Volume 9, Issue 3, March – 2024 International Journal of Innovative Science and Research Technology
ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAR2052

Enhanced Profile Hidden Markov Model for Metamorphic Malware Detection

[1]Ken Carlo D. Javier; [2]Allyza Maureen P. Catura; [3]Jonathan C. Morano; [4]Mark Christopher R. Blanco
[1,2,3,4]Computer Science Department, Pamantasan ng Lungsod ng Maynila, Philippines

Abstract:- Metamorphic malware poses a significant threat to conventional signature-based malware detection since its signature is mutable. Multiple copies can be created from metamorphic malware. As such, signature-based malware detection is impractical and ineffective. Thus, research in recent years has focused on applying machine learning-based approaches to malware detection. Profile Hidden Markov Model is a probabilistic model that uses multiple sequence alignments and a position-based scoring system. An enhanced Profile Hidden Markov Model was constructed with the following modifications: n-gram analysis to determine the best length of n-gram for the dataset, setting frequency threshold to determine which n-gram opcodes will be included in the malware detection, and adding consensus sequences to multiple sequence alignments. 1000 malware executables files and 40 benign executable files were utilized in the study. Results show that n-gram analysis and adding consensus sequence help increase malware detection accuracy. Moreover, setting the frequency threshold based on the average TF-IDF of n-gram opcodes gives the best accuracy in most malware families than just by getting the top 36 most occurring n-grams, as done in previous studies.

Keywords:- *Consensus Sequence, Metamorphic Malware, N-Gram Analysis, Profile Hidden Markov Model, TF-IDF*

I. INTRODUCTION

Metamorphic malware is an emerging threat to traditional signature-based malware detection. With its ability to self-modify the code without changing the semantics, it is more difficult for signature-based malware detection to identify the malware as its signature changes as they modify. Moreover, it is also possible for metamorphic malware to mutate by creating multiple copies of the same malware with different signatures. Thus, many variants created from a single metamorphic malware make signature-based malware detection impractical and ineffective [1]. Due to technology continuously expanding over time, malware evolves to have the ability to modify itself as it propagates. Metamorphic malware rewrites its code using various obfuscation techniques to alter the malware's code. Dead code insertion is an obfuscation technique where block/s of code or whitespaces are inserted without changing the code's functionality. Another technique is variable renaming which refers to changing the name of a variable in the source code. The third technique is instruction reordering, where the declaration of variables is swapped. Function reordering has a similar implementation to instruction reordering. However, functions are arranged differently in various permutations and combinations. Lastly, instruction substitution is another simple technique where the instruction operators are changed. These abovementioned obfuscation techniques are effective in modifying the code but not removing the original functionality of the malware. Furthermore, it is also possible for metamorphic malware to create multiple variants of the same malware using a specific obfuscation technique [2]. As a result, machine learning-based malware detection systems have been utilized and investigated in past years.

Machine learning-based malware analysis and detection are being practiced by anti-malware companies. A particular malware file can be disassembled, and relevant information about the malware can be retrieved, such as opcode and API call sequences. This relevant information can be used to train a machine learning model, such as decision trees and neural networks. After training the model with the data gathered, the model will be applied to testing data, and the model will be able to conclude predictions on which file is malware or benign. Many machine-learning techniques are used in malware analysis, like Random Forest, Support-Vector Machine, and Neural Network [3]. Another machine learning technique used in malware detection is the Hidden Markov Model.

The Hidden Markov model (HMM) is a statistical model that was first proposed by Baum L.E. and uses a Markov process that contains hidden and unknown parameters. This model uses the observed parameters to identify the hidden parameters. These parameters are then used for further analysis [4]. The model consists of an emission probability matrix, transition probability matrix, and initial probability distribution. The emission probability matrix shows the probability of an observation being generated in a hidden state. The transition probability matrix indicates the probability of each state moving to another state. Furthermore, the initial probability distribution indicates the probability that the Markov chain will start in a given observation.

Profile Hidden Markov Model (PHMM) is another variant of the standard HMM. PHMM is a probabilistic model that captures the diversity of biological sequences. HMMs and PHMMs differ significantly because the latter explicitly utilize the positional information in the observation sequences, while standard HMMs cannot do so. In contrast to conventional HMMs, PHMMs accommodate null transitions, which are essential to align sequences with insertions and

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAR2052

deletions. These differences are evident in DNA. PHMM can be applied to metamorphic malware because they are similar to DNA. [5]. Multiple sequence alignments are utilized to create the PHMM, and the position-based scoring system helps detect if a specific sequence is similar to the model [6]. To better understand how PHMM is built with MSA, an example of MSA from sequences using the four bases of DNA is in Fig. 1.

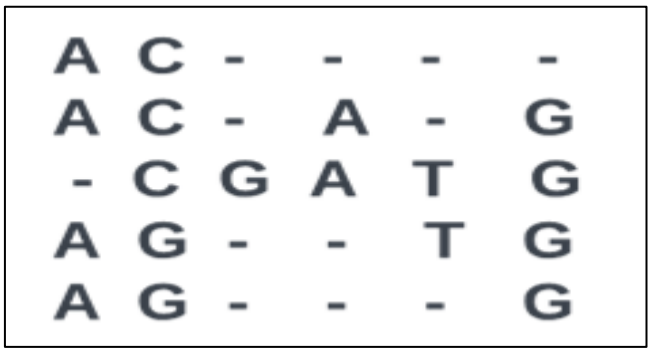

Fig 1: Sample MSA of DNA Sequences

In the initial stages of constructing a PHMM, identifying match and insert states in the MSA is being done. It is a general rule of thumb to use those columns with at least 50% of the characters are symbols, and this column is called match states. On the other hand, those with no symbols or with gaps are called insert states. The match states in Fig. 1 are columns 1, 2, and 6.

The emission probability for column 1 is then calculated in (1).

$$\begin{aligned} eM_1(A) &= 4/4(A) \\ eM_1(C) &= 0/4(C) \\ eM_1(G) &= 0/4(G) \\ eM_1(T) &= 0/4(T) \end{aligned} \quad (1)$$

Most of the emission probabilities are zero, which should be changed because the model should be adaptable. The "Add-one rule" is a straightforward formula that requires us to add 1 to the numerator and the total number of alphabetic symbols to the denominator.

The results of the emission probabilities with the add-one rule applied are in Table I. The general formula that can be used to calculate the emission probabilities is in (2).

$$eN(k) = \frac{Number\ of\ Occurences\ of\ k\ in\ State\ N}{Total\ Number\ of\ Symbols\ in\ State\ N} \quad (2)$$

Since a symbol can be emitted in more than one way, match or insert, the Emission Probabilities matrix (E) of PHMM differs slightly from the Symbol Transition Probability matrix (B) in HMM.

The transition probabilities are then calculated, and the general equation used to do so is in (3).

$$a_{mn} = \frac{No.\ of\ Transitions\ from\ m\ to\ n}{Total\ No.\ of\ Transitions\ from\ m\ to\ Any\ State} \quad (3)$$

Table 1: Emission Probabilities of MSA with the Add-One Rule Applied

Matches	**Inserts**
eM_1(A) = 4+1 / 4+4 = 5/8 eM_1(C) = 4+1 / 4+4 = 1/8 eM_1(G) = 4+1 / 4+4 = 1/8 eM_1(T) = 4+1 / 4+4 = 1/8	eI_1(A) = 0+1 / 0+4 = 1/4 eI_1(C) = 0+1 / 0+4 = 1/4 eI_1(G) = 0+1 / 0+4 = 1/4 eI_1(T) = 0+1 / 0+4 = 1/4
eM_2(A) = 0+1 / 5+4 = 1/9 eM_2(C) = 3+1 / 5+4 = 4/9 eM_2(G) = 2+1 / 5+4 = 3/9 eM_2(T) = 0+1 / 5+4 = 1/9	eI_2(A) = 2+1 / 5+4 = 3/9 eI_2(C) = 0+1 / 5+4 = 1/9 eI_2(G) = 1+1 / 5+4 = 2/9 eI_2(T) = 2+1 / 5+4 = 3/9
eM_3(A) = 0+1 / 4+4 = 1/8 eM_3(C) = 0+1 / 4+4 = 1/8 eM_3(G) = 4+1 / 4+4 = 5/8 eM_3(T) = 0+1 / 4+4 = 1/8	eI_3(A) = 0+1 / 0+4 = 1/4 eI_3(C) = 0+1 / 0+4 = 1/4 eI_3(G) = 0+1 / 0+4 = 1/4 eI_3(T) = 0+1 / 0+4 = 1/4

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAR2052

Table 2: Transition Probabilities of MSA with the Add-One Rule Applied

Beginning/ Matches	Inserts	Deletes
$aBM_1 = 5/8$ $aBI_1 = 1/8$ $aBD_1 = 2/8$	$aI_0M_1 = 1/3$ $aI_0I_0 = 1/3$ $aI_0D_1 = 1/3$	
$aM_1M_2 = 5/7$ $aM_1I_1 = 1/7$ $aM_1D_2 = 1/7$	$aI_1M_2 = 1/3$ $aI_1I_2 = 1/3$ $aI_1D_2 = 1/3$	$aD_1M_2 = 2/4$ $aD_1I_1 = 1/4$ $aD_1D_2 = 1/4$
$aM_2M_3 = 2/8$ $aM_2I_2 = 4/8$ $aM_2D_3 = 2/8$	$aI_2M_3 = 4/8$ $aI_2I_2 = 3/8$ $aI_2D_3 = 1/8$	$aD_2M_3 = 2/8$ $aD_2I_2 = 2/8$ $aD_2D_3 = 2/8$
$aM_3E = 5/6$ $aM_3I_3 = 1/6$	$aI_3E = 1/2$ $aI_3I_3 = 1/2$	$aD_3E = 2/3$ $aD_3I_3 = 1/3$

II. METHODS

A. Dataset

The dataset consists of 800 malware files and 200 benign files. The malware files are from different malware families: Locker, Mediyes, Winwebsec, Zbot, and Zeroaccess. These malware files are extracted from VirusTotal, VirusShare, and Malicia Project. On the other hand, the benign files are legitimate software applications. These are collected from download.cnet.com. These files are divided into training and testing datasets. Specifically, 80% are for training, while the remaining 20% are for testing. For training datasets, 160 malware files per family are utilized. Consequently, 40 malware files per family and 40 benign files are used in testing.

B. Pre-Processing

The malware files were disassembled using Ida Pro. Ida Pro is a well-known disassembler and debugger software used in reverse engineering. The proponents wrote a Python script to disassemble the files in batches rather than doing it per file. Malware and benign executable files were fed into the application, and the outputs were the executable files' assembly code in .ASM file format.

The proponents utilized Visual Studio Code in writing the code for the study. Visual Studio Code is an open-source code editor developed by Microsoft, and it supports a wide range of features such as syntax highlighting, Git integration, IntelliSense, and code refactoring [7]. Opcodes are extracted from the assembly files and created n-grams from these opcodes. In natural language processing, an n-gram refers to a consecutive sequence of n elements extracted from a sequence. The elements are typically words, but they can also be phonemes, characters, or other linguistic units [8]. The n-gram length ranging from one to three are utilized in the study, called unigram, bigram, and trigram. TF-IDF is computed for each n-gram in each file. TF-IDF means frequency-inverse document frequency, and it is a metric employed in information retrieval (IR) and machine learning to assess the relevance of textual representations, such as characters, words, phrases, and lemmas, in a corpus of documents [9]. After collecting all the TF-IDF scores, these n-gram opcodes are sorted by their scores. Different TF-IDF thresholds were implemented in the study, such as the average TF-IDF, 5%, and 10%. These thresholds were the basis on which n-gram opcodes were included in the training of the model. The proponents also filtered out the 36 most occurring n-gram opcodes, as done in previous studies [2], [10], [11]. The n-gram opcodes with TF-IDF greater than or equal to the threshold were converted to alphanumeric and special characters and saved into a JSON file. These characters are going to be utilized to create multiple sequence alignments. Multiple Sequence Alignment (MSA) is the alignment of multiple sequences with similar lengths. By analyzing the resulting alignment, one can deduce homology and explore the evolutionary relationships among the sequences. [12].

C. Training a Profile Hidden Markov Model

The proponents utilized a C++ library, namely SPOA, through its Python binding named pyspoa, where multiple sequence alignments are generated using a partial order alignment algorithm [13]. The package can generate only one consensus sequence. For this reason, the proponents wrote a Python implementation that generates multiple consensus sequences depending on the MSA. Malware behaves the same way as biological viruses. For that reason, methods utilized to eradicate biological viruses are also being used in malware [14]. An example of this is the consensus sequences. A consensus sequence is a sequence that represents a group of sequences shared by two biological entities that can be extracted from multiple sequence alignments. Many scientific studies have already utilized consensus sequences. They said that in bioinformatics, the consensus sequences determine variants of sequences in a group [15]. This is important because it overcomes the diversity of sequences and will most likely represent the valuable sequences [16]. Generating multiple consensus sequences is important because two or more characters may have the same quantity in a column of MSA. Therefore, two or more sequences could represent the structure of the MSA. The MSA is saved as an array, and the residue list in a JSON file. A residue list is a list or array of all the characters used to convert n-gram opcode sequences to alphanumeric and special characters.

The generated MSA will be the input of the Profile HMM builder, together with the residue list. The proponents created the Profile HMM builder with the help of an open-sourced GitHub repository named Bioinformatics by

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAR2052

Armaghan Sarvar. The PHMM is trained using the Baum-Welch learning algorithm and saved as a JSON file. The JSON file contains the transition probability matrix and emission probability matrix.

D. Testing

Forward Algorithm is a dynamic programming algorithm used to score a sequence against the Profile Hidden Markov Model and to determine how well the sequence matches the model. It calculates the degree of similarity between the sequence and the model, a higher score indicates a greater likelihood of the sequence belonging to the malware family. This algorithm solves the first problem in HMM, which is determining the likelihood of a sequence that matches the model [17].

The log-likelihood value for each sequence from the testing dataset is compared to the threshold value, the minimum log-likelihood of the training dataset, obtained during the training phase. The benign software is also tested, and their log-likelihood against the PHMM of malware families is collected.

E. Performance Metrics

The performance metrics utilized in the study are accuracy and the false positive rate. The formula for accuracy is in (4).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

where TP or True Positives refers to the instances the model predicts the sample as malware when it is a malware; TN or True Negatives is when the sample is found to be not malware, and it is not a malware; FP or False Positives is when the model predicts that a sample is a malware when it is not, and FN or False Negatives is when the model determines a sample is not a malware when it is a malware.

On the other hand, the formula for the false positive rate is in (5).

$$False\ Positive\ Rate = \frac{FP}{FP + TN} \quad (5)$$

where FP is False Positives and TN is True Negatives. This rate is when benign samples are incorrectly predicted as malware.

III. RESULTS AND DISCUSSION

This chapter discusses the results obtained from the methods and experiments conducted within the paper. The evaluation metrics used to show the effectiveness of the model are accuracy and false positive rate.

Table III shows the model's effectiveness in classifying Zbot malware files. These results show that increasing the n-gram length improves the accuracy of malware detection and lowers the false-positive rate. Results also show that increasing the number of n-gram opcodes improves malware detection accuracy. Adding of consensus sequence improves the accuracy of malware detection. However, this is only evident in the approach where the top 5% is used as a threshold.

The proponents also used the Zeroaccess malware family to train and test the enhanced model. The results can be found in Table IV. The n-gram length of 1 and 2 have given an accuracy of 100% and a false positive rate of 0%. Additionally, when more filtered n-gram opcodes are utilized when creating MSA, it has been shown to have an accuracy of 100% and a false positive rate of 0%.

Moreover, Table V shows the effectiveness of the improved model in detecting Winwebsec malware files. The results also show that increasing the length of n-gram improves malware detection accuracy and lowers the false-positive rate. In addition, results show that involving more filtered n-gram opcodes in constructing MSA improves malware detection accuracy. Nevertheless, adding consensus sequences when creating MSA does not affect the accuracy and false positive rate. Only the approach when the average is utilized as a threshold has shown that adding consensus sequences slightly improves the model's accuracy.

Another malware family has been used in the training and testing of the dataset: the Locker malware family. The results of the proposed model where Locker malware files are utilized can be found in Table VI. In this malware family, it is not observed that the increasing length of n-grams affects the model's performance. Instead, using only an n-gram length of 1 has provided significant accuracy and a false positive rate of 0. Nonetheless, when the accuracy fell to 74.58% and with a 15% false positive rate, because of the addition of the consensus sequences, this low accuracy was increased to 80% and lowered the false positive rate to 4.17%.

Lastly, Table VII shows the results of the improved approach in detecting Mediyes malware files. The proponents obtained a higher accuracy and low false positive rate when more filtered n-gram opcodes were utilized when creating MSA.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAR2052

Table 3: Test Results of the Proposed Approach for Classifying Zbot

N-gram Type	Threshold	Filtered N-grams	Consensus Sequences	Accuracy	FPR
Unigram	Top 36	36	-	97.08%	3.33%
Bigram	Top 36	36	-	99.58%	0
Trigram	Top 36	36	-	100%	0
Unigram	Top 36	36	Added	97.08%	3.33%
Bigram	Top 36	36	Added	99.58%	0
Trigram	Top 36	36	Added	100%	0
Unigram	>= Average	12	-	99.17%	0
Bigram	>= Average	76, 72, and 73	-	100%	0
Unigram	>= Average	12	Added	99.17%	0
Bigram	>= Average	76, 72, and 73	Added	100%	0
Unigram	Top 5%	5	-	88.33%	19.17%
Bigram	Top 5%	57	-	99.17%	0
Unigram	Top 5%	5	Added	89.17%	17.5%
Bigram	Top 5%	57	Added	99.17%	0
Unigram	Top 10%	10	-	96.67%	0
Bigram	Top 10%	92	-	100%	0
Unigram	Top 10%	10	Added	96.67%	0

Table 4: Test Results of the Proposed Approach for Classifying Zeroaccess.

N-gram Type	Threshold	Filtered N-grams	Consensus Sequences	Accuracy	FPR
Unigram	Top 36	36	-	97.92%	0
Bigram	Top 36	36	-	99.17%	0
Trigram	Top 36	36	-	91.67%	15.83%
Unigram	Top 36	36	Added	97.92%	0
Bigram	Top 36	36	Added	99.17%	0
Trigram	Top 36	36	Added	91.67%	15.83%
Unigram	>= Average	16	-	99.17%	0
Bigram	>= Average	92	-	100%	0

Table 5: Test Results of the Proposed Approach for Classifying Winwebsec

N-gram Type	Threshold	Filtered N-grams	Consensus Sequences	Accuracy	FPR
Unigram	Top 36	36	-	99.58%	0
Bigram	Top 36	36	-	100%	0
Trigram	Top 36	36	-	100%	0
Unigram	Top 36	36	Added	99.58%	0
Bigram	Top 36	36	Added	100%	0
Trigram	Top 36	36	Added	100%	0
Unigram	>= Average	17	-	97.08%	0
Bigram	>= Average	92	-	100%	0
Unigram	>= Average	17	Added	97.5%	0
Bigram	>= Average	92	Added	100%	0
Unigram	Top 10%	11	-	97.5%	1.67%
Unigram	Top 10%	11	Added	97.5%	1.67%

Table 6: Test Results of the Proposed Approach for Classifying Locker

N-gram Type	Threshold	Filtered N-grams	Consensus Sequences	Accuracy	FPR
Unigram	Top 36	36	-	99.58%	0
Bigram	Top 36	36	-	98.33%	0
Trigram	Top 36	36	-	74.58%	15%
Unigram	Top 36	36	Added	99.58%	0
Bigram	Top 36	36	Added	98.75%	0
Trigram	Top 36	36	Added	80%	4.17%
Unigram	>= Average	18	-	97.08%	0
Bigram	>= Average	92	-	95%	0
Unigram	>= Average	18	Added	97.92%	0
Bigram	>= Average	92	Added	95%	0

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAR2052

Trigram	>= Average	92	-	97.5%	0
Trigram	>= Average	92	Added	97.5%	0

Table 7: Test Results of the Proposed Approach for Classifying Mediyes

N-gram Type	Threshold	Filtered N-grams	Consensus Sequences	Accuracy	FPR
Unigram	Top 36	36	-	97.92%	0
Bigram	Top 36	36	-	99.17%	0
Trigram	Top 36	36	-	91.67%	15.83%
Unigram	Top 36	36	Added	97.92%	0
Bigram	Top 36	36	Added	99.17%	0
Trigram	Top 36	36	Added	91.67%	15.83%
Unigram	>= Average	16	-	99.17%	0
Bigram	>= Average	92	-	100%	0
Unigram	>= Average	16	Added	99.17%	0
Bigram	>= Average	92	Added	99.17%	0
Trigram	>= Average	92	-	100%	0
Trigram	>= Average	92	Added	99.17%	0

IV. CONCLUSION

This paper proposed an enhanced Profile Hidden Markov Model (PHMM) was constructed with the following modifications: n-gram analysis to determine the best length of n-gram for the dataset, setting frequency threshold to determine which n-gram opcodes are going to be included in the malware detection, and adding consensus sequences to multiple sequence alignments of each malware family. The experiment showed that n-gram analysis and adding consensus sequences help increase malware detection accuracy. Additionally, setting the frequency threshold that will involve more n-gram opcodes to take part in malware detection gives better accuracy in most of the malware families than just getting only the top 36 most occurring n-gram opcodes, which has been done in previous studies.

For future work, the authors suggest creating an enhanced Profile Hidden Markov Model (PHMM) for other malware families. Next, comparing PHMM from an MSA made by partial-order alignment to the one made by progressive alignment may also help. Then, finding or constructing an MSA builder that allows extended characters can be good. For this research, a C++ library named SPOA has been used, which only allows up to 92 characters (uppercase letters, lowercase letters, and special characters). Finally, setting different gap thresholds when constructing PHMM in a given MSA may be experimented with to determine what will perform best. In this research, the standard gap threshold, which is 50% of the column in MSA has been used, but there are some cases in which increasing it up to 60% to 80% is needed, or else the PHMM cannot be built.

REFERENCES

[1]. Campion, M., Dalla Preda, M., & Giacobazzi, R. (2021). *Learning metamorphic malware signatures from samples. Journal of Computer Virology and Hacking Techniques, 17*(3), 167-183.

[2]. Wadhwani, A. (2019). *JavaScript Metamorphic Malware Detection Using Machine Learning Techniques.* https://doi.org/10.31979/etd.8rtn-buzk

[3]. Andreopoulos, W. B. (2021). Malware Detection with Sequence-Based Machine Learning and Deep Learning. In *Springer eBooks* (pp. 53–70). https://doi.org/10.1007/978-3-030-62582-5_2

[4]. Lan, Y., Zhou, D., Zhang, H., & Lai, S. (2017). Development of early warning models. In Early warning for infectious disease outbreak (pp. 35-7. Academic Press.

[5]. Attaluri, S. (2007). Detecting Metamorphic Virusis with Metamorphic Viruses. Department of Computer Science, San Jose State University, http://www.cs.sjsu.edu/faculty/stamp/students/Srilatha_cs298Report.pdf

[6]. Oliveira, L. G., & Gruber, A. (2021). Rational Design of Profile Hidden Markov Models for Viral Classification and Discovery. In *Exon Publications eBooks* (pp. 151–170). https://doi.org/10.36255/exonpublications.bioinformatics.2021.ch9

[7]. Heller, M. (2022, July 8). What is Visual Studio Code? Microsoft's extensible code editor. *InfoWorld.* https://www.infoworld.com/article/3666488/what-is-visual-studio-code-microsofts-extensible-code-editor.html

[8]. Aghammadzada, E. (n.d.). N-Grams NLP | Data Science and Machine Learning. *Kaggle.* https://www.kaggle.com/discussions/getting-started/186392

[9]. Anirudha Simha, Principle Associate Software Engineer, Kai Chatbot Team. (2021). Understanding TF-IDF for Machine Learning. *Capital One.* https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAR2052

[10]. Ali, M., Hamid, M., Jasser, J., Lerman, J., Shetty, S., & Di Troia, F. (2022). *Profile Hidden Markov Model Malware Detection and API Call Obfuscation.* https://doi.org/10.5220/0011005800003120
[11]. Alipour, A., & Ansari, E. (2020a). An advanced profile hidden Markov model for malware detection. *Intelligent Data Analysis*, *24*(4), 759–778. https://doi.org/10.3233/ida-194639
[12]. Embl-Ebi. (n.d.). *Bioinformatics Tools for Multiple Sequence Alignment < EMBL-EBI.* https://www.ebi.ac.uk/Tools/msa/
[13]. Vaser, R., Sović, I., Nagarajan, N., & Šikić, M. (2017). Fast and accurate de novo genome assembly from long uncorrected reads. *Genome Research, 27*(5), 737–746. https://doi.org/10.1101/gr.214270.116
[14]. Kostadimas, D., Kastampolidou, K., and Andronikos, T. (2021). Correlation of biological and computer viruses through evolutionary game theory. *arXiv (Cornell University).* https://doi.org/10.48550/arxiv.2108.00508
[15]. Vaschetto, L. (2022, December 20). *The Significance of Consensus Sequences in Bioinformatics.* News-Medical.net. https://www.azolifesciences.com/article/The-Significance-of-Consensus-Sequences-in-Bioinformatics.aspx#
[16]. Mohabati, R., Rezaei, R., Mohajel, N., Mm, R., Azadmanesh, K., and Roohvand, F. (2020). Optimizing Consensus Generation Algorithms for Highly Variable Amino Acid Sequence Clusters. *bioRxiv (Cold Spring Harbor Laboratory).* https://doi.org/10.1101/2020.11.08.373092
[17]. Jurafsky, D. & Martin, J. (2023). *Hidden Markov Models* [PDF file]. Stanford University Speech and Language Processing: https://web.stanford.edu/~jurafsky/slp3/