CMD Obfuscation Detection Tool

Functional Specification

Student: Melanie Dudas
Student Number: C00253245
Supervisor: Dr Hisain Elshaafi

Table of Contents

Table of Figures

Introduction

As stated in the accompanying research document, relying on traditional-style detections does not provide the ability to detect command-line obfuscation. For that reason, this project aims to develop a machine-learning tool for command-line obfuscation.

The purpose of this document is to provide a functional specification for the CMD Obfuscation Detection tool. It will include detailed information about the tool's design, capabilities, and how users can interact with it.

Project Scope

To produce a standalone CMD Obfuscation Detection tool that will accurately classify commands so that an authorised person can review the activity and respond accordingly if obfuscation is detected.

The tool must be able to accept user input, classify that input and output the classification results back to the user. The tool should also be able to run in the background, automatically fetch commands executed on endpoints and alert the authorised person in case of obfuscation.

Assumptions and Constraints

Project assumptions are things believed to be true while the project constraints are limitations imposed on the project. As the project is being developed, assumptions and constraints may be re-analysed and refined.

The primary assumption regarding this project is that Sysmon service is installed on all systems where obfuscation wants to be detected. Another assumption is that the tool will run on a system which is never powered off so that the tool can run constantly.

The main constraint is that the project must be finalised by April 2023. Because of the time constraint, the primary goal of this project will be to get the tool to work with manual input. If there is enough time, the goal would be to get the tool to run in the background and automatically fetch each executed command.

System Architecture

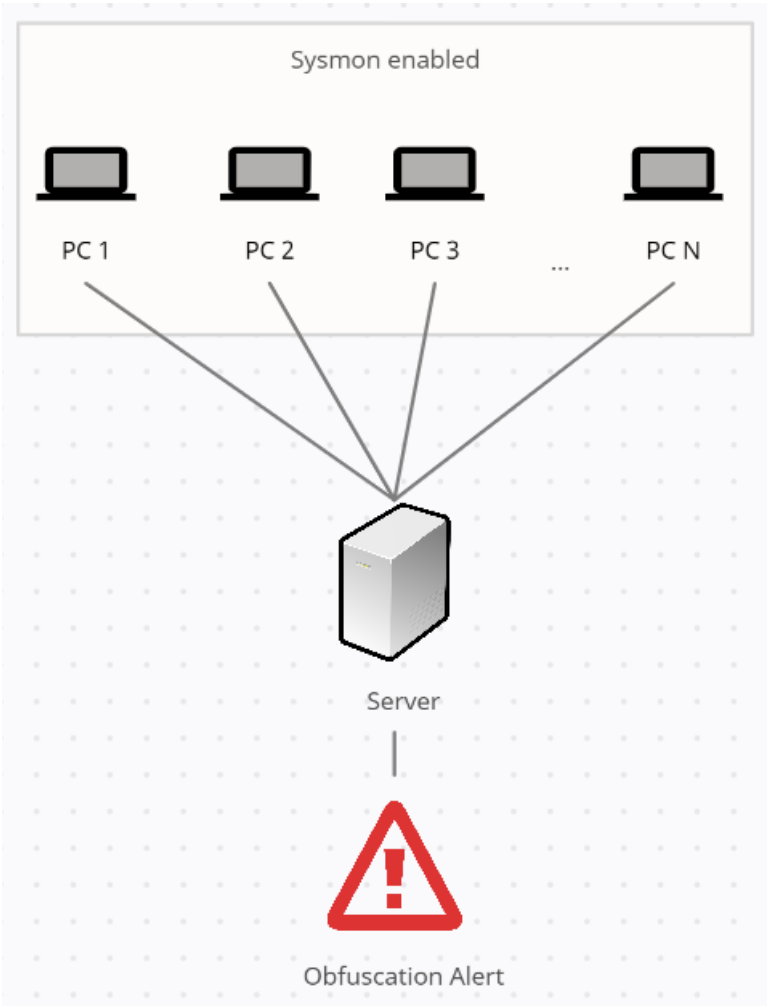

Figure 1 *System Architecture diagram*

Figure 1 presents the system architecture in which the tool can automatically fetch executed commands. All the PCs which need monitoring for CMD obfuscation must be running Sysmon service. Sysmon is then able to collect all executed commands and send each one to the server. The server keeps track of all commands and the information about the device where they were executed. The CMD Obfuscation detection tool runs on that same server. If obfuscation is detected, the server sends an alert to an authorised user containing information about the command and device details.

Use Case

A context diagram is a high-level diagram that shows how external entities interact with a system. It is frequently drawn as a single process that summarises the entire system.

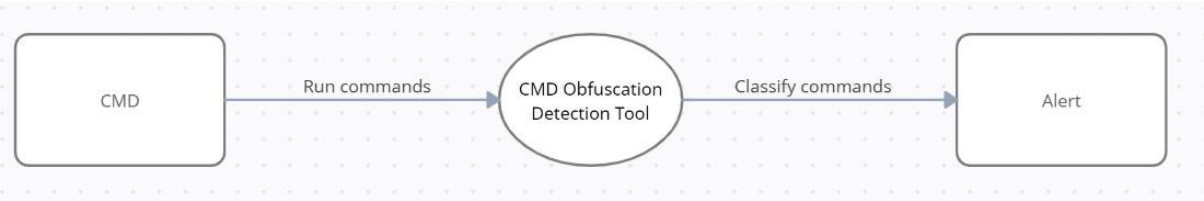

Figure 2 *CMD obfuscation detection tool - Context diagram*

As can be seen in Figure 1, CMD is the external entity from where the data flow starts. Once a command is run in CMD, it goes through the CMD Obfuscation Detection tool to be examined for obfuscation. If obfuscation is detected, the tool alerts an authorised person or people. The alert could be sent to SIEM software or by email.

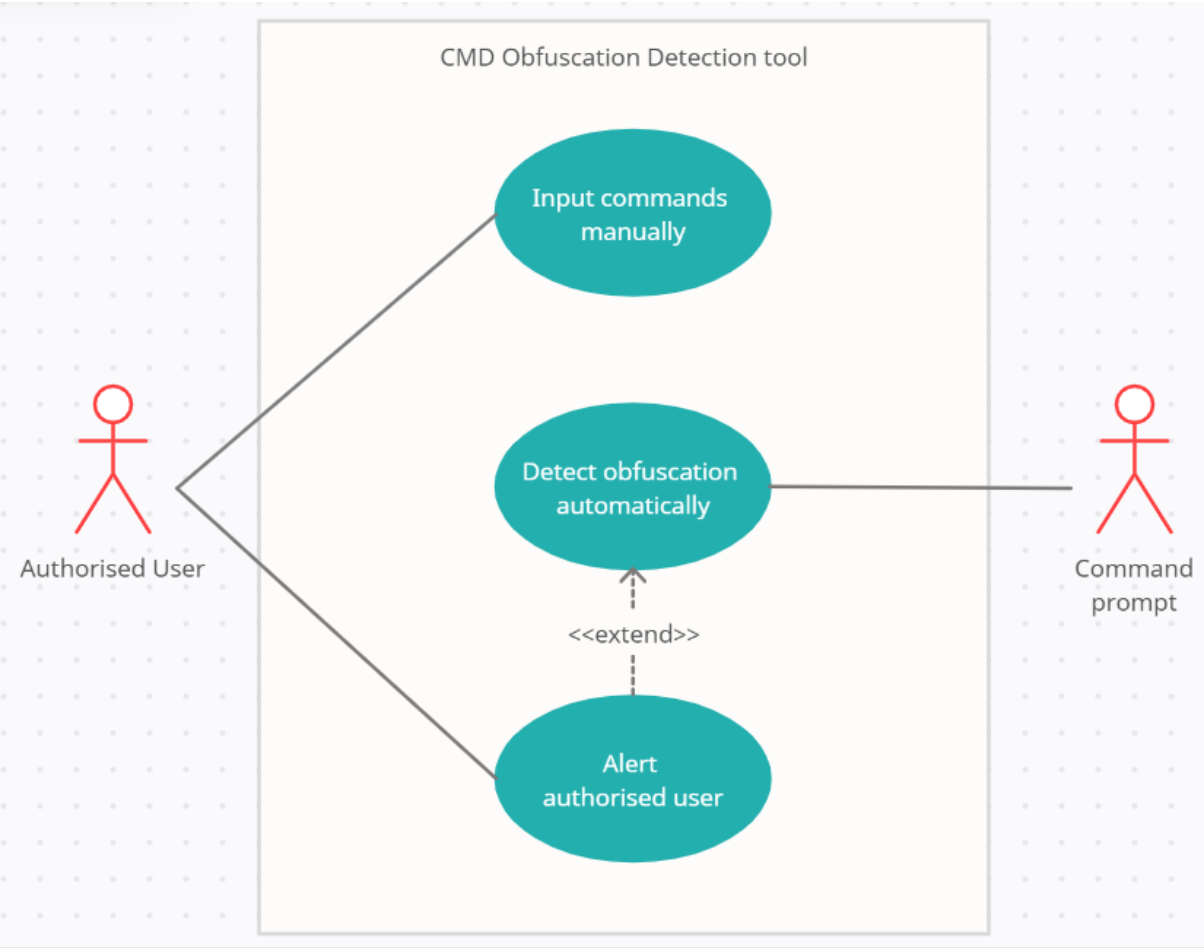

Figure 3 *CMD obfuscation detection tool - Use Case diagram*

Figure 2 presents a Use Case diagram which shows more information than the Context diagram. Two types of actors are displayed, an authorised user, e.g. security employee, and a user, which can be a malicious actor or non-security employee. There are only three features of the tool, manual input of commands for detection, automatic detection from CMD and alert function.

Use Case scenarios

In this section, the features of the app will be described in more detail.

Input commands manually

Name: Input commands manually

Actors: Authorised user

Description: An authorised user wants to input a command manually into the tool to see the classification output. The user chooses the option for manual input, inputs the command, and the result is outputted back to the user.

Main success scenario:

1. The user is able to select the option for manual input.
2. The tool accepts the input for classification.
3. The tool outputs the results to the user immediately.

Detect obfuscation automatically

Name: Detect obfuscation automatically

Actors: Command prompt / Authorised user

Description: A user or an authorised user is running commands in CMD. Each of the commands is sent to the tool for classification.

Main success scenario:

1. The tool is able to get every command that was executed in CMD.
2. The tool is able to classify each of the commands as they come.

Alert authorised user

Name: Alert authorised user

Actors: Authorised user

Description: Obfuscation is detected in one of the commands run in CMD. The tool sends an alert to the authorised user.

Main success scenario:

1. The tool is able to send an alert via email or to SIEM software.
2. The alert has detailed information about the command in question and the device from which the command originated.

Requirements

Functionality

#	Function	Description	Criticality	Dependency
1	Manual input	Accept manually inserted commands	High	
2	Fetch command	Automatically fetch commands	Medium	
3	Classify	Classify commands	High	#1 / #2
4	Output prediction	Output classification result directly to user	High	#1, #3
5	Alert	Send alert if obfuscation is detected	Medium	#2, #3

Usability

Usability refers to the users of the program and their interaction with it. It should be easy for users to input the commands if they wish to do so manually, and the output should be easy to understand. It should also be easy for users to find and select the option for manual input.

Reliability

The tool should have at least 95% reliability in relation to crashes. It must not create vulnerabilities in the system. Load testing will be performed to see how the tool would behave in real-world conditions.

Performance

To function, the tool must use as few resources as possible. The tool must be fast enough to be able to examine each of the commands as soon as they are executed. It should take around 10 seconds for the tool to start up.

Supportability

The tool must be able to work on all modern Windows operating systems. The installation of the tool must be an easy process with little user involvement necessary. The tool must be able to function without internet connection.

Metrics

Metrics in this project refer to the methods used to evaluate the tool's effectiveness. The following are the key metrics that should be used to assess the project's success:

- ✓ The tool must be able to run on Windows Operating Systems and should be compatible with Windows Server
- ✓ The tool is easy to setup
- ✓ The tool is easy to use manually
- ✓ Commands can be inputted manually into the tool
- ✓ The tool can use the machine learning algorithm to classify the commands accurately
- ✓ Accuracy results of this tool are the same or higher than those of the existing tools
- ✓ The tool can produce classification output
- ✓ The tool must display output in a comprehensible manner
- ✓ The tool can automatically get executed commands via Sysmon
- ✓ The tool can send obfuscation alerts containing information about the command run and device details
- ✓ All non-functional requirements must be achieved

Project Plan

Plan	Due Date	Deliverable
Submit Research Document	25/11/2022	Finish Research document draft and submit to Blackboard.
Submit Functional Specification Document	16/12/2022	Finish Functional Specification document draft and submit to Blackboard.
Test different ML algorithms	31/12/2022	Test three different ML algorithms to find the most accurate one.
Accept input Function	10/01/2022	Give the tool functionality of accepting user input.
Classify Function	17/01/2023	Give the tool functionality of classifying user input.
Output Function	24/01/2023	Give the tool functionality of classifying user input.
Fetch command Function	14/02/2023	Create a way for the tool to be able to fetch every executed command.
Alert Function	24/02/2023	Give the tool ability to alert authorised user when obfuscation is detected.
Reserved	10/03/2023	Finish any outstanding work and allow time for final changes.
Testing	10/04/2023	Test the tool's performance, test for any security issues, load testing.
Project Document	21/04/2023	Finish Project document and submit to Blackboard.

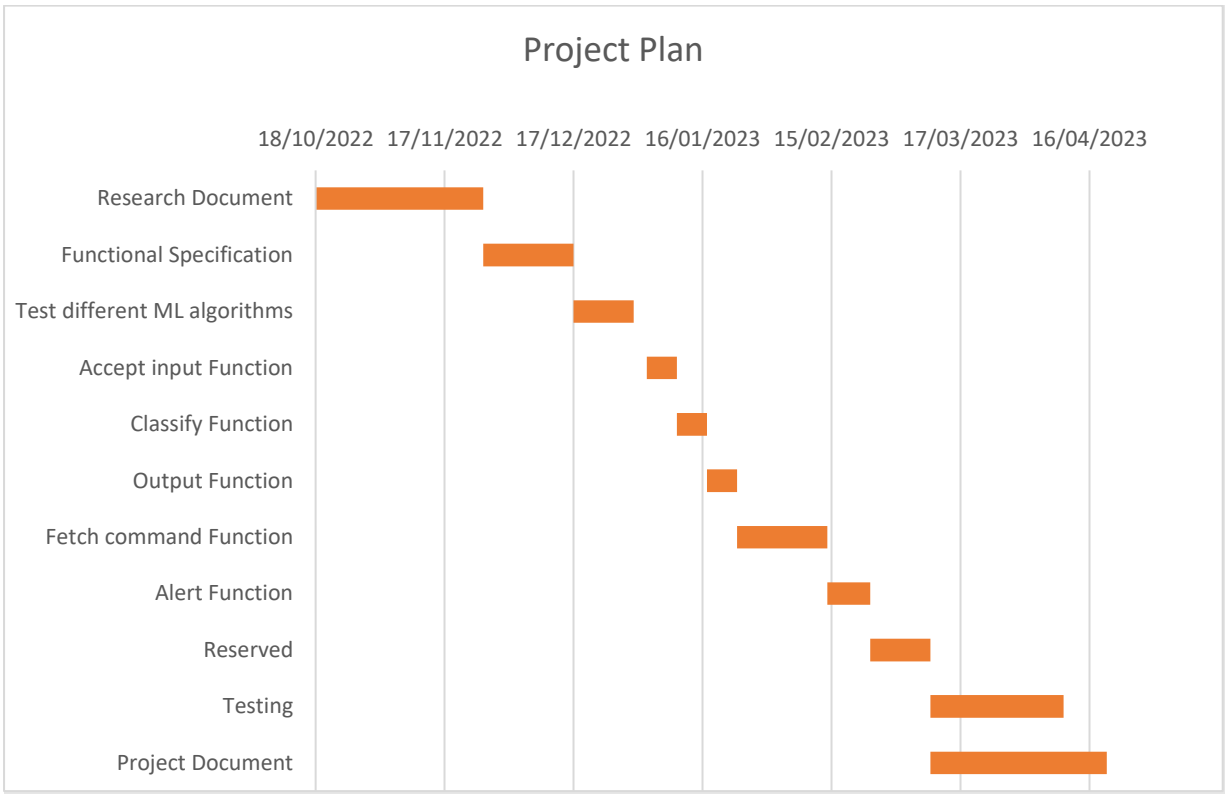

Figure 4 *Project plan - Gantt Chart*

Precedent

My work placement served as a major precedent for this project. During my role as a Security Operations intern, the team tested the company's detection ability. It was noticed that neither the company nor the external parties were able to detect an otherwise highly detectable activity if the commands were obfuscated.

One of the most critical functions performed by Security Operations analysts is threat detection and response. When monitoring tools issue alerts, the analysts investigate them and, in case of a true positive, they must respond quickly by, for example, shutting down or isolating an endpoint. (Trellix, n.d.) Suppose command line obfuscation makes all company tools incapable of detecting suspicious activity. In that case, it means the analysts are also unaware that the activity has occurred, thus preventing them from doing their job.

Related Work

As already stated in the Research Document, two machine-learning tools have been developed for detecting Command Line obfuscation. Only the tool developed by Adobe is open sourced and can be evaluated or benchmarked against this project's tool once it is developed.

There are three main disadvantages of Adobe's tool. Firstly, it has been built to detect only the obfuscation techniques presented in the DOSfuscation paper (Bohannon, 2018) researched by Daniel Bohannon. (Tang, 2021) Although Bohannon extensively researched Command Line obfuscation, researcher Weitze Beukema furthered the research by investigating synonymous command lines, which are capable of staying obfuscated even in the recorders used by detection software. (Beukema, n.d.) That means Adobe's solution is capable of detecting many obfuscation techniques; however, it is much less capable of detecting the primary concern when it comes to Command Line obfuscation.

Secondly, Adobe tested only the Convolutional Neural Network algorithm and implemented it in the tool. Convolutional Neural Network, or CNN, is primarily used for analysing visual imagery, but it has also shown promising results in natural language processing. However, regarding obfuscation classification, CNN has produced less accurate results in both PowerShell and CMD obfuscation.

Thirdly, Adobe's tool allows only manual input. In order to use Adobe's tool for obfuscation classification, each of the commands needs to be inputted manually into it, and the classification results can be read in the tool's output. Considering a company can have hundreds or even thousands of commands run daily, using Adobe's solution could be impossible. That is why this project will aim to make a tool able to fetch the commands as they are executed and classify them automatically.

References

Beukema, W. (n.d.). *Windows Command-Line Obfuscation*. Retrieved October 23, 2022, from WietzeBeukema: https://www.wietzebeukema.nl/blog/windows-command-line-obfuscation

Bohannon, D. (2018, March 22). *DOSfuscation: Exploring the Depths of Cmd.exe Obfuscation and Detection Techniques*. Retrieved November 06, 2022, from Mandiant: https://www.mandiant.com/resources/blog/dosfuscation-exploring-obfuscation-and-detection-techniques

Tang, W. (2021, August 24). *Using Deep Learning to Better Detect Command Obfuscation*. Retrieved November 06, 2022, from Adobe Tech Blog: Using Deep Learning to Better Detect Command Obfuscation

Trellix. (n.d.). *What Is a Security Operations Center (SOC)?* Retrieved December 11, 2022, from Trellix: https://www.trellix.com/en-us/security-awareness/operations/what-is-soc.html