neural network, the trained neural network code is converted to a class and deployed as a library (.dll). This library is used in a C# code for the implementation of this model. The mutating part of this model generated new derived virus signatures up to three generations. These are used for testing the model. The signatures are presented to the model and some of the results are shown in the table below.

| First generation | | | Second generation | | | Third generation | | |
|---|---|---|---|---|---|---|---|---|
| 0,0454 | 0,9756 | Derived | 0,1007 | 0,9144 | Derived | 0,4794 | 0,6531 | Derived |
| 0,8673 | 0,1376 | Existing | 0,0327 | 0,9748 | Derived | 0,6708 | 0,5171 | Existing |
| 0,0053 | 0,9936 | Derived | 0,2285 | 0,7798 | Derived | 0,9293 | 0,142 | Existing |
| 0,8727 | 0,1574 | Existing | 0,3077 | 0,6984 | Derived | 0,4246 | 0,4912 | Derived |
| 0,0053 | 0,9922 | Derived | 0,0324 | 0,973 | Derived | 0,4206 | 0,6591 | Derived |
| 0,7363 | 0,5559 | Existing | 0,9402 | 0,1404 | Existing | 0,6369 | 0,5265 | Existing |
| 0,5687 | 0,6287 | Derived | 0,7385 | 0,4545 | Existing | 0,6299 | 0,509 | Existing |
| 0,0867 | 0,8473 | Derived | 0,5812 | 0,3784 | Existing | 0,5481 | 0,4281 | Existing |

Table 1: Some results of the model on newly derived virus signatures

Table 1 shows some of the result of the experiment conducted with the model. The first result (0,0454 and 0,9756) for the first generation has its second value greater than the first. This means the model categorized the input

signature as a derived virus signature since we decoded derived virus signatures as “0, 1” during the training. Furthermore, the second result (0,8673 and 0,1376) has its first value greater than the second. This means the model categorized the input signature as an existing virus signature since we decoded existing virus signatures as “1, 0” during the training. The other results for the second and third generations are interpreted same way.

The three generations consist of the same number of virus signatures. In the first generation, the model was able to classify 90.5% signatures successfully as derived virus signatures and 9.5% signatures were incorrectly classified as existing virus signatures. Furthermore, in the second generation, 82.9% signatures were successfully classified as derived virus signatures and 17.1% signatures were incorrectly classified as existing virus signatures. Lastly, in the third generation, the model successfully classified 65.3% signatures as derived virus signatures and 34.7% signatures were incorrectly classified as existing virus signatures. The figure below shows the graphical representation of the results.

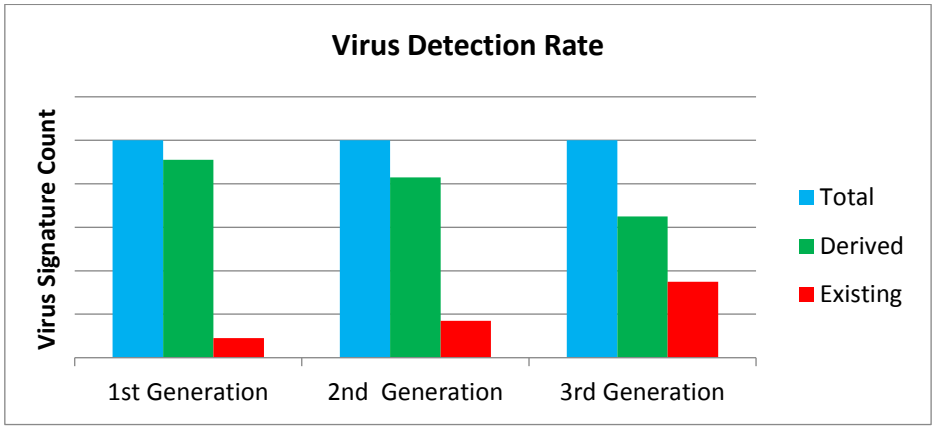



Figure 3: Graphical representation of experiment result

The first generation showed the highest number of derived virus detection rate than the other two generations. The second generation also showed a higher detection rate than the third generation which has the least detection rate. Based on this, a conclusion can be drawn that the correlation between the actual virus signature and its derivatives decreases further down along the generations. This means that after many generations of a virus changing forms, its variants will no longer look like the original. The variants will look like a completely new virus even though the variants and the original virus will always have same behavior and operation with similar effects. The next section concludes the paper and provides recommendations for future work.

# 5. Conclusion

This paper presents a novel model for detecting derived virus. The ability of our model to accurately classify derived virus signatures means that it can be used as virus detector in a computer system. The model has proven to work better than many solutions that are based on signature scanning. This is mainly because the model does not depend solely on a set of pre-defined virus signature; it also postulate what future signatures might look like and makes an attempt to detect them. Furthermore, the model is able to classify viruses used for the training and also their new variants that were not used during the training. Currently, the proposed model has an average success rate of:

- 80.2% on detecting existing signatures,
- 85.2% on signatures used for training and
- 80% on the average for derived virus signatures.

In terms of the derived virus signatures; this includes success rate of 91% on first generation, 83% on second generation and 65% on third generation. Of note in these results is that the accuracy of detecting derived viruses decreases linearly as the number of generations increase. For example, the success rate would have been even

lower on the fourth generation than it is on the third. This is to be expected because as we continue to add more generations to the training set; the difference between the original and the derivative virus signature gets even bigger. Furthermore, this explains the difficulty in current anti-virus systems to detect new viruses. Although, the result seems to have a lesser accuracy in percentile as compared to some existing solutions; it is important to note that the proposed model was able to detect derived viruses which cannot be detected by existing systems. Our future work will try and improve the accuracy of the model. Furthermore, future work will experiment with variable virus signature sizes and those that contain special characters. Finally, the model is to be trained encrypted virus signatures for robustness.

# References


Andree, L., Nhien-An, L., 2016. Control Flow Change in Assembly as a Classifier in Malware Analysis. *2016 4th IEEE International Symposium on Digital Forensics and Security,* pp. 2-7.

Chen, Y., Narayanan, A., Pang, S., Tao, B., 2012. Multiple sequence alignment and artificial neural networks for malicious software detection. *2012 8th International Conference on Natural Computation*, (Icnc), pp.261–265. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6234576.

Clements, A., 2014. The No Operation Instruction. Available at: http://alanclements.org/nops.html [Accessed August 11, 2016].

Daoud, E. A., 2009. Metamorphic Viruses Detection Using Artificial Immune System*, 2009 International Conference on Communication Software and Networks Metamorphic,* pp.168–172.

Feng, M., Gupta, R., 2009. Detecting virus mutations via dynamic matching. In *IEEE International Conference on Software Maintenance 2009*. Canada: IEEE, pp. 105–114.

Filiol, E., 2005. *Computer viruses : From theory to Applications* First Edition, Springer Berlin Heidelberg New York.

Gang, G., Zhongquan, C., 2014. A Kind of Malicious Code Detection Scheme Based on Fuzzy Reasoning. *2014 7th International Conference on Intelligent Computation Technology and Automation*, pp.19–22.

Golovko, V., Bezobrazov, S., 2015. Neural Network Artificial Immune System for Malicious Code Detection. *Brest State Technical University*, pp. 1–7.

Hamza, A., Hussain, D.J., 2014. Computer Virus Detection Based on Artificial Immunity Concept. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, 3(2), pp.68–74.

Heaton, J., 2011. *Programming Neural Networks with Encog3 in Java* 1st Edition, Heaton Research, Inc.

Ivanov, A., Makrushin, D., van der Wiel, J., Garnaeva, M. Namestnikov, Y., 2015. Kaspersky Security Bulletin 2015. Overall statistics for 2015. Available at: https://securelist.com/analysis/kaspersky-security-bulletin/73038/kaspersky-security-bulletin-2015-overall-statistics-for-2015/ [Accessed October 25, 2016].

Kakad, A.R., Kamble, S.G., Bhuvad, S.S., Malavade, V.N., 2014. Study and Comparison of Virus Detection Techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(3), pp.251–253.

Kamarudin, I.E., Sharif, S.A.M and Herawan, T., 2013. On Analysis and Effectiveness of Signature Based in Detecting Metamorphic Virus. *International Journal of Security and Its Applications*, 7(4), pp.375–386.

Kaspersky Lab, 2016. Kaspersky Security Bulletin: Overall statistics for 2016. Available at: https://securelist.com/statistics/ [Accessed January 16, 2017].

Kjell, B., 2015. The JUMP Instruction. Available at: https://chortle.ccsu.edu/AssemblyTutorial/Chapter-17/ass17_5.html [Accessed January 23, 2017].

Konstantinou, E., 2008. *Metamorphic Virus: Analysis and Detection*. Available at: https://www.ma.rhul.ac.uk/static/techrep/2008/RHUL-MA-2008-02.pdf [Accessed January 11, 2017].

Kumar, A., 2016. What is a Polymorphic Virus and how do you deal with it. Available at: http://www.thewindowsclub.com/polymorphic-virus [Accessed October 26, 2016].

Kuriakose, J., Vinod, P., 2014. Metamorphic Virus Detection using Feature Selection Techniques. In *2014 5th International Conference on Computer and Communication Technology*. IEEE, pp. 141–146.

Le, Q., Mikolov, T., Com, T.G., 2014. Distributed Representations of Sentences and Documents. *31st International Conference on Machine Learning, Beijing, China, 2014* , 32, p.1.

Mishra, U., 2010. *Methods of Virus Detection and Their Limitations*, Available at: http://ssrn.com/abstract=1916708 [Accessed January 1, 2017].

Nguyen, B.T., Ngo, B.T., Quan, T.T., 2012. A Memory-Based Abstraction Approach to Handle Obfuscation in Polymorphic Virus. *2012 19th Asia-Pacific Software Engineering Conference*, pp.158–161.

Rad, B.B., Masrom, M., Ibrahim, S., 2012. Camouflage in Malware : from Encryption to Metamorphism. *IJCSNS International Journal of Computer Science and Network Security*, 12(8), pp.74–82.

Silverman, J., 2001. Understanding Polymorphic Viruses. *2001*. Available at: http://www.commercialventvac.com/UnderstandingPolymorphicViruses.html [Accessed January 12, 2017].

Vinod P., Laxmi, V., Gaur, M.S., Chauhan, G., 2012. MOMENTUM : Metamorphic Malware Exploration Techniques Using MSA signatures. *2012 International Conference on Innovations in Information Technology*, pp.232–237.

Vinod, P., Jain, H., Golecha, Y.K., Gaur, M.S. Laxmi, V., 2010. MEDUSA : Metamorphic malware dynamic analysis using signature from API. *Proceedings of the 3rd International Conference on Security of Information and Networks*, pp. 1-6.

Wang, Q., 2008. Fast Signature Scan. , p.1. Available at: https://www.google.com/patents/US7454418 [Accessed January 25, 2017].

Zainal-Mokhtar, K., Mohamad-Saleh, J., 2013. An Oil Fraction Neural Sensor Developed Using Electrical Capacitance Tomography Sensor Data. *OALib Journal,* 13(9), pp. 11392.