

**SE
TU**

Ollscoil
Teicneolaíochta
an Oirdheiscirt

South East
Technological
University

CMD Obfuscation Detection Tool

Final Report

Student: Melanie Dudas

Student Number: C00253245

Supervisor: Dr Hisain Elshaafi

Abstract

This document outlines the final product of the 4th year Cybercrime and IT Security project developed to improve an organisation's threat detection capabilities. The product aims to detect potential threats early to limit damage and prevent further malicious activity.

The document includes details on the product development process, key features, and benchmarking of three machine learning algorithms with the highest accuracy.

The product runs entirely in CMD and is designed for use by Security Operations. Its primary function is to identify the execution of obfuscated commands in CMD, providing an organisation with visibility into previously undetectable indicators of compromise.

Table of Contents

Abstract.....	2
Table of Contents.....	3
Table of Figures.....	4
1. Introduction	5
2. Final Product	6
2.1. Features	6
2.2. Future Work.....	9
3. Machine Learning Component.....	10
3.1. Dataset	10
3.2. Text Pre-processing.....	10
3.3. Benchmarking Algorithms.....	11
3.3.1. Accuracy.....	11
3.3.2. Speed and resource usage	12
4. Testing.....	13
5. Learning Outcomes	14
5.1. Technical	14
5.2. Personal	14
6. Conclusion and Project Review.....	15
7. Acknowledgements.....	16

Table of Figures

Figure 1 Main Menu.....	6
Figure 2 Classification of manually entered commands.....	7
Figure 3 Classification of executed commands.	8
Figure 4 Highest-scoring Convolutional Neural Network model.....	11
Figure 5 Highest-scoring Gradient Boosting model.....	11
Figure 6 Highest-scoring Support Vector Machine model.....	11
Figure 7 Support Vector Machine - command classification time.....	12
Figure 8 Convolutional Neural Network - command classification time.....	12
Figure 9 Gradient Boosting algorithm – command classification time	12

1. Introduction

Command obfuscation is a significant threat capability that can be challenging to detect. The rules and alerts organisations have in place to detect specific commands fail to work when the commands are obfuscated, even if only one character has been added. While two machine learning tools have already been developed to detect command line obfuscation, they focus on non-synonymous command lines that do not appear obfuscated in logging tools like Sysmon. As a result, an organisation's existing rules and alerts for specific command and argument pairings will still be effective against these types of obfuscated commands.

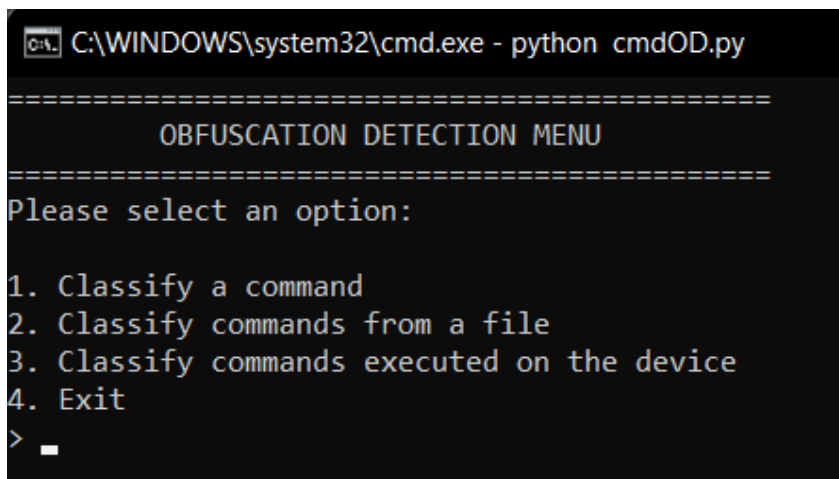
However, obfuscation in synonymous commands poses a more significant risk to organisations as the command remains obfuscated even in the logging tool, making detection and response more challenging. To address this issue, this product was developed to identify the execution of synonymous commands. The product is designed to inform Security Operations when an obfuscated command has been executed, allowing them to investigate and respond accordingly.

It is important to note that the tool was not designed to prevent the execution of obfuscated commands or to provide details on the nature of the obfuscated command. Instead, its primary purpose is to alert Security Operations of potentially malicious activity, as there is no legitimate reason for obfuscating commands.

This document provides detailed information on product development and the challenges faced. It details the creation of a custom dataset, the implementation of three different machine learning algorithms (SVM, Convolutional Neural Network (CNN), and Gradient Boosting Classifier (GBC)), and a comparison of their performance metrics. It also discusses the tool's features, system resource usage, and benchmarking of the algorithms. Furthermore, the document outlines the testing process and the learning outcomes from the project and highlights the project's successes and limitations. Suggestions for future improvements are also provided.

2. Final Product

The project's final product uses the Support Vector Machine (SVM) algorithm, also known as SVM. Although the SVM algorithm did not achieve the highest score among the three tested algorithms, its faster processing time compared to the highest-scoring algorithm, the Convolutional Neural Network (CNN), compensates for the difference. Figure 1 shows the tool's Main Menu, the first screen that appears when the tool is launched. The Main Menu provides an overview of the tool's features and allows users to select the option they want.



```
C:\WINDOWS\system32\cmd.exe - python cmdOD.py
=====  
OBFUSCATION DETECTION MENU  
=====  
Please select an option:  
1. Classify a command  
2. Classify commands from a file  
3. Classify commands executed on the device  
4. Exit  
> _
```

Figure 1 Main Menu

2.1. Features

The first feature of the tool is the classification of manually entered commands. Upon selecting this option, the user is prompted to enter the desired command, and the tool will output its classification. Figure 2 illustrates the user interface for this feature and displays how the tool presents the classification.

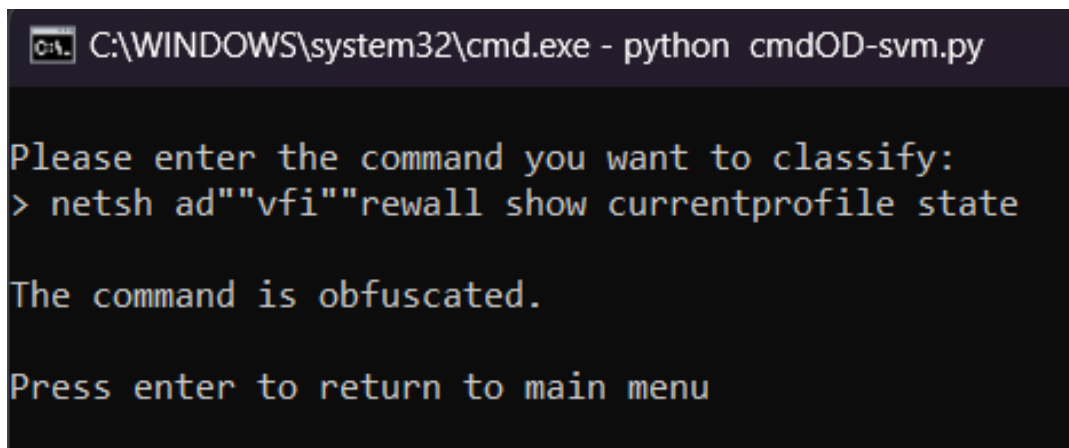
The second available feature is the classification of a larger number of commands from a file. Upon selecting this option, the user provides the path to the file, and the tool examines the commands contained in the file.

The third feature focuses on classifying commands executed on the device by integrating Sysmon and PowerShell into the tool. Sysmon is used for logging all executed commands on the device. To achieve this, the tool is configured to monitor Process Creation events without ignoring any events, even though professional environments might filter out noisy events. This approach ensures a larger number of commands are captured for testing and accommodates companies that may record all events.

PowerShell plays a crucial role in gathering the Process Creation events logged by Sysmon by extracting the command line part and writing each command to a file for classification. The script first checks if it runs with administrator privileges and relaunches with elevated permissions if

necessary. It then retrieves the last recorded timestamp from a file. If there is no timestamp, it sets the start time to the minimum possible value, assuming that the program was never before executed on the device and that all commands need to be classified. The purpose of the timestamp file is to keep track of the starting point for classifying commands. This ensures that the tool does not have to classify all commands every time the script is run, significantly reducing the processing time and resources required for classification. The tool can efficiently focus on classifying new commands executed since the last run by maintaining a record of the previously processed commands through the timestamp file.

Next, the script filters Sysmon events with ID 1 (Process Create) and extracts the CommandLine property (index 10) from each event. These commands are saved to a file, and the new timestamp is stored in the timestamp file for future reference. After PowerShell completes its execution, the tool individually classifies each command and reports the number of obfuscated and unobfuscated commands.



```
C:\WINDOWS\system32\cmd.exe - python cmdOD-svm.py

Please enter the command you want to classify:
> netsh ad""vfi""rewall show currentprofile state

The command is obfuscated.

Press enter to return to main menu
```

Figure 2 Classification of manually entered commands.

```
=====
OBFUSCATION DETECTION MENU
=====
Please select an option:

1. Classify a command
2. Classify commands from a file
3. Classify commands executed on the device
4. Exit
> 3
5505 commands processed, 2 obfuscated, 5503 not obfuscated

Average time taken for classification: 0.001176 seconds

Obfuscated commands:
regsvr32 /s /u %i:inputs\file.sct scrobj.dll
arp -%

Press enter to return to main menu
```

Figure 3 Classification of executed commands.

2.2. Future Work

At the beginning of the project, the tool was envisioned to be able to run in the background, automatically collect all executed commands, and classify them without user intervention. The tool was intended to alert an authorised user via SIEM or email if an obfuscated command was detected. However, that feature was not developed due to time constraints and the difficulty of developing a background process that would not expose the device to additional issues.

To overcome this challenge, an alternative plan was devised that resulted in the third feature of the tool, which focuses on classifying executed commands. This approach requires a user to run the tool on the device instead of running automatically in the background. Because an authorised user is already using the tool, an alert is unnecessary.

While the tool does not currently run in the background, it still is able to classify already executed commands on a device which is a great alternative for detecting potential threats and proving that already executed obfuscated commands can be detected. In future project iterations, the current code for gathering commands from Sysmon with PowerShell should be utilised to create the program as a background process.

3. Machine Learning Component

3.1. Dataset

One of the primary challenges faced during this project was the lack of an available real-world dataset consisting of executed commands from a professional environment. As a result, a custom dataset was created containing both obfuscated and unobfuscated commands. Constructing this dataset proved to be time-consuming, as obfuscation could not be automated due to its varying nature and the acceptance of different obfuscation types by different programs. To ensure the dataset's realism, all commands were obfuscated manually.

This inability to source a pre-existing dataset led to two key issues. First, the dataset comprised only 850 commands, which is smaller than typical dataset sizes. Second, it is important to note that the unobfuscated commands were also manually created and may need to be more representative of commands encountered in professional settings. This may affect the model's accuracy in real-world scenarios, as it was specifically designed for use in professional environments. Further verification and testing with real-world data should further improve the model's accuracy and ensure its effectiveness.

3.2. Text Pre-processing

Three different machine learning algorithms were implemented for command classification: Support Vector Machine (SVM), Convolutional Neural Network (CNN), and Gradient Boosting Classifier (GBC). Each model underwent different data pre-processing.

For the SVM model, the text data was pre-processed using CountVectorizer with character-level n-grams ranging from 1 to 3. This method encodes the text by counting the frequency of n-grams in the text, which provides the feature vectors for the SVM model. That was done to try and capture patterns and structures within the text. The data was then split into training and testing sets, and the SVM model was trained on the vectorised data.

In the CNN model, the Tokenizer class was utilised at the character level to tokenise and encode the text data. The program also included an additional feature, the number of special characters in the command. The encoded text data was then concatenated with the additional feature and split into training and testing sets. The CNN model included a Conv1D layer to learn patterns in text, MaxPooling1D to select important features, Dense layers to combine all features, and a Dropout to prevent overfitting. The model was then trained on the reshaped input data.

The Tokenizer class was again used at the character level for the GBC model to tokenise and encode the text data. However, this model included two features: entropy and the ratio of special characters in the command. The encoded text data was concatenated with these features and split into training and testing sets. The GBC model was then trained on that data.

3.3. Benchmarking Algorithms

3.3.1. Accuracy

The CNN model achieves the highest F1 score, which indicates a superior balance between precision and recall. This suggests that the CNN model is more effective in identifying true positives while minimising false positives and false negatives.

The SVM model demonstrates a slightly higher F1 score than the GBC model and has a better recall. This implies that the SVM model is more successful in detecting true positives but may produce more false positives than the GBC. However, the SVM model's overall performance still falls short of the CNN model.

In contrast, the GBC exhibits the highest precision among the three models but suffers from the lowest recall. This indicates that the GBC is more conservative in predicting positive cases, which may lead to a higher rate of false negatives. Despite having a higher accuracy than the SVM model, the GBC model's overall performance, as measured by the F1 score, is lower than that of the SVM and CNN models, suggesting that it may not be as well-suited for this specific task.

Based solely on these metrics, the CNN model appears to be the most effective choice for this detecting obfuscation. The SVM model offers a reasonable alternative with slightly lower performance, while the GBC model's lower recall makes it less desirable for this task.

```
F1-Score: 0.9215686274509803
Accuracy: 0.9503105590062112
Precision: 0.9591836734693877
Recall: 0.8867924528301887
```

Figure 4 Highest-scoring Convolutional Neural Network model

```
Accuracy: 0.9216867469879518
Precision: 0.9512195121951219
Recall: 0.78
Final F1-Score: 0.8571428571428571
```

Figure 5 Highest-scoring Gradient Boosting model

```
Accuracy: 0.9036144578313253
Precision: 0.9310344827586207
Recall: 0.8181818181818182
F1-Score: 0.8709677419354839
```

Figure 6 Highest-scoring Support Vector Machine model

3.3.2. Speed and resource usage

It is essential to consider the time and system resource usage associated with each algorithm. Let's compare the time each model would take to process 10,000 commands to provide a more tangible comparison.

The GBC model shows an average classification time of 4.13 seconds for 10,000 commands. The SVM model demonstrates a slightly longer classification time per command, resulting in around 5.37 seconds for 10,000 commands. GBC and SVM models have the same system resource usage, with the highest CPU usage reaching 40%. The main increase in CPU usage for these models is attributed to PowerShell gathering commands. More details on the resource usage for SVM can be found in the Testing section of the report.

On the other hand, the CNN model shows significantly higher classification times, averaging 0.149982 seconds per command. This leads to a considerably longer processing time for 10,000 commands, taking approximately 24.997 minutes. Furthermore, the CNN model shows substantially higher system resource usage, with CPU reaching 100%, memory at 92%, and disk usage at 2%.

CNNs are known to be computationally intensive compared to algorithms like SVM and GBC. This is because CNNs are designed to process grid-like data, such as images, and their architecture consists of multiple layers. The multi-layer structure adds to the computational complexity of CNNs, slowing it down. Additionally, CNNs rely heavily on matrix operations like convolutions, which are computationally expensive. (IBM, n.d.) In contrast, algorithms like SVM and GBC do not involve such extensive matrix operations, resulting in shorter processing times.

In conclusion, even though the CNN model provides superior performance in terms of balancing precision and recall, it is accompanied by significantly increased classification times and system resource usage compared to the GBC and SVM models. Considering these factors is crucial when choosing the most appropriate model for the task for a real-world application.

```
2024 commands processed, 0 obfuscated, 2024 not obfuscated
Average time taken for classification: 0.000537 seconds
```

Figure 7 Support Vector Machine - command classification time

```
2024 commands processed, 0 obfuscated, 2024 not obfuscated
Average time taken for classification: 0.149982 seconds
```

Figure 8 Convolutional Neural Network - command classification time

```
2024 commands processed, 0 obfuscated, 2024 not obfuscated
Average time taken for classification: 0.000413 seconds
```

Figure 9 Gradient Boosting algorithm – command classification time

4. Testing

This project was tested in multiple stages to ensure the tool was functioning as intended. Initially, testing focused on validating individual functions within the Python script. Tests were also conducted to confirm that PowerShell was correctly elevating privileges and collecting commands from Sysmon while Python processed them as intended.

Tests were conducted using a laptop with the following specifications: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx (2.10 GHz), 8 GB DDR4 RAM, 256 GB SSD, AMD Radeon Vega 8 Graphics, running Windows 11 Home (64-bit). The software environment included Python 3.9.15, Keras 2.10.0, scikit-learn 1.1.3, Pandas 1.5.2, and NumPy 1.23.4.

The tool's start-up time was measured, and it was found that it takes between 2 and 4 seconds to initialise. The system was also tested with a large number of commands to determine its performance under stress, and it was found that the number of commands processed did not significantly impact the tool's performance.

To test the tool's impact on system resources, the following measurements were made by taking an average after 5 minutes of observation:

Idle state:

- CPU usage: 6%
- Memory usage: 68%
- Disk usage: 1%

Typical activities state (web browsing, office applications, etc.):

- CPU usage: 50%
- Memory usage: 88%
- Disk usage: 10%

Normal operation of the tool (tool running and classifying one command):

- CPU usage: 15%
- Memory usage: 87%
- Disk usage: 1%

Peak operation of the tool (getting commands from Sysmon and classifying a large number of them):

- CPU usage: 40%
- Memory usage: 88%
- Disk usage: 2%

Comparing the baseline testing results with the resource usage while running the tool, it can be concluded that the tool has a relatively low impact on system resources. During normal operation, the tool decreases CPU usage by 35% and maintains similar memory and disk usage compared to the typical activities state. During peak operation, the tool reduces CPU usage by 10% compared to the typical activities state while keeping memory usage at the same level. The disk usage increases by only 1% compared to the idle state. The testing device has high memory usage during typical activities, and the tool does not significantly increase resource consumption, indicating that it operates efficiently within the available system resources.

5. Learning Outcomes

In this project, I achieved several technical and personal learning outcomes which will help me in my future career.

5.1. Technical

From a technical standpoint, I gained proficiency in Python, a programming language I had not used previously. My experience was limited to C++ and Java. Additionally, I had the opportunity to learn about machine learning, specifically Convolutional Neural Networks (CNNs), Support Vector Machines (SVMs) and Gradient Boosting Classifier. I used Keras and Scikit Learn to construct various machine learning models to classify obfuscated commands successfully.

Moreover, I learned how to pre-process and handle text data, load datasets, engineer features, and divide data into training and testing sets. My project also entailed writing and executing PowerShell scripts to extract data from Sysmon logs and relaunch scripts with administrator privileges. Ultimately, I successfully integrated the machine learning model, Python script, PowerShell and Sysmon, allowing for the classification of commands executed on a device.

5.2. Personal

On a personal level, I improved my time management skills by allocating appropriate time to various project stages, including research, implementation, evaluation, and testing. Adaptability also played a significant role in my project, as I learned new concepts, programming languages, and tools to address project challenges.

Lastly, I had the opportunity to improve my writing skills by creating all the necessary documentation for this project. This was a valuable experience, as English is not my first language.

6. Conclusion and Project Review

During the project, several aspects contributed to the overall success. Despite not having prior knowledge of machine learning, I successfully implemented various machine learning models. Furthermore, I employed three different algorithms and adjusted each to achieve an F1 score above 0.85. Finally, I developed a working product capable of classifying obfuscated commands executed on a device.

On the other hand, I had an unexpected challenge of finding a suitable dataset led to creating a new dataset from scratch. While the resulting dataset may not fully capture the range of commands executed in a professional environment, it is a starting point that can be refined with further feedback and usage. With the potential to be integrated into anti-malware products or other security tools, the tool has the opportunity to evolve and improve over time.

Regarding the potential improvements of the project, one is that the tool does not run automatically in the background. As a result, it does not alert users via SIEM or email when obfuscated commands are detected. However, this presents an opportunity for the tool's future development, and implementing this feature could be an excellent direction for further work.

If I were to approach this project again, I would allocate more time to creating the dataset or finding a company willing to share their employee's command history with me to develop this tool. Furthermore, I would invest more time researching the complexity of background processes and understanding their potential associated challenges, enabling the development of a more advanced and effective tool for detecting obfuscated commands.

Rather than spreading my efforts across multiple algorithms, I would concentrate on a single algorithm, choosing SVM or GBM for their speed. I would work diligently to optimise the algorithm and achieve an even better F1 score, aiming as close to 1 as possible. Alternatively, if they cannot surpass their current highest F1 score, I would focus entirely on developing the tool as a background process. In that case, I'd suggest perfecting the CNN model, as speed is less of a concern if commands are classified as they are executed rather than processing thousands of them all at once. This focused approach would likely result in a more efficient and effective tool.

Regarding technology choices, using Python, Keras, and the Anaconda environment for implementing machine learning models was the right choice due to their versatility and extensive community support.

In conclusion, the project can be considered a success because it achieved its primary goal of detecting obfuscated commands in the CMD environment. I believe the overall results make this project a valuable contribution to cybersecurity.

7. Acknowledgements

I want to thank my project supervisor Hisain Elshaafi for the guidance and support throughout the project. His expertise and insights helped shape the project's direction and ensure its success.

I would also like to thank the other lecturers at SETU who provided valuable feedback and support during this project and the entire course. Their contributions and encouragement were greatly appreciated.

Declaration of Plagiarism



I declare, this document in this submission in its entirety is my own work except for where duty acknowledged. I have cited the sources of all the quotations, paragraphs, summaries of information, tables, diagrams, or other material. This includes software and other electronic media that is integral property rights may reside. I have provided the complete biography at the end of my document detailing all the works and resources used in the presentation of this submission. I am aware that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offense.

Student	Melanie Dudas – C00253245
Project Supervisor	Hisain Elshaafi
Institution	South East Technological University
Title	CMD Obfuscation Detection Tool - Final Report
Submission Date	17/04/2023

Melanie Dudas

Signature