Profile Hidden Markov Model Malware Detection and API Call Obfuscation

Muhammad Ali[1], Monem Hamid[1], Jacob Jasser[1], Joachim Lerman[1], Samod Shetty[1] and Fabio Di Troia[2][a]

[1]*Department of Computer Engineering, San Jose State University, San Jose, CA, U.S.A.*

[2]*Department of Computer Science, San Jose State University, San Jose, CA, U.S.A.*

Keywords: PHMM, Malware Detection, Malware Obfuscation, API Calls, Dynamic Detection, Machine Learning.

Abstract: Profile Hidden Markov Models (PHMM) have been used to detect malware samples based on their behavior on the host system and obtained promising results. Since PHMMs are a novel way of categorizing malware and there is limited research work on such detection method, there is no data on the impact that certain obfuscation techniques have on PHMMs. An obfuscation tool that could weaken PHMM based detection has not yet been proposed. Our novel approach is based on applying PHMM detection by training the machine learning models on API calls that are dynamically extracted from the malware samples, and then attempting to elude detection by the same models using obfuscation techniques. Hence, in our paper, we created a PHMM model trained on API call sequences extracted by running malware in a sandbox, then we tried to undermine the detection effectiveness by applying different state-of-the-art API obfuscation techniques to the malware. By implementing sophisticated API calls obfuscation techniques, we were able to reduce the PHMM detection rate from 1.0, without API call obfuscation, to 0.68.

1 INTRODUCTION

Malicious software (malware) has been with us since the dawn of computer development. There are reports of malware being detected as far back as the 1970s (Suenaga, 2009), and since then they continue to evolve and infect various devices while increasing in quantity. It is a constant race between threat actors and security analysts to find an edge over the other. Malware threats have evolved from harmless annoyances to serious breaches such as ransomware and cyberterrorism (McKnight, 2017). Traditional approach for malware detection is based on signature detection, which relies on static analysis of malware beforehand, and security analysis software detecting malware based on those signatures (Sathyanarayan et al., 2008). Over time, however, malicious software has become increasingly complex, and signature detection cannot be trusted anymore for hundred percent accuracy in detection. That is where dynamic analysis and machine learning come into play. Recent advancements in machine learning techniques have provided the opportunity of applying it in the area of malware detection. There has been a lot of research into using statistical models, such as Hidden Markov Models trained on existing malware by extracting API calls (Alqurashi and Batarfi, 2017). This paper focused on the possibility of using PHMMs, and explored any weaknesses in its efficiency in catching unknown malware samples by applying state-of-the-art obfuscation techniques.

The remainder of this paper is organized as follows. In Section 2 we provide a selective survey of relevant work in this area. In Section 3, background topics are discussed, with a focus on the machine learning technique employed in this research. Section 4 covers the methodology used, with a description of the project architecture and its implementation. Section 5 gives our experimental results and analysis. Finally, Section 6 summarizes our results and includes a discussion of possible directions for future work.

[a] https://orcid.org/0000-0003-2355-7146

Ali, M., Hamid, M., Jasser, J., Lerman, J., Shetty, S. and Di Troia, F.
Profile Hidden Markov Model Malware Detection and API Call Obfuscation.
DOI: 10.5220/0011005800003120
In *Proceedings of the 8th International Conference on Information Systems Security and Privacy (ICISSP 2022)*, pages 688-695
ISBN: 978-989-758-553-1; ISSN: 2184-4356

2 RELATED WORK

Looking at the literature of malware detection, there are generally three approaches. The first is done by taking large surveys where the authors take broad known obfuscation techniques and compare them to detection methods currently deployed, such as in (You and Yim, 2010). The second approach is based on using new techniques in the field of malware detection, such as Profile Hidden Markov Models (PHMM) algorithms, and pairing them with currently known methods of analysis such as dynamic analysis (Alkhateeb, 2017), API calls and constructing birthmarks (Vemparala et al., 2016). These methods aid in trying to improve detection quality and create a more robust malware detection framework. The third approach taken by the literature is using malware detection as a baseline and applying novel techniques to obfuscate itself and escape detection, for example in (Srivastava et al., 2008). While the former approach provides a favorable overview of the state of current techniques, the latter two approaches to literature are in a sort of arms race. One side is trying to create methods to evade detection while the other side is creating methods to detect them. As it stands currently, there are many papers for obfuscation of a malware's signature, and there is substantial research for its detection. However, there is currently no research on how to obfuscate malware samples to hamper PHMM based detection. The majority of research work is broad and more surface level. It does not deepen into a technique, but rather it tests and measures a wide variety of techniques. Research is extensive for exactly how polymorphic and metamorphic malware uses techniques to modify itself in intelligent ways to evade detection at a high success rate (Singh and Singh, 2018). For example, they take certain types of evasion techniques commonly implemented by polymorphic and metamorphic malware, such as adding dead code, transposing code, or reordering subroutines (You and Yim, 2010). There is more research showing how there are already various techniques for malware detection, both static and dynamic to render common signature based detection obsolete, for example, the work in (Damodaran et al., 2017). This research mostly gives analysis on the arms race between malware detection and obfuscation, and provides good context and data to compare different methods of detection or obfuscation. There is also novel research being performed to evade detection. In fact, novel techniques such as conditional code obfuscation were developed and determined to evade state-of-the-art malware detection (Sharif et al., 2008). With new obfuscation schemes, new methods of detection are needed if malware can defeat detection and can hide malicious behavior and activity. With novel obfuscation techniques being developed, there are also novel detection methods being researched, particularly by using bioinformatics tools to analyze and catch how malware evolves (Wadkar et al., 2020). There is research done for specific kinds of evasions of malware where the malware's behavior will change if it thinks it is being analyzed (Kirat and Vigna, 2015). Additionally, there is research being done with using PHMM for malware detection. PHMM is usually used for categorizing protein families, but it has been found to be very effective in catching and categorizing malware families. PHMM has been used to detect malware in many different ways. PHMM has been incorporated in malware detection by using dynamic birthmarks (Vemparala et al., 2016) and has been used to classify known malware (Pranamulia et al., 2017). While some research obtained high sensitivity scores, they had a high false positive rate as well. Newer research has been performed and reached high accuracy and confidence in detection with a very low false positive rate (Alipour and Ansari, 2020). PHMM has also been used in a variety of ways to tackle detection using static and dynamic forms of analysis such as analyzing system call sequences (Pranamulia et al., 2017), behavior based analysis (Ravi et al., 2013), static analysis based on opcode sequences (Alipour and Ansari, 2020), and using dynamic analysis techniques (Vemparala, 2015). In parallel, there is also a lot of research being achieved for extracting API calls and using those calls for malware analysis. PHMM in particular has also been shown to be robust on a variety of operating systems and successfully detects malware using API extractions on various platforms such as Android (Sasidharan and Thomas, 2021). Research has been done for static detection of malware based on the binary's executable (Fu et al., 2008). Furthermore, there has been plenty of work demonstrating efficacy in extracting API calls and being able to classify them based on API sequences (Uppal et al., 2014), or based on API call frequency (Garg and Yadav, 2019). These methods are often paired with machine learning and yield promising results. Combining this approach with PHMM has shown to be very efficacious for detecting malware. Malware detection using dynamic birthmarks and PHMM has been measured to be very effective by using a windows sandboxing tool called Buster Sandbox Analyzer(Buster, 2021) to dynamically extract API calls (Damodaran et al., 2017).

3 BACKGROUND

In this Section, we briefly introduce the PHMM algorithm and the obfuscation techniques implemented in our experiments. A more detailed description of our implementation can be found in Section 4.

3.1 Profile Hidden Markov Model

Profile Hidden Markov Models (PHMMs) are probabilistic models that we used to train and test malware API calls. It could be seen as an advanced version of Hidden Markov Models (HMMs). In fact, one of the drawbacks of HMM is the Markov assumption, i.e. the current state depends only on the previous state. PHMMs, instead, uses positional information of sequences/symbols, and thereby gives a better fit for dynamic malware detection that looks at API calls being generated by malware (Stamp, 2017). PHMM algorithm is mainly based on two steps, that is, the pairwise alignment and the multiple sequence alignment (MSA). Pairwise alignment is a technique to align two sequences, where a substitution matrix and gap penalty function are used to align such sequences. In this paper, we used local alignment where two pairs are aligned locally using dynamic programming. MSA is generated from a collection of pairwise alignments. In this paper, we used Kruskal algorithm to construct a minimum spanning tree of pairwise alignments from highest scored pair to lowest sequence. Starting from the highest scoring pair, MSA is progressively aligned to include each sequence in every iteration. PHMM is then derived directly from MSA. More details about PHMM can be found in (Stamp, 2017), while implementation details followed in this paper are described more in in depth in Section 4.

3.2 Obfuscation Techniques

One of the obfuscation techniques that we implemented was to change the instructions after decompiling the malware samples, resulting in a change in the sample appearance while its core functionality stays unchanged. Our expectation was to get variable API logs with the changed instructions and, thus, hampering the dynamic detection based on such sequences. We also tested some existing state-of-the-art obfuscation tools, such as running malware files through Enigma Protector (The Enigma Protector Developers Team, 2021), which injects itself at the start of the code flow, and padding and copying API calls (Suenaga, 2009). Another tool that we tested was CallObfuscator (Mahmood, M., 2021). The tool works by replacing specific API calls in any of the DLLs in windows API, while keeping the functionality of the original PE intact.

4 METHODOLOGY AND PROJECT ARCHITECTURE

Architecture for PHMM based malware detection has multiple set-ups involving different subsystems for API call logs collections, deriving PHMM, and training and testing the models. Architecture is modularized into subsystems with different set-ups. Each subsystem is responsible for its function. Here, we see a brief description of each.

API Logs Collector. This set-up leverages Sandboxie and Buster Sandbox Analyzer to collect API call logs from malware and application execution.

Derive PHMM. API call logs collected by the above set-up are translated into sequences. These sequences are used to generate pairwise alignments and MSA. PHMM is then derived from MSA to be trained.

Train and Test PHMM. This system trains the PHMM using API call sequences belonging to various malware families. Once trained, we test the model against other malware and application API call sequences.

Obfuscate to Avoid Detection. In order to test PHMM efficiency and how effective it is at dynamic analysis of suspected malware samples, it is suggested to explore relevant obfuscation techniques and attempt to defeat the trained ML model that was developed.

A depiction of this architecture is given in Figure 1.

4.1 Implementation

In a Windows 10 virtual machine, we installed Buster Sandbox Analyzer and Sandboxie Classic (Sandboxie, 2021). On every new boot of the VM, we made sure to disable Windows Real-time protection as this would unintentionally delete malware test files in the operating system. This can cause issues if tested malware files are deleted, since our malware files come pre-packaged and compressed. Under "C:\Windows\Sandboxie.ini", we added the following configurations for a chosen sandbox:

```
InjectDll=[path to bsa's logapi32.dll]
InjectDll=[path to bsa's logapi64.dll]
OpenWinClass=TFormBSA
NotifyDirectDiskAccess=y
```

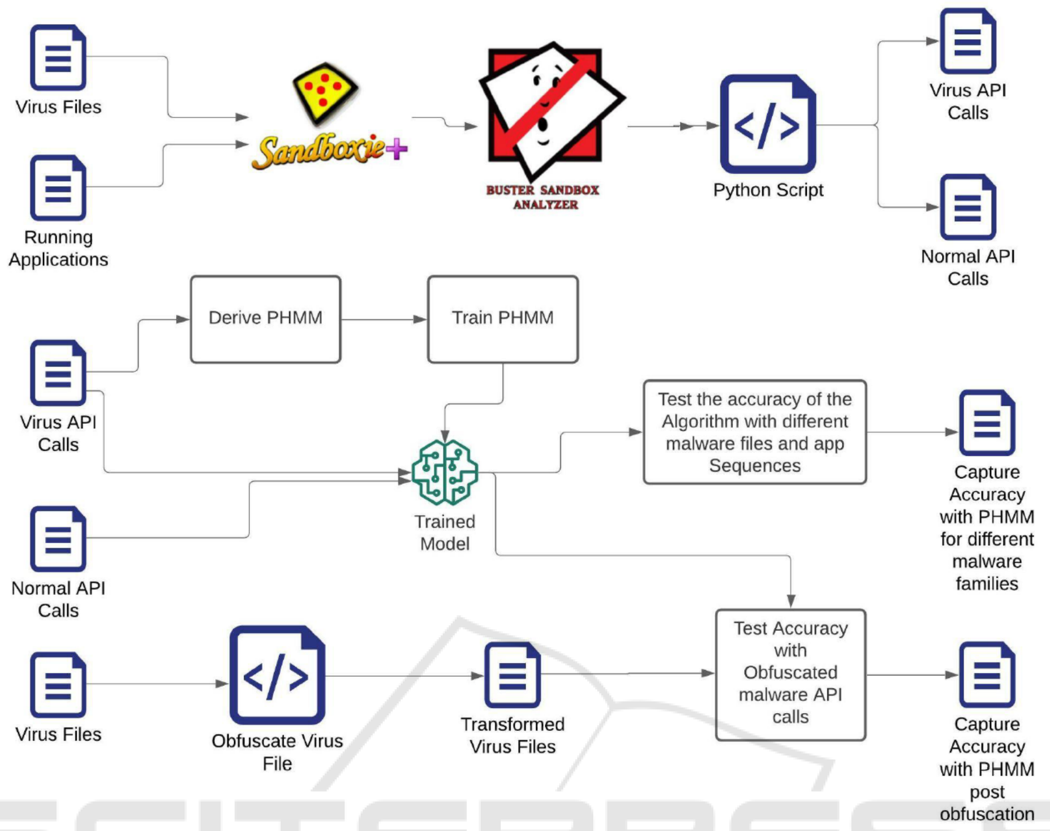
Virus Files
Running Applications
Sandboxie+
BUSTER SANDBOX ANALYZER
Python Script
Virus API Calls
Normal API Calls
Virus API Calls
Derive PHMM
Train PHMM
Trained Model
Normal API Calls
Test the accuracy of the Algorithm with different malware files and app Sequences
Capture Accuracy with PHMM for different malware families
Virus Files
Obfuscate Virus File
Transformed Virus Files
Test Accuracy with Obfuscated malware API calls
Capture Accuracy with PHMM post obfuscation

Figure 1: Architecture for PHMM Virus Detection and Obfuscation.

Then, we started Buster Sandbox Analyzer (BSA) and selected the Sandboxie folder containing the malware samples. We allowed the malware to run and, when it completed its execution, the malware log APIs was produced in the "Reports" directory in a file called "LOG_API.txt". A script was created to run BSA on a whole directory of applications.

After collecting API data on the malware families, we took the 36 most common API calls and assigned them an alphanumeric character. All other less common API calls where given the "*" character. The value 36 is the optimal number of most common API calls based on the work in (Vemparala et al., 2016). In fact, the top 36 API calls constitute more than the 99.8% of the total API calls for each family tested. A python script was implemented to convert the API logs generated from the Log API collection step to its corresponding character, then a shell script was used to convert all log files into a sequence file for a given malware family. Another Python program was written to generate the sequences from API call logs.

4.1.1 Pairwise Alignment

MSA Pairwise Alignment is a method of aligning sequences. This is a first step in generating MSA. For analyzing the performance of PHMM against different malware families, we generated an MSA for each malware family and a consolidated MSA representing all malware families. From the API log sequences generated by the script above, we shortlisted 10 sequences belonging to each malware family. Every pair of sequences was aligned locally with the score. We used a substitution matrix with a match having score of 10 and no-match having score of -10. We used a linear gap penalty function where "-" in the initial gap and extension of gap had the same penalty of -5. A python script and the Biopython library (Biopython, 2021) were used to align sequences as mentioned above. Biopython has a "pairwise2" module that has various functions that help in aligning the sequences. Based on the score associated with each pair, a minimum spanning tree is constructed with the highest scoring pair at the top. Each pair was translated into edges, and Kruskal algorithm was used to derive the highest performing minimum spanning tree. Nine pairs representing all 10 sequences were formed. A Multiple Sequence

Alignment (MSA) is constructed by implementing the Feng-Doolittle algorithm with progressive alignment of sequences. We followed the method explained in (Attaluri et al., 2009). In this method, alignment is accomplished in the order dictated by pairwise alignments in a minimum spanning tree. MSA is iteratively constructed with the highest scoring pair being the first to be added. Each pair is then progressively added to MSA, each time aligning the sequences. Figure 2 depicts the process to generate MSA.

This program outputs a "malware_family_msa.txt" file that will have an MSA which is going to be used to derive PHMM and training the model. More precisely, once we had MSA for each malware family, we loaded these MSAs into R. We used the aphid library (RDocumentation, 2021) to derive PHMM. PHMM was derived with residues consisting of alphanumeric characters (A-Z, 0-9) and an asterisk (*). The *k* value of 2 was chosen. Once PHMM was derived, it was trained using the BaumWelch algorithm. More than 600 malware samples were used to train this model. This trained model was saved onto the disk for future use to test malware detection. The PHMM model that was trained above was used for malware detection. We had separate test data consisting of more than 150 malware samples used to obtain metrics on the performance of the model. Forward algorithm (Stamp, 2017) was used to test the model against these samples. The algorithm returned the probability score. We also tested the model against the application logs and noted down the probability score. Later, both the results were fed into an AUC function to get the AUC score. Details on AUC score and the performance of the PHMM model can be found in Section 5. The code used in this research can be accessed at (Ali et al., 2021b) and (Ali et al., 2021a).

4.2 Dataset

The dataset is made of five families, that is, ZeroAccess, Zbot, Harebot, Trojan, and Winwebsec, for a total of 730 malware files split among the such families, plus 40 benign applications. To train the models, 600 malware samples are used, while the rest of the malicious files plus the benign samples are used for testing. The malware samples are part of the Malicia project (IMDEA Software Istitute, 2013).

5 EXPERIMENTS AND RESULTS

In this Section, we describe the experiments accomplished in this research. These experiments are based on three different ways to obfuscate API calls, to which we applied a trained PHMM model to quantify its ability to detect the obfuscated samples.

In the first experiment, we used a disassembler tool to explore compiled executables files (malware). This helped us in opening files, disassembly, and reading the code (assembly code). After decompiling the malware sample, instructions were added randomly at different places in the decompiled. The goal was to confirm that adding instructions would not change the API calls sequence. After recompiling a malware sample with additional instructions at different points, we ran the same sample in Sandboxie and captured the API logs via BSA. We confirmed that there was no change in the actual API call log for those obfuscated samples. This proved that obfuscating the code by changing the list of instructions does not affect the sequence of API calls.

In the second experiment, we swapped certain API calls with others in the Windows DLL IAT (Mahmood, M., 2021) with similar ones, thus to not alter the original behavior of the program. Hence, we modified the virus files so that the sequence of API calls when logged through BSA would change considerably. First we ran an unmodified malware file through BSA to capture the API call sequence. Next, we ran the dump to obtain the API calls list without obfuscation. Next, we chose an API call to swap, and modified the PE. Lastly, we reran the unmodified file through BSA to obtain the new obfuscated API call sequence. The resulting "API_LOG" was then rerun through a sequence mapping script, and the sequence was tested against the trained malware detection model to observe the result. The model was trained on the non-obfuscated version of the samples. What we observed was that such API modifications did not change the sequence enough to avoid detection, and dropped further obfuscation efforts through this tool. The results of this last experiment are shown in Figure 3, 4, 5, 6, and 7 for, respectively, Zeroaccess, Zbot, Winwebsec, Trojan, and Harebot family. In Figure 8, instead, we see the results of our multiclass classification experiment with all such families. Performance of the PHMM model is measured by drawing the ROC curve and measuring the area under it. PHMM performance is significantly better in malware families like Zeroaccess, Zbot, and Winwebsec (1.0, 0.97, and 0.95 AUC, respectively) compared to Trojan (0.82 AUC), that is the only family that has been affected considerably by the tested ob-

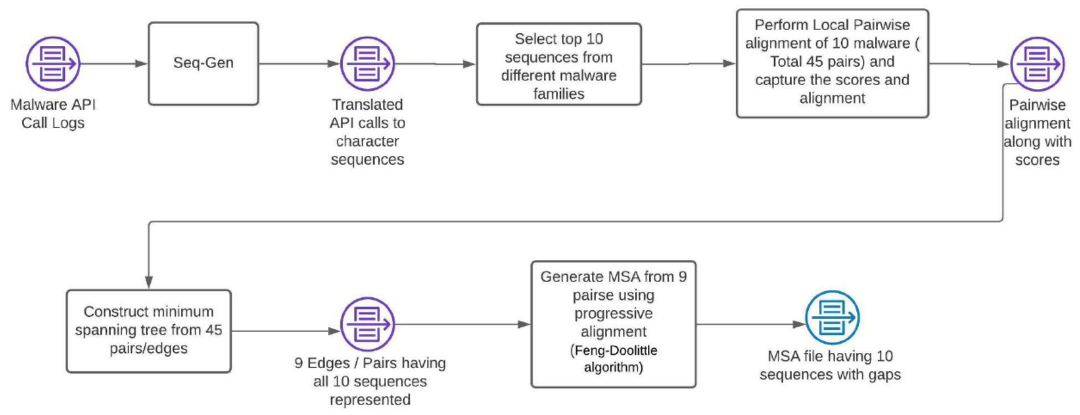
Malware API Call Logs
Seq-Gen
Translated API calls to character sequences
Select top 10 sequences from different malware families
Perform Local Pairwise alignment of 10 malware (Total 45 pairs) and capture the scores and alignment
Pairwise alignment along with scores
Construct minimum spanning tree from 45 pairs/edges
9 Edges / Pairs having all 10 sequences represented
Generate MSA from 9 pairse using progressive alignment (Feng-Doolittle algorithm)
MSA file having 10 sequences with gaps

Figure 2: Generating MSA.

fuscation technique. As a mean of comparison, we ran our model against the non-obfuscated version of the malware samples. The results obtained were all compatible to the work in (Vemparala et al., 2016), with a consistent 1.0 AUC for all the families. A summary of this experiment is given in Table 1.

In the third experiment, we attempted different techniques for obfuscation using callObfuscator, manually changing DLL instructions, and enigmaProtector. Out of all the various techniques, obfuscation achieved using enigmaProtector resulted in significant reduction of the AUC score when we utilized the API call logs from the obfuscated malware. Figure 9 shows the ROC curve for our multiclass experiment with obfuscated malware via enigmaProtector, which significantly reduced the AUC score from 0.92 (swapping the API calls with alternative ones) to 0.68.

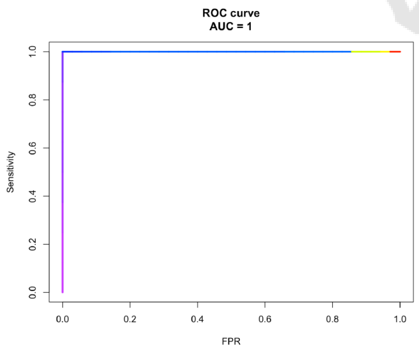

Figure 3: ROC curve for Zeroaccess family.

6 CONCLUSIONS

There is no evidence in literature of the influence of obfuscation strategies to evade PHMM-based malware detection. The potential shortcomings of PHMM-based detection, as well as how to circumvent detection, were yet to be proved using an ob-

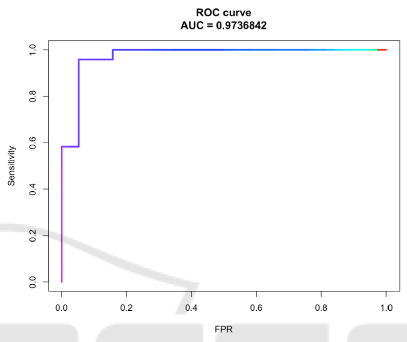

Figure 4: ROC curve for Zbot family.

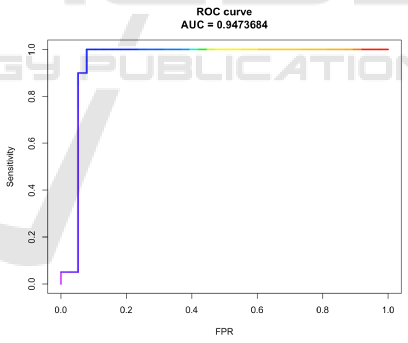

Figure 5: ROC curve for Winwebsec family.

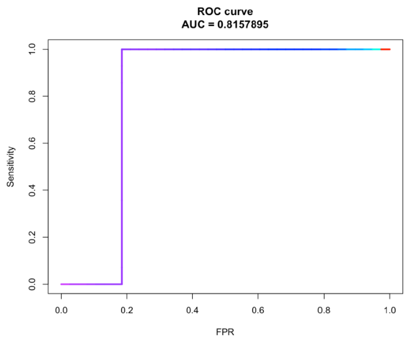

Figure 6: ROC curve for Trojan family.

Table 1: Malware Family AUC Scores.

Malware Family	AUC Score (obfuscated with enigmaProtector)	AUC Score (obfuscated with API swapping)	AUC Score (non-obfuscated)
ZeroAccess	0.77	1.0	1
Winwebsec	0.95	0.95	1
Zbot	0.53	0.97	1
Harebot	0.79	0.92	1
Trojan	0.61	0.82	1
Multiclass	0.68	0.92	1

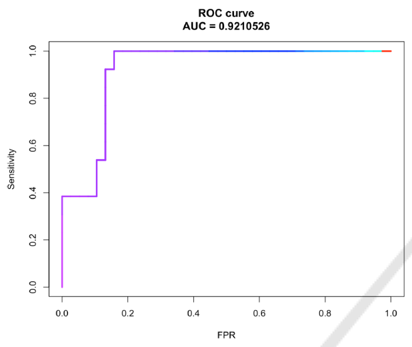

Figure 7: ROC curve for Harebot family.

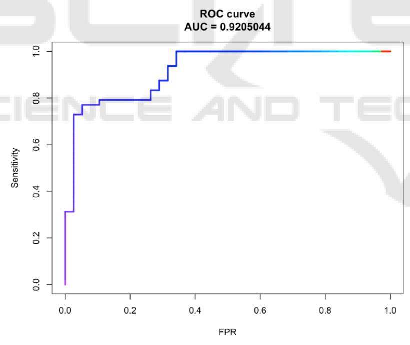

Figure 8: ROC curve for our multiclass experiment.

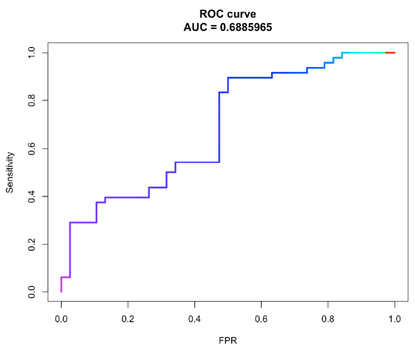

Figure 9: ROC curve after obfuscating the samples with enigmaProtector (multiclass).

fuscation tool. It is possible, in theory, to build machine learning models based on API calls collected from known malware samples, and then use malware obfuscation to evade detection by the same models. To test this strategy, we used PHMM to detect different malware families based on the API calls, and then tried a few state-of-the-art obfuscation techniques to evade detection. We used a tool called enigmaProtector which uses padded and copied API obfuscation, we decompiled the samples, added dummy instructions (such as NOPs) at different places within each sample, and then re-compiled the modified samples so that we could run it using our model. Finally, we tried modifying the malware executables with CallObfuscator which modifies the Import Address Table. We did not obtained much success with NOPs and CallObfuscator, however, by using enigmaProtector we got positive results in terms of evading detection. In fact, it reduced the AUC score from 0.92 to 0.68 which shows moderate effectiveness to evading detection from PHMMs.

Further research would include scripting custom malware obfuscation techniques such as dead code insertion, instruction changes, substitution, and padding to observe their effect on the list of API calls. Furthermore, utilizing additional malware families to observe the AUC resuls would be beneficial, also, observing the accuracy of the classification by integrating the dataset with both benign and malicious programs. Finally, expanding the feature set by extracting API calls both dynamically and statically, with the introduction of additional features to oppose the effect of enigmaProtector on the PHMM model.

REFERENCES

Ali, M., Hamid, M., Jasser, J., Lerman, J., Shetty, S., and Di Troia, F. (2021a). Malware-training-detection. https://github.com/SJSU-PHMM/

malware-training-detection. Online; accessed November 2021.

Ali, M., Hamid, M., Jasser, J., Lerman, J., Shetty, S., and Di Troia, F. (2021b). MSA-gen. https://github.com/SJSU-PHMM/msa-gen. Online; accessed November 2021.

Alipour, A. A. and Ansari, E. (2020). An advanced profile hidden markov model for malware detection. *Intelligent Data Analysis*, 24(4):759–778.

Alkhateeb, E. M. S. (2017). Dynamic malware detection using api similarity. In *2017 IEEE International Conference on Computer and Information Technology (CIT)*, pages 297–301. IEEE.

Alqurashi, S. and Batarfi, O. (2017). A comparison between api call sequences and opcode sequences as reflectors of malware behavior. In *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 105–110. IEEE.

Attaluri, S., McGhee, S., and Stamp, M. (2009). Profile hidden markov models and metamorphic virus detection. *Journal in computer virology*, 5(2):151–169.

Biopython (2021). Biopython. https://biopython.org/. Online; accessed November 2021.

Buster (2021). Buster Sandbox Analyzer. https://bsa.isoftware.nl/. Online; accessed November 2021.

Damodaran, A., Di Troia, F., Visaggio, C. A., Austin, T. H., and Stamp, M. (2017). A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques*, 13(1):1–12.

Fu, W., Pang, J., Zhao, R., Zhang, Y., and Wei, B. (2008). Static detection of api-calling behavior from malicious binary executables. In *2008 International Conference on Computer and Electrical Engineering*, pages 388–392. IEEE.

Garg, V. and Yadav, R. K. (2019). Malware detection based on api calls frequency. In *2019 4th International Conference on Information Systems and Computer Networks (ISCON)*, pages 400–404. IEEE.

IMDEA Software Istitute (2013). Malicia. http://www.malicia-project.com/dataset.html. Online; accessed November 2021.

Kirat, D. and Vigna, G. (2015). Malgene: Automatic extraction of malware analysis evasion signature. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 769–780.

Mahmood, M. (2021). CallObfuscator: Obfuscate specific windows apis with different apis. https://github.com/d35ha/CallObfuscator. Online; accessed November 2021.

McKnight, J. (2017). *The evolution of ransomware and breadth of its economic impact*. PhD thesis, Utica College.

Pranamulia, R., Asnar, Y., and Perdana, R. S. (2017). Profile hidden markov model for malware classification—usage of system call sequence for malware classification. In *2017 International Conference on Data and Software Engineering (ICoDSE)*, pages 1–5. IEEE.

Ravi, S., Balakrishnan, N., and Venkatesh, B. (2013). Behavior-based malware analysis using profile hidden markov models. In *2013 International Conference on Security and Cryptography (SECRYPT)*, pages 1–12. IEEE.

RDocumentation (2021). Aphid. https://www.rdocumentation.org/packages/aphid/. Online; accessed November 2021.

Sandboxie (2021). Sandboxie classic. https://sandboxie-plus.com/sandboxie/. Online; accessed November 2021.

Sasidharan, S. K. and Thomas, C. (2021). Prodroid—an android malware detection framework based on profile hidden markov model. *Pervasive and Mobile Computing*, 72:101336.

Sathyanarayan, V. S., Kohli, P., and Bruhadeshwar, B. (2008). Signature generation and detection of malware families. In *Australasian Conference on Information Security and Privacy*, pages 336–349. Springer.

Sharif, M. I., Lanzi, A., Giffin, J. T., and Lee, W. (2008). Impeding malware analysis using conditional code obfuscation. In *NDSS*. Citeseer.

Singh, J. and Singh, J. (2018). Challenge of malware analysis: malware obfuscation techniques. *International Journal of Information Security Science*, 7(3):100–110.

Srivastava, A., Lanzi, A., and Giffin, J. (2008). System call api obfuscation. In *International Workshop on Recent Advances in Intrusion Detection*, pages 421–422. Springer.

Stamp, M. (2017). *Introduction to machine learning with applications in information security*. Chapman and Hall/CRC.

Suenaga, M. (2009). A museum of api obfuscation on win32. *Symantec Security Response*.

The Enigma Protector Developers Team (2021). Enigma Protector. https://www.enigmaprotector.com/. Online; accessed November 2021.

Uppal, D., Sinha, R., Mehra, V., and Jain, V. (2014). Malware detection and classification based on extraction of api sequences. In *2014 International conference on advances in computing, communications and informatics (ICACCI)*, pages 2337–2342. IEEE.

Vemparala, S. (2015). Malware detection using dynamic analysis. Master's thesis.

Vemparala, S., Di Troia, F., Visaggio, A. C., Austin, T. H., and Stamp, M. (2016). Malware detection using dynamic birthmarks. In *Proceedings of the 2016 ACM on international workshop on security and privacy analytics*, pages 41–46.

Wadkar, M., Di Troia, F., and Stamp, M. (2020). Detecting malware evolution using support vector machines. *Expert Systems with Applications*, 143:113022.

You, I. and Yim, K. (2010). Malware obfuscation techniques: A brief survey. In *2010 International conference on broadband, wireless computing, communication and applications*, pages 297–300. IEEE.