Masaryk University
Faculty of Informatics

Clustering of Command Histories from Cybersecurity Training

Bachelor's Thesis

Daniel Popovič

Brno, Fall 2020

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Daniel Popovič

Advisor: RNDr. Valdemar Švábenský

Acknowledgements

I would like to thank my friends and family for supporting me. Great deal of thanks also belongs to my supervisor RNDr. Valdemar Švábenský for his invaluable help during my work on this thesis. I would also like to thank my consultant RNDr. Jaroslav Čechák.

Abstract

Cybersecurity training is an effective way for students to improve their cybersecurity skills and expand their knowledge. While assessing their progress may reveal valuable insights, it is often a challenging task. Data mining provides a multitude of useful techniques for analyzing data gathered during cybersecurity training. In this thesis, we applied clustering analysis to discover similarities between trainees' approaches to the training, frequent mistakes, and their progress through different stages. We used 5,747 shell-command logs gathered from 73 trainees during 17 training sessions. As we analyzed and visualized the resulting clusters, we discovered that while most of the trainees chose similar approach, only a minority of them completed the training. To gain further insight, we also interpreted clusters of successful trainees, discussed different reasons why many trainees failed, and presented alternative training approaches. The results we gathered can help instructors to better support struggling trainees and improve training design.

Keywords

clustering, education, educational data mining, data mining, data analysis, security, Python, KYPO, Google Colab

Contents

List of Figures

1 Introduction

With the growing economical and societal dependence on computer systems, new threats appear, with attackers trying to exploit cybersecurity vulnerabilities [15]. To successfully counter these emerging threats, we need skilled cybersecurity professionals. However, educating students into becoming cybersecurity experts can be challenging as it requires intricate knowledge of the field.

Cybersecurity training allows trainees to gain practical experience while discovering new tools. The training often includes the usage of command-line tools, which may be recorded in command history logs. Data stored in these logs contain valuable educational insights, as they capture details about trainees' progress. Insights from the captured data allow instructors to identify challenging parts of the training, design new training tasks, or provide better feedback. However, if there is a large amount of data, or they are saved in an format impractical for manual evaluation, gaining insights from them may require more advanced methods.

Clustering provides one such method. It forms groups of data based on similar characteristics [27], allowing us to discover distinct trainee groups based on their behavior during the training.

This thesis aims to explore and identify different approaches the trainees took while solving cybersecurity tasks. We introduce several relevant clustering methods and select the one suitable in our setting. Then we transform the trainees' data to apply the clustering algorithm. After using the chosen algorithm, we visualize the results and interpret the clusters we found.

We divided this thesis into six chapters. In Chapter 2, we explain concepts relevant to this thesis and provide the background necessary for understanding later chapters. Chapter 3 follows by presenting the related work. In Chapter 4, we describe the data format we created, the methods we used for data preprocessing, and the clustering algorithm's application. Next, in Chapter 5, we present, visualize, and interpret the resulting clusters. We also discuss notable outliers. In Chapter 6, we conclude our work and outline the possible directions for future research.

2 Theoretical background

This chapter, we provide the theoretical knowledge needed for understanding this thesis. In Section 2.1, we describe data mining techniques and Educational Data Mining(EDM). In Section 2.2, we focus on clustering, different clustering types, and present three clustering algorithms. In Section 2.3, we describe similarity measures.

2.1 Educational data mining

Data mining is the process of information extraction and pattern discovery from data. It includes pre-processing the data, applying a selected data mining algorithm, and interpreting the results. The most prominent data mining techniques are classification, clustering, association rule mining, and regression [13].

EDM is an interdisciplinary field that applies data mining methods on data from the educational environment. These data come from different sources, such as e-learning and educational software, student interviews, and written tests. The main objective of EDM is to improve learning. EDM is used to help educators in understanding students, to improve courses and educational processes in general, to recommend suitable tasks to students, to model students, and to predict their behavior [27].

2.2 Clustering

Clustering is the process of assigning data into groups based on how similar they are to each other. Data in one group tend to be similar to each other while also being dissimilar to data from other groups [19]. It is essential to mention that choosing a similarity measure can affect the result of clustering. We present these measures in Section 2.3.

Clustering falls into the category of unsupervised machine learning. This type of machine learning does not use previously labeled data to assess new data. Instead, the goal is to group unlabeled data into meaningful categories, making clustering algorithms suitable for finding previously hidden patterns in data.

2.2.1 Types of clustering

Many clustering algorithms emerged to resolve specific tasks, each with its own approach to clustering. The most common types of clustering include:

- Hierarchical clustering
- Partitional clustering
- Fuzzy clustering
- Density-based clustering
- Distribution-based clustering

Hierarchical clustering creates a structure consisting of multiple levels of clusters. Usually, the two most similar clusters at one level merge into one at a higher level. This process ends when there is a desired number of clusters or when all clusters are merged into one. Methods that start at the bottom and merge clusters upwards are called agglomerative. Divisive methods split clusters top-down. Results of hierarchical clustering are displayed in a tree structure called a *dendrogram*. Vast datasets may cause unreasonably long execution times, which requires hierarchical clustering to be optimized. Popular hierarchical algorithms include SLINK, CLINK, and CURE [4].

Partitional clustering divides data into clusters only on one level. There is no structure or hierarchy between individual clusters. Partitional algorithms require the number of clusters k specified at the beginning of clustering. Partitional algorithms are sensitive to noise and outliers, as they may alter cluster shapes and make the results less reliable. Arbitrarily shaped clusters or clusters that overlap are also hard to detect as partitional algorithms usually use Euclidean distance as a similarity measure. *k-Means*, one of the most popular clustering algorithms, is an excellent example of partitional clustering [18].

Sometimes it is hard to determine which cluster the data point belongs to, for example, when one data point has the same distance to two centroids (the central point of a cluster, calculated as a mean of all points belonging to the same cluster). *Fuzzy* clustering methods solve this problem by assigning each data point the probability of belonging

to a cluster. Being closer to the centroid results in a higher chance of belonging to it. The sum of probabilities for every cluster is equal to one. A typical fuzzy algorithm is *fuzzy C-means* [9].

Density-based clustering regards a cluster as an area with a high density of data points. Low-density areas separate individual clusters. Thanks to this approach, density-based methods are great at recognizing arbitrarily shaped clusters and filtering noise or outliers. This clustering is not complete, meaning that not all data points may end up belonging to a cluster. If the dimensionality of the data is too high, density-based clustering is prone to the *curse of dimensionality*. With the increasing dimensionality of the data, the distances to the nearby points become close to the distance to the farthest data point [5]. The high number of dimensions also causes the efficiency of distance measures to drop rapidly [2]. DBSCAN and OPTICS are common examples of density-based clustering.

Distribution-based clustering is the type of clustering where the cluster formation is based on different probability distributions of the data. Data points belong to the same cluster when they belong to the same distribution. The distribution-based methods are most suitable for artificial data as most of the generated points will follow some distribution. A popular distribution-based method is the Gaussian mixture model.

2.2.2 *k*-Means

In this part, we present the *k*-Means clustering algorithm. Since it is popular and easy to understand, it provides a simple introduction to clustering algorithms. Also, it helps to understand density-based algorithms better, which we will describe and use later. It also has several drawbacks that highlight the advantages of DBSCAN and OPTICS algorithms.

k-Means algorithm works by dividing the dataset D into k clusters. Every point $d \in D$ belongs to exactly one cluster. First, it chooses k points randomly as the centroids for new clusters. Every other point is then assigned to the nearest cluster, usually based on Euclidean distance, although other similarity measures may be used. After the assignment, new centroids are computed based on newly assigned points. These steps repeat until point assignment is stable or another

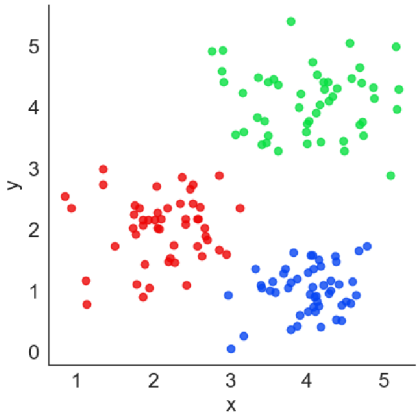

Figure 2.1: An example result of k-Means clustering.

stopping criterion is reached. An example of the result of the k-Means algorithm is visualized in Figure 2.1.

While being simple and easy to implement, k-Means has several drawbacks. Choosing the right value of k greatly affects the results, although several methods exist to estimate the correct k [17]. Using a Euclidean distance metric causes k-Means to find more spherical clusters and ignore more arbitrary shapes. One of the biggest challenges is the initialization of the original centroids. If one cluster is significantly larger than others, more centroids are probably selected from this cluster, leading to incorrect division. Many extensions or variants of k-Means exist to deal with these drawbacks, such as ISODATA, bisecting k-Means, and fuzzy c-means [27].

2.2.3 DBSCAN

DBSCAN is a widely used density-based algorithm. We describe it because it is related to OPTICS, which we will use. Understanding DBSCAN first will make understanding OPTICS much easier.

The main idea of DBSCAN is that to be a part of a cluster, each point must have a certain number of points within its radius. The algorithm takes two parameters, $MinPts$ for the number of points required and ϵ for the size of its neighborhood.

To decide which points are in an area dense enough to be considered a cluster, the algorithm defines the density-reachability relation. To define it, we first need to define direct density-reachability [12].

Definition 1. *If a point p has at least MinPts points in its ϵ-neighborhood, then every point q in its ϵ-neighborhood is considered directly density-reachable from p.*

However, this relation is not enough to create a complete cluster as it only considers ϵ-neigbhourhood points. The definition of density-reachability relation broadens its scope to include every reachable point.

Definition 2. *If there is a point q directly density-reachable from p and r is directly density-reachable from q, then r is density-reachable from p.*

The algorithm selects a random point and checks whether there are any density-reachable points. If there are, the point is considered a core point, and a new cluster is created. If not, then the point is considered a border point, and the algorithm continues. Some border points may be added to a cluster later if they are density-reachable from another point. This process usually produces border points that are not a part of any cluster. They are considered noise [12]. Figure 2.2 shows types of points DBSCAN uses. Clusters formed by DBSCAN thus consist of core points, points directly density-reachable with the core points, and their respective border points.

Thanks to its density-based nature, DBSCAN has many advantages. There is no need to specify the number of clusters, and there are no initialization problems caused by unequal point distribution in clusters. It is resistant to outliers and correctly determines arbitrarily shaped clusters.

One of the main disadvantages is that DBSCAN performs poorly on datasets with clusters that have different densities. Parts of the dataset with a lower density may not be detected if ϵ and *MinPts* are set too high. On the other hand, if the value of parameters is too low, all points will be recognized as one cluster. OPTICS algorithm, an extension of DBSCAN, deals with these problems better by ordering the points in a reachability plot.

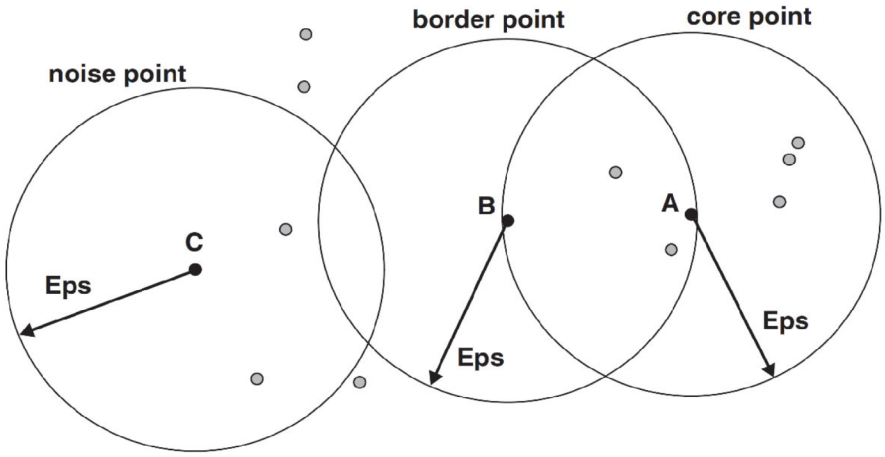

Figure 2.2: Different point types based on number of other points in their ϵ-neighborhood [31, p. 233].

2.2.4 OPTICS

OPTICS algorithm is different from other means of clustering, since it orders the points in the dataset so that the ordering reflects the structure of the data. The results obtained from this ordering correspond to the results obtained by DBSCAN clustering algorithm [3].

Similar to DBSCAN, OPTICS takes ϵ and *MinPts* as parameters and recognizes the density-reachability relation. To obtain the correct ordering, however, these properties are not enough. The algorithm orders the points based on their distance from one another and defines two types of distance [3].

Definition 3. ***Core distance*** *is the distance between point p and its MinPts' closest point q in its ϵ-neighborhood.*

Definition 4. ***Reachability distance*** *is the distance between point p and point q in its ϵ-neighborhood. If the distance between p and q is smaller than core distance, then reachability distance is the same as core distance.*

The algorithm goes through the points in the dataset until it finds the first core point (the point that has at least *MinPts* points in its ϵ-neighborhood). When such a point is found, the reachability distance of directly-density reachable points is calculated, and the points are inserted into the priority queue. In the front of the queue are the

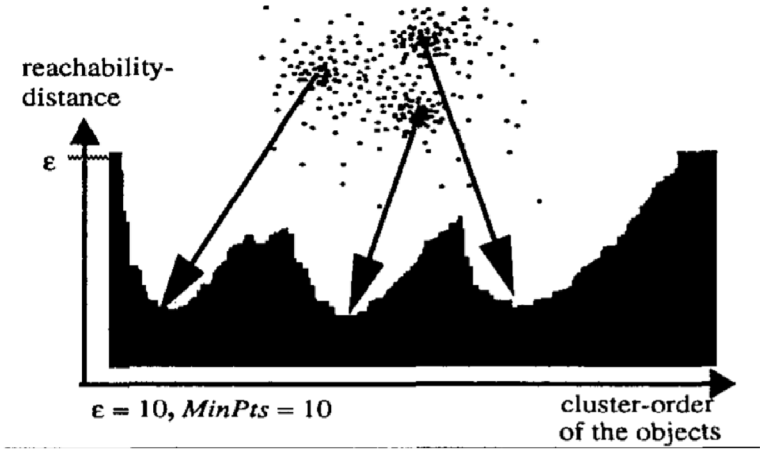

Figure 2.3: Reachability plot orders objects on the x-axis based on their similarity. Values on the y-axis represent the distance of an item from a previous one. Several similar data points form a valley, representing a cluster, while spikes in the ordering represent outliers [3].

points with the lowest reachability distance from the current point. New reachability distance is computed for the points already in the queue if they are reachable from the currently processed point. After being processed, the current point is added to the reachability plot. It is then removed from the queue, and the point with the lowest distance is processed next.

Reachability plot is the result of the OPTICS algorithm. It contains all points from the dataset in the order in which they were processed, with their reachability distance. Figure 2.3 shows an example of reachability plot. The data structure is clearly visible in this plot, with lower areas corresponding to clusters and spikes representing noise points. The plot also shows what clusters would different values of ϵ produce. Instead of setting a single value for ϵ, OPTICS considers a range of possible values. Still, a maximum value for ϵ needs to be set. Setting ϵ, however, is not as a crucial decision as is the case when using the DBSCAN algorithm. Additionally, clusters with very different densities are also recognized. On the whole, the reachability plot offers excellent insight into the data, its structure, and distribution.

2.3 Measuring similarity

As clustering is the process of grouping similar items, it is necessary to determine how much alike two items are. Similarity measures are used to compute a numeric value that represents the similarity between two items. Unfortunately, no single universal similarity measure exists. Moreover, research suggests that choosing a suitable similarity measure can affect the results of clustering more than the choice of clustering algorithm itself [25]. The most popular distance measures are Euclidean distance, cosine similarity, and the Pearson correlation coefficient [22].

3 Related work

In this chapter, we present related works. Section 3.1 presents works that survey the current state of EDM and clustering. The applications of clustering algorithms are presented in Section 3.2. In Section 3.3 we present papers that are focused on similarity measures.

3.1 Educational data mining

The Handbook of Educational Data Mining by Romero et al. [27] is an extensive overview of the current state of the art of EDM. The book explains basic DM techniques and includes several scientific papers that provide a more in-depth look at the introduced methods. Chapter 6 explains the motivation behind using clustering in DM and describes several types of clustering. It also provides a brief overview of literature where clustering was applied to solve educational problems.

Romero et al. [26] also survey relevant papers in the field of EDM. The authors divided research papers into categories based on their goal and the utilized DM method. Clustering is used mainly in studies with the goal of improving feedback to the instructors, detecting undesirable or otherwise unusual student behavior, and grouping students by various characteristics.

Another survey by Dutt et al. [10] reviewed clustering in the context of EDM. Clustering has been used to analyze and model student behavior, group students according to their learning style, discover ineffective student behavior, and provide better feedback to teachers. The survey offers an overview of academic works, their objective, the clustering algorithm they used, and a brief description of the dataset. The authors also pointed out unexplored areas for further research and noted the advantages of clustering in educational research.

3.2 Applied clustering methods

3.2.1 Hierarchical clustering

Jovanović et al. [16] used sequence analysis and agglomerative hierarchical clustering to identify students' learning behavior and differ-

ent learning approaches. The study used data containing students' interactions with a learning management system. Researchers used exploratory sequence analysis to uncover different interaction sequences. Clustering them showed common patterns in learning processes. Clustering was also applied to find student groups according to the sequence patterns they used. The authors then described different student groups and evaluated the performance of the learning strategies they employed. Results could help encourage the students to adopt more effective learning strategies and improve their performance.

3.2.2 Partitional clustering

Yee-King et al. [33] used *k*-Means++, a variation of the popular *k*-Means algorithm, to discover the way students develop programs. Students attended an online course, where they received a sample code and were free to experiment with it. Code changes were represented by a tree structure, where children nodes represent copies students made and further developed. Researchers used metrics such as the number of code edits, rate of code editing, and new vocabulary added by students to describe their coding behavior. The study found clustering to be a helpful tool in getting insight into previously unknown behavior and whether the students work with the code as expected by instructors.

Quille et al. [23] examined how promoting a growth mindset (a belief that talents and skill can be improved) affects students' performance. First, they explored whether mindset intervention improved performance. After comparing the test group with the group from the previous semester, results revealed a statistically significant improvement when the intervention was employed. However, the high-level analysis did not provide enough detail on how different groups of students are affected. By utilizing *k*-Means clustering, the authors found that the mindset intervention improved the performance and self-efficacy of new and less experienced students. At the same time, it lowered the performance of more experienced senior students.

McBroom et al. [20] mined submission logs from an auto-grading system for program code. They clustered weekly submissions to find how students approach each assignment while also analyzing the behavior over the whole semester to learn how students develop. Thanks

to this two-step approach, researchers detected common behavioral patterns. These patterns were identifiable not only at the end of a semester but as early as in week three, and students' behavior largely remained the same whole semester. Authors suggest that teachers use the gained insight to intervene when students belong to the group with a higher risk of failure.

3.2.3 Density-based clustering

Yin et al. [34] used OPTICS to cluster student programming assignments. Their goal was to enable instructors to grade students automatically based on the type of their solution. Student source code was represented in the form of an abstract syntax tree, with the normalized tree edit distance as the similarity measure. The instructors could use the insight gained from this approach to show the model solution and explain the resulting benefits and drawbacks.

3.2.4 Distribution-based clustering

Bouchet et al. [7] utilized the Expectation-Maximization algorithm to find different learning styles. They used log files containing data about users' interactions with an online tutoring system. The study found differences between different styles of learners in performance, time spent in the tutoring environment, and student behavior in it. The system also used prompts to guide and help the students. As these prompts were based on students' interactions, they also differed between clusters. However, these differences did not favor one style over the other. The authors further suggest using the extracted clusters for the online classification of different student types.

Emerson et al. [11] used Bayesian Gaussian Mixture Models to explore novice coding misconceptions in block-based programming. Students engaged in two programming activities in a block-based programming environment. As their goal was finding misconceptions, the researchers used system logs of unsuccessful attempts as input data. To represent students' programs, they used three families of features: basic block features, counts of specific block sequences, and the number of interactions with the system. Results revealed three distinct groups of students: exploratory, disorganized, and a near-miss

cluster. Authors conclude that by using logged features, clustering can reveal student groups with similar difficulties.

3.3 Evaluating similarity measures

Huang [14] compared five similarity measures when clustering text documents. To examine their effectiveness, she applied the *k*-Means clustering on seven datasets containing news articles, web pages, and academic papers. The author measured the quality of discovered clusters by examining how many items in a single cluster belong to the same category. The analysis of the results revealed that the performance of similarity measures was very close, except for the Euclidean distance, which performed significantly worse than the rest.

A similar study by Strehl et al. [30] examined the impact of four similarity measures on clustering high-dimensional sparse data from web-pages. The researchers compared the effect of different similarity measures in combination with several clustering algorithms. To evaluate their performance, they compared labels generated by clustering with labels assigned to pages manually. Results revealed that Euclidean distance was not a suitable measure for high-dimensional data, while cosine similarity, Jaccard similarity, and Pearson correlation performed comparably.

Pelánek et al. [22] discussed different approaches to measuring the similarity of items in introductory programming. When measuring item similarity, there are several types of usable input data: item statement (description of a task), item solution, and performance data. These data can be transformed into a feature matrix, which can then be used to compute the similarity matrix. After transformation, data are ready for visualization or clustering. The researchers compared combinations of these different approaches and evaluated how a choice of similarity measure affects the results. After a thorough analysis, their proposed approach was to use item solutions as input data and use the similarity matrix computed from a normalized feature matrix. Authors note that the item solutions are a suitable choice for input data as they are readily available and easy to process. However, the exact approach and similarity measure depend on the particular application.

4 Research methods

In this chapter, we describe the research methods used in our thesis. Section 4.1 explains the origin of the data. In Section 4.2, we present the structure of the data. Section 4.3 explains data preprocessing. In Section 4.4, we discuss the selection and application of the OPTICS clustering algorithm. Finally in Section 4.5 we describe how we examined the resulting clusters to draw conclusions.

4.1 Cybersecurity training

The data used in this thesis were collected from cybersecurity training running in the KYPO cyber range platform [32]. KYPO is a cloud platform developed at Masaryk University, which offers a scalable virtual environment suitable for cybersecurity training. KYPO also stores commands submitted in the training sandbox, which can be analyzed later.

In the training we analyzed, the participant is in the role of the attacker, intending to exploit vulnerabilities in the network of virtual machines. The training is divided into several phases called levels. At each level, the participant is given a task to complete. He achieves this by typing the correct commands in a terminal and finding a string called "flag". After obtaining the flag, participants submit it to progress to the next level.

During training, participants use several command-line tools. They are free to use the internet to search for help with using these tools. If they are unable to complete the task on their own, they can take hints or view the entire solution. The participants are usually computer science students from the Faculty of Informatics at Masaryk University, although high school students also participated in some training.

In the examined training sessions, the trainees' first task is to scan the target server using `nmap`. Next, they need to identify a vulnerable service and exploit it using Metasploit tools to gain access. Then, they have to copy a private SSH key, crack its passphrase using `john`, and use it to access another remote host that contains secret files.

4.2 Data format

The commands typed in the terminal are stored in logs, which capture the progress of each participant through the training. Logs are stored in a JSON format; one file represents the training session of one participant. The dataset we used contains command logs from 73 trainees in 12 cybersecurity training sessions, with information describing 5,747 commands. Participants could execute two types of commands: Linux terminal commands and commands from the Metasploit tool [24], which is a framework that enables users to scan remote computers for vulnerabilities and to execute exploits. Log representing an example of a captured command from our dataset is shown in Figure 4.1.


```
{
      "training_name" : "charlie2020may",
      "cmd_type"      : "bash-command",
      "man_ip"        : "192.168.67.192",
      "pool_id"       : 1,
      "sandbox_id"    : 340,
      "timestamp_ms"  : 1589444433,
      "timestamp_str" : "2020-05-14T10:20:33.933808+02:00",
      "hostname"      : "attacker",
      "cmd"           : "nmap -v -A 10.1.26.9",
      "uname"         : "root",
      "uid"           : 0,
      "wd"            : "/root",
      "fromhost_ip"   : "10.1.135.83",
      "programname"   : "command"
}
```


Figure 4.1: A log record containing details of an executed command.

As the logs contain a lot of attributes that are not useful for our purposes, we will describe only the fields relevant to our work. The `cmd_type` field shows the type of the executed command. Its value can be either `bash-command` for Linux terminal commands or `msf-command` for Metasploit commands. The `timestamp_ms` attribute is a time at which the user executed the command, in the Unix time. The most important feature, the `cmd` field, contains a string representing the executed command and tells us what tool and arguments the trainee

used. All of the collected data are anonymous and do not contain any personal information.

4.3 Data preprocessing

Data saved in logs are not directly suitable for clustering. To apply a clustering algorithm, we first need to automatically convert the data into a more usable format. We use this internal representation as a basis for creating feature matrices. We used Google Colaboratory to host the Jupyter Notebooks we wrote. It is a suitable environment for our research as it allows results to be easily reproduced [21].

4.3.1 Internal representation

To accurately represent both the training session and the commands, we created two custom classes in Python 3. The `Command` class represents one logged command. It contains information about the type of executed command, the tool and the options used, time at which the command was executed, and whether the typed command contained options used to get information about the tool, such as `--help`. The format of the Command class is presented in Figure 4.2

```
Command:
cmd_type = "command"
tool = "nmap"
options = "-v -A 10.1.26.9"
time = 1589444433
help = False
```

Figure 4.2: An internal representation of the command log presented in Figure 4.1.

The `Training` class represents a training of one trainee. It contains a list of `Command` objects with all the commands a trainee typed, the count of the terminal and Metasploit commands throughout the whole training, the average time difference between two executed commands, the number of times trainee consecutively tried the same tool with different options, the count of help commands, and the name of the

file so that every training can be identified. We show an example of a Training class object in Figure 4.3.

```
Training:
command = [Command_1, Command_2, Command_3, ...]
bash_count = 35
msf_count = 13
average_delay = 126
opt_changes = 9
help_count = 1
file_name = "sandbox-396-useractions.json"
```

Figure 4.3: An internal representation of a training session.

4.3.2 Data irregularities

To convert the log history files into the internal representation, we wrote a parser that transforms all relevant files from a given directory. All invalid trailing characters before the beginning of the log record are disregarded.

There was an issue of long delays between two typed commands, as some training sessions were paused and continued several hours later or the time zone setting on the computer changed, resulting in a negative time difference. We chose to ignore gaps lasting longer than one hour, as this value proved best at eliminating extremely long delays while not eliminating longer but still sensible ones.

Another problem we encountered involved an RSA key (a long string) submitted as a sequence of terminal commands. We chose to ignore all lines where the `tool` attribute was longer than 35 characters, to eliminate these instances while preserving the valid commands. We determined the `tool` length so that it does not remove any valid `tool`.

4.3.3 Creating feature matrices

The next step in the data processing was creating the feature matrix, which would be used as an input for a clustering algorithm. We constructed one feature matrix based on the *bag of words* model and one from the features we selected to represent a training session.

4.3.4 Bag of words matrix

One of the standard techniques when obtaining features from text is to use the *bag of words* model [22]. In this model, each text document is represented by a set of words it contains and their count. Documents are converted into vectors of the same length. Every word in a dataset is included in the vector. If a document does not contain a particular word, the count is set to zero.

We can use this approach to represent training sessions with a set of commands the trainee used. Each `tool` is equivalent to a word in a document. We can create vectors containing information about the tools a trainee used, how often, and we can compare them to other trainees' vectors.

```
{"cmd" :"nmap --help"}
{"cmd" :"nmap -v 172.18.1.5"}
{"cmd" :"msfconsole"}
{"cmd" :"search webmin"}
{"cmd" :"use exploit/unix/webapp/webmin_backdoor"}

Count dictionary: {"nmap: 2", "msfconsole: 1", "search: 1", "use: 1"}
```

Figure 4.4: A dictionary containing tool occurrences from the simplified command logs.

We implemented methods to count occurrences of different tools used in every training. Example of a dictionary containing `tool` count is shown in Figure 4.4. Then we employed a vectorizer to build a bag of words feature matrix. We only use the `tool` part of the command for feature generation as using it together with the `options` would result in too many unique features (1,658 features using whole commands compared to 314 using only the `tool` part). A sparse matrix with a lot of features could lead to worse performance of the clustering algorithm due to the curse of dimensionality, which we mention in Chapter 2. However, we do not entirely disregard the `options` used, as they form an essential part of the correct training solution. We visualize the relationship between the `tools` and `options` used in Chapter 5.

We also tried applying the TF-IDF transformation, which is a statistical technique that weighs the importance of a word in a document

based on its frequency, since it is also recommended as a typical normalization technique in text mining [22]. We found it unsuitable for our setting since TF-IDF transformation suppresses the weight of the most common terms, which would overlook often used tools (such as `nmap` or `ssh`) that form a substantial part of trainee's progress.

4.3.5 Matrix of selected features

While the *bag of words* model captures what commands trainees have used, it does not consider other information available in the logs, such as the time of execution, the delay between commands, and the type of the command. We have selected five custom features to capture other insights into how trainees progressed through the training. We explain these features and the rationale behind their choice below.

- *bash-count*: The number of submitted Bash commands. A small number may suggest trainee did not progress far in training. The high number may indicate using a trial and error approach.
- *msf-count*: The number of Metasploit commands trainee used. The Metasploit environment may be new for some trainees, and the high number of executed commands may indicate they had difficulties in progressing through this part of the training.
- *avg-delay*: The average delay between two commands. Large gaps between commands may suggest the trainee did not understand how to use a tool and looked for the information online. Small delays may indicate brute-force guessing.
- *opt-changes*: The number of times trainee used the same tool twice in a row but changed the options. A high count may show the trainee's unfamiliarity with the tool or his inability to use it with the correct options.
- *help-count*: The number of times trainee displayed help information or manual page for any tool. It may also indicate the trainees' unfamiliarity with it.

As the high correlation between features may make them redundant, we calculated the correlation coefficient between them. While there

was a slight correlation between *opt-changes* and *msf-count* with coefficient value around 0.61, all other features were correlated less (absolute values ranged from 0.1 to 0.51).

We calculated and stored feature values for each training while processing the corresponding log history file. Then, we created a feature matrix using a vectorizer, similarly to the bag of words feature matrix.

4.4 Clustering analysis

To cluster data, most clustering algorithms require a similarity measure for calculating the distance between points in the dataset. We selected the cosine similarity as the most suitable measure since it performs well on high-dimensional data and is often used to compute text similarity [29].

We chose the OPTICS algorithm to perform cluster analysis on our data. OPTICS offers several advantages compared to other clustering algorithms described in Chapter 2. OPTICS computes cluster-ordering, which can be easily visualized in a reachability plot. Thanks to the properties of reachability plot, OPTICS can visualize even high-dimensional data and is resistant to outliers [3].

Furthermore, the hierarchical ordering provided by OPTICS contains the same information as dendrograms formed by hierarchical clustering [28], which is a suitable method for clustering small datasets such as ours [1].

For the setup, OPTICS requires only one parameter *MinPts* to set the minimum number of points required for cluster formation. Research suggests setting the number of *MinPts* to $ln(n)$, where n is the number of points in the dataset [6]. For our dataset, the recommended value should be close to $ln(75) \approx 4$. However, we experimentally chose to set this number to 5 for both feature matrices as it resulted in more clusters and fewer points designated as outliers. We also experimented with both lower and higher values up to 15, but they proved unsuitable.

4.5 Cluster examination

We first examined the formed clusters by visualizing the most common tools and their options to see which commands trainees utilized the most. The tools trainees used would help us determine the most problematic parts of the training and show how far the trainees progressed. To determine whether the trainees in a cluster were successful, we examined the `cat` tool frequency needed to get the final flag.

Then we examined the relationship between the delays and frequent tools to see which tools were associated with longer delays, as tools with longer delays may have been more challenging for the trainees.

To see if the cluster formation was impacted by the specific training session, we looked for clusters dominated by data points from the same file location, implying a particular training impacted these clusters.

We also considered the selected features to get additional information about the clusters, compared them and noted the differences when they were notable.

5 Results and discussion

In this chapter, we present and interpret the results and visualize the discovered clusters. In Section 5.1, we show the results obtained from clustering of the bag of words matrix and discuss the differences between individual clusters. In Section 5.2 we analogously present the results from the matrix of selected features.

5.1 Bag of words matrix results

We applied the OPTICS algorithm on the bag of words matrix, with the *MinPts* set to 5 and the cosine distance as the similarity measure. The resulting reachability plot is visualized in Figure 5.1. Clustering revealed three distinct clusters discussed below. However, the results also show a large number of outliers discussed in Section 5.1.4.

5.1.1 Cluster 1

Cluster 1 (red) contains 15 data points. This cluster is the most compact out of the three formed, with low reachability distance between points (visualized in Figure 5.1), implying high degree of similarity between the trainees during their progress through the training.

The most utilized commands for Cluster 1 are shown in Figure 5.2. The most distinguishing feature of this cluster is the use of `cd` and `ls` tools. These tools are used to change directories and list their content, which trainees often did in the later stages of the training to access necessary files. The high usage of these tools may suggest they had trouble locating these files. Similarly to other clusters, `ssh`, `nmap` and `set` tools were among the most frequent. The `nmap` proved to be the most challenging of the three as it is associated with long delays and multiple attempts to figure out the correct options. While the `set` tool was used often, it is probably because it requires multiple inputs to set all parameters. The `set` frequency is also lower in comparison with other clusters. A relatively low number of Metasploit commands and the lack of option variety suggest that Cluster 1 trainees did not struggle much with Metasploit environment.

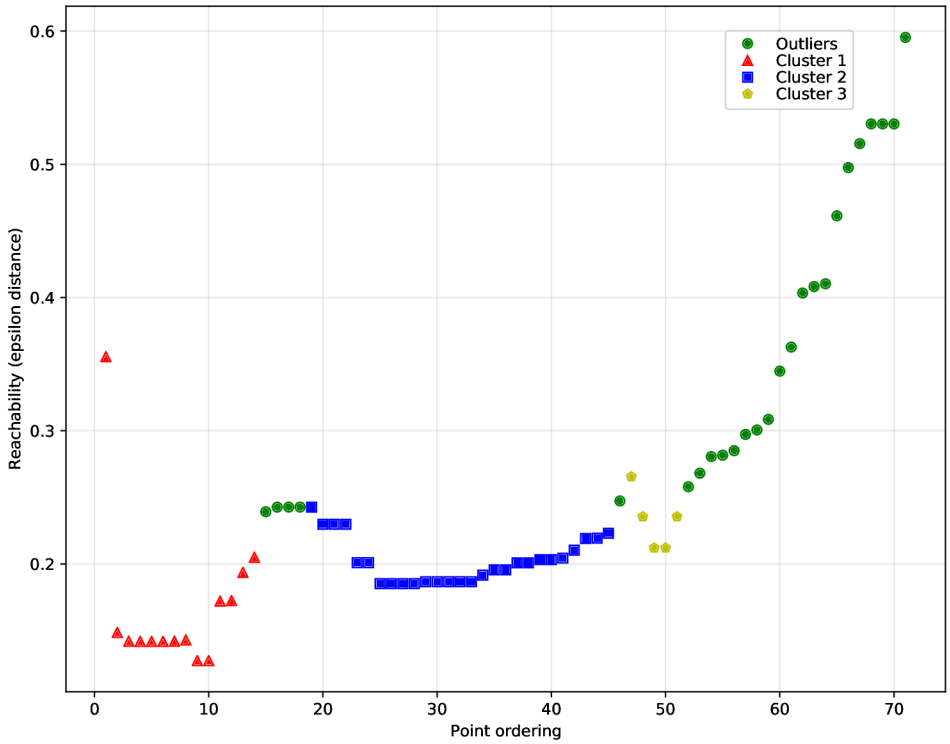

Figure 5.1: Reachabilty plot showing the results of clustering. Each cluster is marked by a different color: red for Cluster 1, blue for Cluster 2, and yellow for Cluster 3. The outliers are colored green. The higher the *Reachability* value is, the more the data point differs from the previous one. The ordering is decided by the OPTICS algorithm.

We noticed the trainees in Cluster 1 differ from trainees in Cluster 2 in a number of used Bash commands as they use a significantly higher number of them than the Cluster 2 trainees.

To confirm this observation, we compared two lists containing the Bash command count for each cluster. We used the T-test for the null hypothesis that Cluster 1 and Cluster 2 have the same average number of Bash commands. We used the implementation from the SciPy Python library [8]. The resulting p-value (0.034) was smaller than 0.05, allowing us to reject the null hypothesis.

The most apparent reason for the difference between clusters is the widespread use of `ls` and `cd` commands by the Cluster 1 trainees.

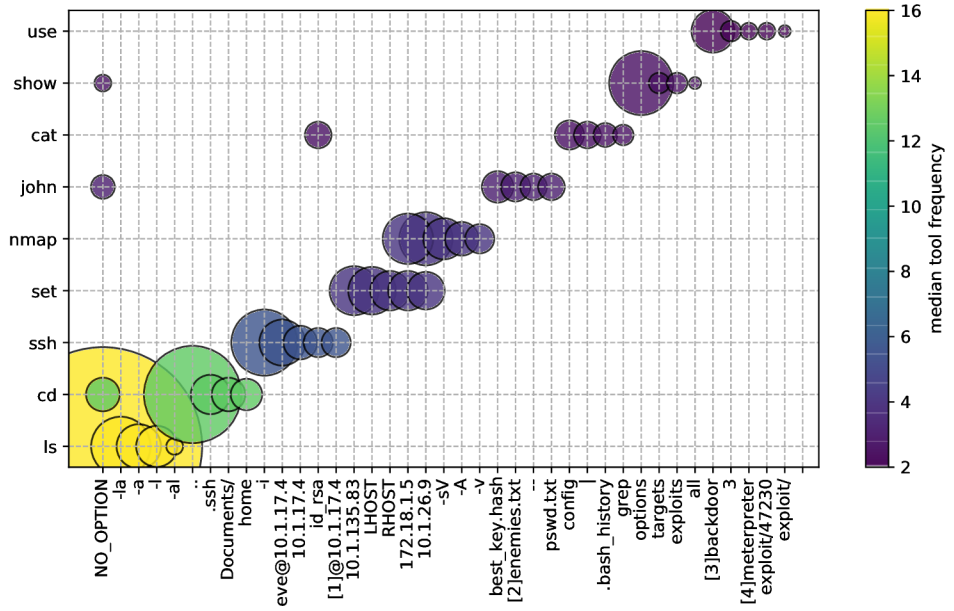

Figure 5.2: Bubble plot showing the most common `tool` and `option` combination for Cluster 1. The size of the bubble is correlated with the option frequency. The color represents median tool frequency for the cluster.

Another reason for this may be the changing of options of `nmap` and `ssh` tools mentioned before. The second difference lies in the higher number of times these trainees used `--help` option, however the differences were not high enough to be significant.

While the trainees used a variety of options for each tool, Figure 5.2 shows that trainees did not struggle much with any particular tool. Since the number of commands with `cat` tool combined with the name of the file containing the last flag revealed that half of the students completed the training, we can conclude that Cluster 1 trainees performed well and generally used the right tools with the correct option combination.

5.1.2 Cluster 2

Cluster 2 (blue) forms the largest cluster with 27 data points. Trainees in this cluster used fewer Bash commands but more Metasploit com-

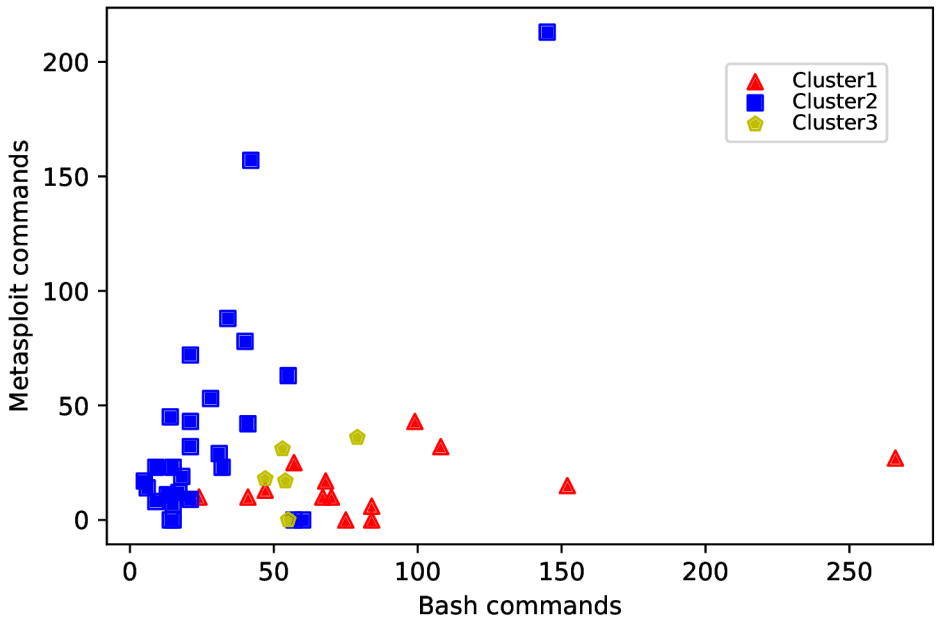

Figure 5.3: Scatter plot visualizing the Bash/Metasploit command relationship for each cluster.

mands. We visualize the Bash/Metasploit ratio for the discovered clusters in Figure 5.3.

Figure 5.3 suggests a subgroup within this cluster formed by trainees who utilized a minimal number of both types of commands. As these trainees did not progress far into the training, the lack of motivation or skill is the most straightforward explanation.

A higher number of commands executed in Metasploit is also reflected in Figure 5.4 as the Metasploit tools are the most frequently used. The `set` tool was the most common. Cluster 2 trainees used similar options for this tool as trainees in other clusters. The main difference were low delays and high numbers of options, suggesting they struggled to figure out the correct option combination. The frequent use of `run` and `show options` commands further proves that trainees were unsure whether they used the `set` tool correctly. Trainees also often returned to Metasploit environment by the use of the `msfconsole` tool, possibly to alter their previous settings. Surprisingly, while the trainees struggled with Metasploit tools, they had no major issues

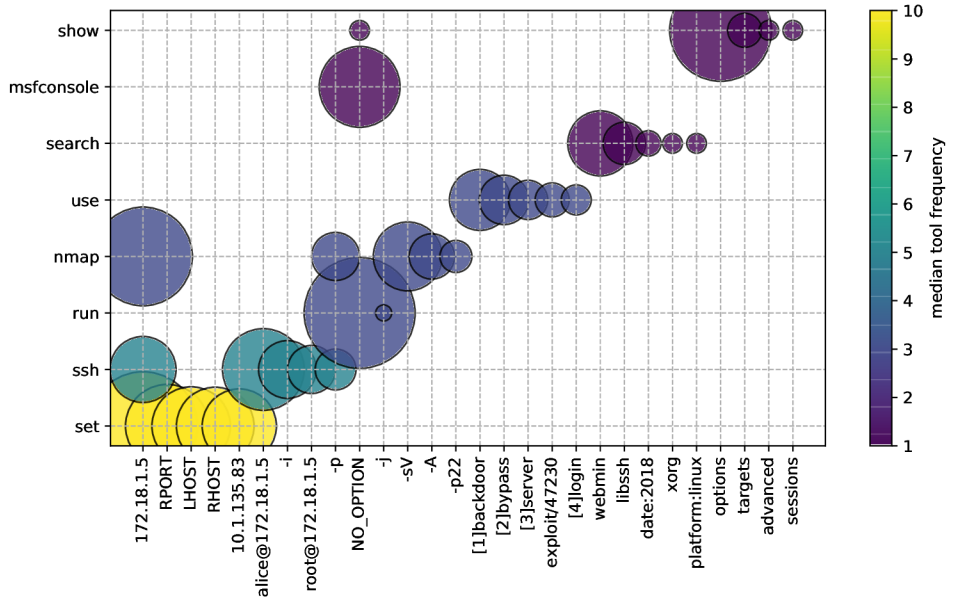

Figure 5.4: Bubble plot showing the most common `tool` and `option` combination for Cluster 2.

using `nmap` and `ssh` tools. In both instances, they mostly utilized the same options.

The overall performance of Cluster 2 trainees was low, as they did not reach the later parts of the training, experienced difficulties while using Metasploit tools, or ended the training very early.

This generalization may not apply to all trainees, as a large part of Cluster 2 is formed by the trainees from *Alpha2018 – House of Cards* training. The last task in these training instances was to use the `ssh` tool. As the most of Cluster 2 were not able to progress beyond this task, their similar command composition caused unsuccessful trainees to be assigned to the same cluster as successful trainees from *Alpha2018 – House of Cards* training.

5.1.3 Cluster 3

Cluster 3 (yellow) is the smallest of the three, containing only 5 points, the minimum required for cluster to form. The Figure 5.3 shows that trainees in Cluster 3 had a similar command ratio to Cluster 1. They

were also relatively successful since three trainees completed their training, while the two others reached later stages.

The `set` tool was again the most frequent. In contrast to the previous cluster, Cluster 3 trainees did not experience such prominent issues with Metasploit environment and progressed further. The `john` tool seems to have caused the most trouble for the trainees, as they used it surprisingly often and with longer delays than the rest of trainees. Another frequent tool was `ls`, which trainees used more frequently in later stages. It is not, however, accompanied by the change of directories as in Cluster 1. Lack of `cd` tool may suggest the trainees had a better idea about the location of the last files.

5.1.4 Notable outliers

Since the number of outliers (26) is almost as large as the size of the largest cluster (27), we present the some of them to offer additional insight. While these points do not belong to any particular cluster, they still provide valuable information.

Delta2020-sandbox-329-useractions.json is an outlier located between the first and second cluster. This trainee differed from these clusters as he experimented with the `find` tool at the beginning of the training. The other difference was that trainee did not use `cd` and `ls` tools to access the final file. Instead, he used a full file path to access it and completed the training without using these tools. It is notable that even small differences in overall training progression such as these were enough for the trainee to be considered an outlier.

In *Delta2019-sandbox-181-useractions.json* log file, the trainee did not employ the widely used `nmap` tool. He chose to use the `nikto` tool to scan the network. He completed a large part of the training but did not finish it. This outlier shows that instructors can identify unique approaches from the outliers clustering produces.

The *Echo2019-sandbox-181-useractions.json* outlier is located close to Cluster 3. The trainee completed the training, and similarly to Cluster 3 trainees struggled with the `john` tool. However, he used it 59 times until he finished the game, which poses a significant difference from the median number of 6 in Cluster 3.

The *Echo2019-sandbox-85-useractions.json* and *Echo2019-sandbox-86-useractions.json* trainees both tried to exploit a known vulnerability

with remote execution of Python code. When this approach failed, the latter trainee switched to `nmap` tool and continued the training, while the former tried to execute the code with different parameters several times until he grew frustrated and ended the training.

5.2 Selected features results

We applied the OPTICS algorithm on the matrix of selected features with the same parameters as on the bag of words matrix (MinPts = 5, cosine similarity measure). The reasoning behind the selected features is outlined in Section 4.3.5. Clustering algorithm produced four clusters. We visualize the reachability plot in Figure 5.5. As the feature values define clusters formed by the selected feature matrix, we visualize value differences in Figure 5.7. We do not mention outliers from this clustering as most of them were designated as cluster points in the bag of words matrix. The only significant overlap between clusters from the two matrices is mentioned in Section 5.2.4.

5.2.1 Cluster 1

Cluster 1 (red) consists of 15 points. Its dominant feature is the high frequency of options that provide information about the currently used tool. The distribution of `help_count` is shown in Figure 5.6. The number that high may be explained by the trainees' unfamiliarity with the Bash tools, especially `nmap` as it is the most used tool for this cluster. While the number of Metasploit commands is comparable to other clusters, both the lower number of Bash commands and the lack of tools commonly used in later stages of the training (`john`, `cat`, `ls`) suggest that the trainees had troubles completing the whole training. This is also reinforced by the fact that only 2 of the 15 trainees completed training. Figure 5.7 also shows that trainees did not change tool options much, though that may be due to the overall lower number of commands.

The most common tools are Metasploit tools such as `set`, `show`, `use`, and `msfconsole`. The Metasploit tool `help` shows that Cluster 1 trainees were unaccustomed to the Metasploit environment. However, some of the trainees were able to solve Metasploit tasks, as another

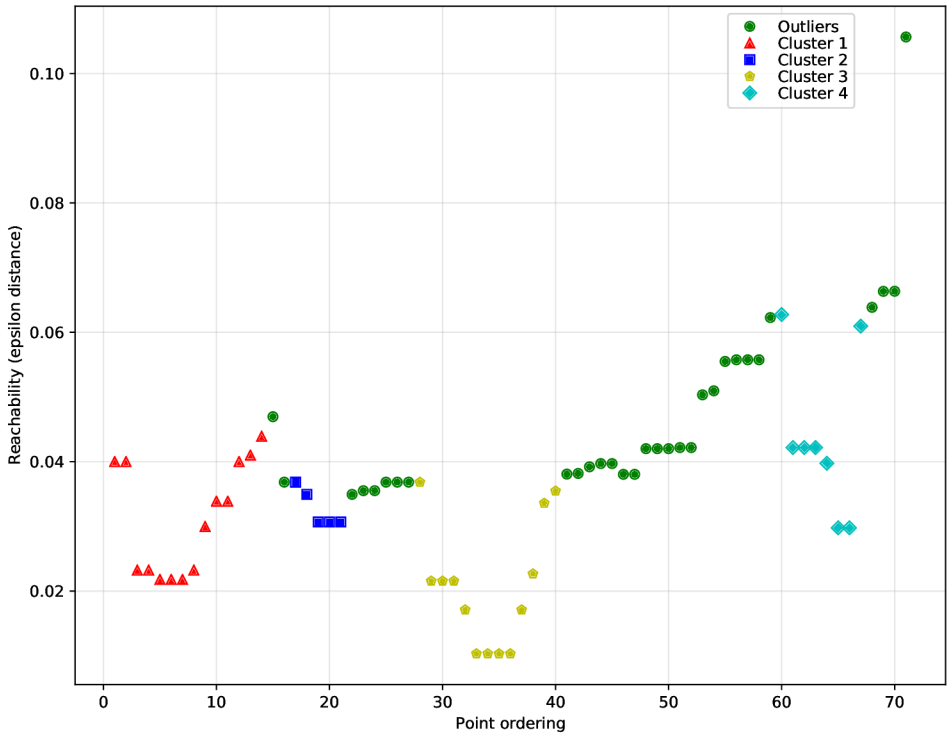

Figure 5.5: Reachabilty plot showing the results of clustering. Each cluster is marked by a different color, red for Cluster 1, blue for Cluster 2, yellow for Cluster 3, and cyan for Cluster 4. The outliers are colored green. The higher the *Reachability* value is, the more the data point differs from the previous one. The ordering is decided by the OPTICS algorithm.

frequent tool is `ssh`, which is mostly used in subsequent tasks. The `nmap` tool proved difficult for trainees as it is used frequently, often with `--help` option.

5.2.2 Cluster 2

Cluster 2 (blue) is small, containing only 5 points. Trainees in Cluster 2 had the highest median value of used Bash commands, as shown in Figure 5.7. The tools they utilized suggest trainees completed most of their tasks. Among the most frequent belong `john`, `cat`, and `ls`.

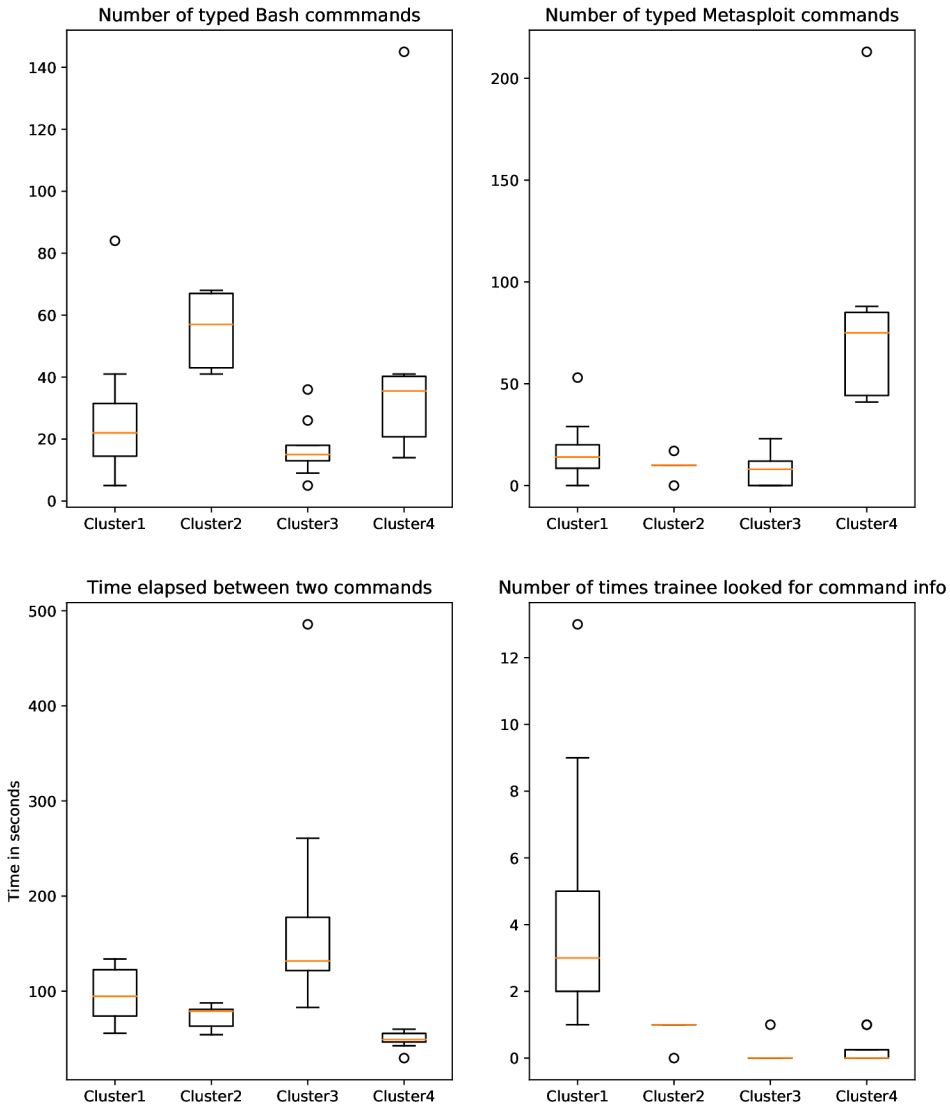

Figure 5.6: The box plots showing the distribution of feature values for each cluster. The orange line in each box reprents median value.

While the trainees used more Bash commands and changed options frequently, their success in completing training (4 out of 5) shows these high numbers are not caused by the incorrect use of Bash tools.

The number of Metasploit tools, on the other hand, was low and more similar to the clusters of trainees that did not progress far, show-

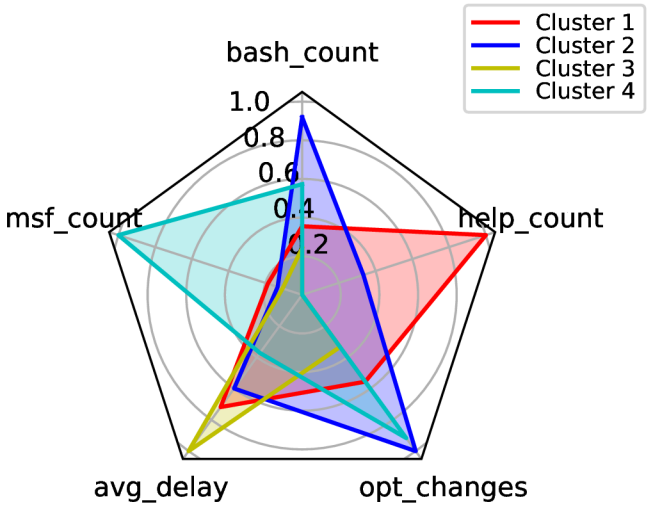

Figure 5.7: Radar chart showing the median values of clusters for each selected feature. Feature values are scaled by the feature maximum.

ing that the successful solution does not require many Metasploit commands. The most common were `set` as it requires multiple uses to set all parameters, followed by `show options` and `use exploit`. These tools were not frequently used, further proving Cluster 2 trainees experienced no significant issues with Metasploit. The `nmap` and `ssh` were very frequent, showing that even these successful trainees needed more tries to figure out their correct usage. The delays between commands were slightly lower than in the other clusters, which may suggest trainees had more experience with the tools they used.

5.2.3 Cluster 3

Cluster 3 (yellow) contains 13 points. It is distinct from the rest of the clusters in every feature. The defining feature of Cluster 3 is the low number of both Bash and Metasploit commands. Trainees ended their training very early, usually entering the Metasploit environment and then quitting. Simultaneously, many trainees did not complete the second task as they used no Metasploit tools. The low rate of the option changes also corresponds to the low number of commands overall. There are two possible explanations for trainees' failure to

complete even the early tasks: the lack of motivation and the lack of experience with command-line tools.

The `help_count` feature shows that practically no trainee used options to learn more about a tool's usage. The delay times between commands also vary with regard to other clusters. The average delays of Cluster 3 trainees are longer than that of the other trainees. Figure 5.8 also shows that trainees with long delay times tend to use fewer commands. A possible explanation for this tendency may be that inexperienced trainees spent more time searching for tool information online. A combination of inexperience with the tools, lower motivation, and inability to use the tool correctly may cause these trainees to give up early.

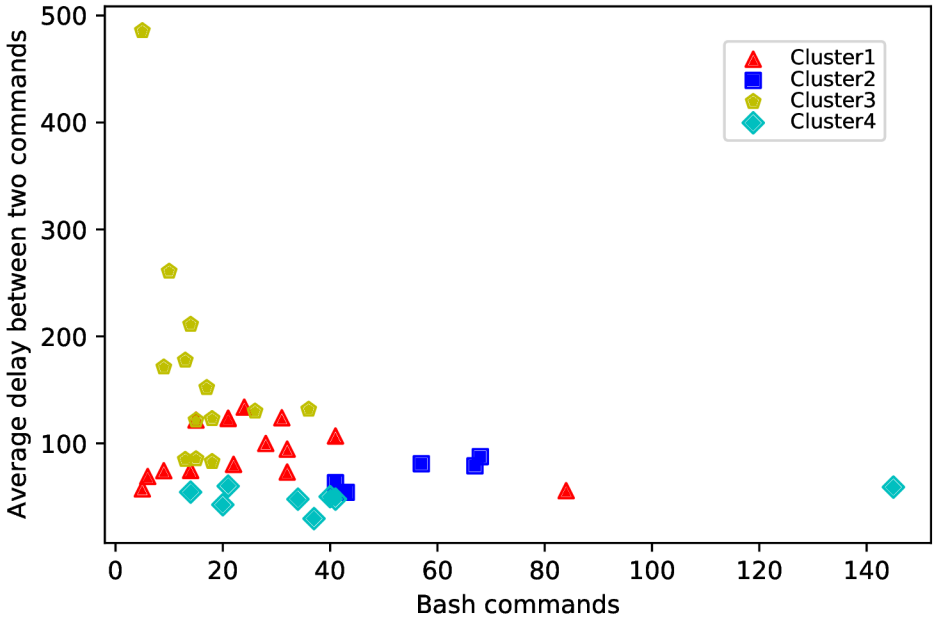

Figure 5.8: Scatter plot showing the relationship between high delays and the number of Bash commands.

5.2.4 Cluster 4

Cluster 4 (cyan) has only 8 points. In some features, it is similar to Cluster 2 as trainees in both clusters used a higher number of Bash

commands and changed options at the same rate. The trainees in Cluster 4, however, used far more Metasploit commands than their Cluster 2 counterparts. The `set` tool proved especially difficult for the trainees as the median value for its use was 19, four times more than in other clusters. Cluster 4 trainees also utilized the rest of the Metasploit tools (`show`, `use`, `exploit`, `msfconsole`) more than other trainees. Their delays were also lower, suggesting they tried setting different Metasploit parameters in quick succession.

Cluster 4 command composition is similar to Cluster 2 from the bag of words matrix described in Section 5.1.2. The similarity is mainly caused by the *Alpha2018 – House of Cards KYPO Summer School* training as it forms a large part of both clusters. The difference is that Cluster 2 contains far more points than Cluster 4, which is almost exclusively formed by trainees from the training mentioned above.

5.3 Limitations

Our work's main limitation was the small sample size, as the clustering is usually applied on larger datasets. With the small dataset, the number of clusters we found is limited as well as their size. Larger and more numerous clusters would help us find new approaches or reinforce the characteristics of the ones we found. The small size of the dataset is caused mainly by the difficult process of obtaining training log files. Organizing these training sessions requires significant human resources as well as a large portion of instructors' time.

Another limitation was judging the success of trainees' approaches. As the information about what tasks trainees completed was unavailable, we had to evaluate their progress based on the number of commands associated with each task. If the trainees used commands from the later part of the training, we judged the cluster successful. To confirm our assessment, we checked how many times they used the `cat` tool with the final file location as the option, which was the last step in a successful session. Still, our approach is not flawless as some trainees could have used the required tools but still fail to complete the task.

6 Conclusion

In this thesis, we explored different approaches of trainees who solved tasks in cybersecurity training. We applied a clustering algorithm on the data we extracted from command logs, then we visualized and interpreted the results.

Most of the trainees used similar tools to complete their training. The `nmap`, `set`, and `ssh` tools form the basis of many trainees' progression. On the other hand, there were enough differences in what tools were difficult for trainees, how they switched between tools and changed their options, and how far they were able to progress to reveal several distinct groups of trainees.

The most challenging part for a large part of the trainees was the Metasploit environment and the `set` tool specifically. Many players struggled with finding the final file to complete the training and used a combination of `ls` and `cd` tools to list through different directories. We also identified a cluster of trainees who experienced trouble with the `john` tool later in their training.

Outliers produced by clustering proved to be useful when discovering trainees who chose tools different from the most, such as `nikto` tool instead of `nmap`. The fact that alternative approaches are mostly designated as outliers further proves that most trainees used the same tools to complete their training.

The results show that clustering analysis is a viable method to gain interesting and potentially useful insight into the data from cybersecurity logs, such as the most utilized tools, standard tool and option combinations, difficult parts of the training, and trainees' approach when encountering difficulties. The two separate feature matrices also proved useful as they allowed us to view the data from different perspectives.

The thesis's contribution is in the insights into the trainees' progress it provides for the instructors. Information about the training's challenging parts could be valuable to the instructors when identifying problematic tasks or developing new ones. Instructors could also intervene to help the struggling trainees if they exhibit signs typical for trainees from low-performing clusters.

6.1 Future work

There are multiple ways in which this research may be continued. The obvious one is to apply the clustering algorithm to more data, which could lead to new clusters or reinforce existing ones. Additional data could also provide information we did not consider in this thesis, such as the exact level player reached during the training and how long it took, or how many hints he took while completing a task.

One other possibility is to improve the bag of words model and assign different weights to specific tools to increase or decrease their importance. Another problem is that the bag of words model disregards the order of commands. To get a better idea of how combinations of different tools affect the trainees' progress, we could use the bag of n-grams model instead, as it counts occurrences of two or more consecutive commands.

Another way to improve our results is to use different clustering algorithms. Hierarchical clustering could be used as it is suitable for small datasets. We could also compare results from several algorithms either to determine the most suitable one or to get different views on the data.

Our results could also be used as a basis for a recommender system that would provide hints for stuck trainees. If a trainee needs help, the system could recommend a hint that helped some other trainee from the same cluster.

Bibliography

[1] Osama Abu Abbas. "Comparisons Between Data Clustering Algorithms." In: *International Arab Journal of Information Technology (IAJIT)* 5.3 (2008).

[2] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. "On the Surprising Behavior of Distance Metrics in High Dimensional Space". In: *Database Theory — ICDT 2001*. Ed. by Jan Van den Bussche and Victor Vianu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 420–434. ISBN: 978-3-540-44503-6.

[3] Mihael Ankerst et al. "OPTICS: Ordering Points to Identify the Clustering Structure". In: *SIGMOD Rec.* 28.2 (June 1999), pp. 49–60. ISSN: 0163-5808. DOI: 10.1145/304181.304187.

[4] P. Berkhin. "A Survey of Clustering Data Mining Techniques". In: *Grouping Multidimensional Data: Recent Advances in Clustering*. Ed. by Jacob Kogan, Charles Nicholas, and Marc Teboulle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 25–71. ISBN: 978-3-540-28349-2. DOI: 10.1007/3-540-28349-8_2.

[5] Kevin Beyer et al. "When Is "Nearest Neighbor" Meaningful?" In: *Database Theory — ICDT'99*. Ed. by Catriel Beeri and Peter Buneman. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 217–235. ISBN: 978-3-540-49257-3.

[6] Derya Birant and Alp Kut. "ST-DBSCAN: An algorithm for clustering spatial–temporal data". In: *Data & Knowledge Engineering* 60.1 (2007). Intelligent Data Mining, pp. 208–221. ISSN: 0169-023X. DOI: https://doi.org/10.1016/j.datak.2006.01.013.

[7] François Bouchet et al. "Clustering and profiling students according to their interactions with an intelligent tutoring system fostering self-regulated learning". In: *Journal of Educational Data Mining* 5.1 (2013), pp. 104–146.

[8] The SciPy community. *T-test SciPy implementation*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html [Online; accessed 2021-03-01].

[9] J. C. Dunn. "A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters". In: *Journal of Cybernetics* 3.3 (1973), pp. 32–57. DOI: 10.1080/01969727308546046.

[10] A. Dutt, M. A. Ismail, and T. Herawan. "A Systematic Review on Educational Data Mining". In: *IEEE Access* 5 (2017), pp. 15991–16005. DOI: 10.1109/ACCESS.2017.2654247.

[11] Andrew Emerson et al. "Cluster-Based Analysis of Novice Coding Misconceptions in Block-Based Programming". In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. SIGCSE '20. Portland, OR, USA: Association for Computing Machinery, 2020, pp. 825–831. ISBN: 9781450367936. DOI: 10.1145/3328778.3366924.

[12] Martin Ester et al. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *Knowledge Discovery and Data mining*. 1996, pp. 226–231. URL: http://www.aaai.org/Library/KDD/1996/kdd96-037.php.

[13] J. Han, J. Pei, and M. Kamber. *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2011. ISBN: 9780123814807. URL: https://books.google.sk/books?id=pQws07tdpjoC.

[14] Anna Huang. "Similarity measures for text document clustering". In: *Proceedings of the 6th New Zealand Computer Science Research Student Conference* (Jan. 2008).

[15] Julian Jang-Jaccard and Surya Nepal. "A survey of emerging threats in cybersecurity". In: *Journal of Computer and System Sciences* 80.5 (2014). Special Issue on Dependable and Secure Computing, pp. 973–993. ISSN: 0022-0000. DOI: https://doi.org/10.1016/j.jcss.2014.02.005.

[16] Jelena Jovanović et al. "Learning analytics to unveil learning strategies in a flipped classroom". English. In: *Internet and Higher Education* 33 (Apr. 2017), pp. 74–85. ISSN: 1096-7516. DOI: 10.1016/j.iheduc.2017.02.001.

[17] Trupti M Kodinariya and Prashant R Makwana. "Review on determining number of Cluster in K-Means Clustering". In: *International Journal* 1.6 (2013), pp. 90–95.

[18] S. Lloyd. "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. DOI: 10.1109/TIT.1982.1056489.

[19] T Soni Madhulatha. "An overview on clustering methods". In: *IOSR Journal of Engineering* 2.4 (2012), pp. 719–725. DOI: 10.9790/3021-0204719725.

[20] Jessica McBroom et al. “Mining Behaviours of Students in Autograding Submission System Logs.” In: *International Educational Data Mining Society* (2016).

[21] Mark J. Nelson and Amy K. Hoover. “Notes on Using Google Colaboratory in AI Education”. In: ITiCSE ’20. Trondheim, Norway: Association for Computing Machinery, 2020, pp. 533–534. ISBN: 9781450368742. DOI: 10.1145/3341525.3393997.

[22] Radek Pelánek et al. “Measuring Item Similarity in Introductory Programming”. In: *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*. L@S ’18. London, United Kingdom: Association for Computing Machinery, 2018. ISBN: 9781450358866. DOI: 10.1145/3231644.3231676.

[23] Keith Quille and Susan Bergin. “Promoting a Growth Mindset in CS1: Does One Size Fit All? A Pilot Study”. In: ITiCSE ’20. Trondheim, Norway: Association for Computing Machinery, 2020, pp. 12–18. ISBN: 9781450368742. DOI: 10.1145/3341525.3387361.

[24] Rapid7. *Metasploit*. https://www.metasploit.com/ [Online; accessed 2021-03-01].

[25] Jiří Řihák and Radek Pelánek. “Measuring Similarity of Educational Items Using Data on Learners’ Performance”. eng. In: *Proceedings of the 10th International Conference on Educational Data Mining*. Wuhan, China.: International Educational Data Mining Society, 2017, pp. 16–23.

[26] C. Romero and S. Ventura. “Educational Data Mining: A Review of the State of the Art”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40.6 (2010), pp. 601–618. DOI: 10.1109/TSMCC.2010.2053532.

[27] C. Romero et al., eds. *Handbook of educational data mining*. English. Chapman and Hall/CRC data mining and knowledge discovery series. CRC Press, 2011. ISBN: 978-1-4398-045-7-5.

[28] Jörg Sander et al. “Automatic Extraction of Clusters from Hierarchical Clustering Representations”. In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Kyu-Young Whang et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 75–87. ISBN: 978-3-540-36175-6. DOI: 10.1007/3-540-36175-8_8.

[29] Ali Seyed Shirkhorshidi, Saeed Aghabozorgi, and Teh Ying Wah. "A Comparison Study on Similarity and Dissimilarity Measures in Clustering Continuous Data". In: *PLOS ONE* 10.12 (2015), pp. 1–20. DOI: 10.1371/journal.pone.0144059.

[30] Alexander Strehl, Joydeep Ghosh, and Raymond Mooney. "Impact of similarity measures on web-page clustering". In: *Workshop on artificial intelligence for web search (AAAI 2000)*. Vol. 58. 2000, pp. 58–64.

[31] Wang Tang, Dechang Pi, and Yun He. "A Density-Based Clustering Algorithm with Sampling for Travel Behavior Analysis". In: *Intelligent Data Engineering and Automated Learning – IDEAL 2016*. Ed. by Hujun Yin et al. Cham: Springer International Publishing, 2016, pp. 231–239. ISBN: 978-3-319-46257-8.

[32] Masaryk University. *KYPO Cyber Range Platform*. https://crp.kypo.muni.cz/ [Online; accessed 2021-03-01].

[33] Matthew Yee-King et al. "Examining Student Coding Behaviours in Creative Computing Lessons Using Abstract Syntax Trees and Vocabulary Analysis". In: *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE '20. Trondheim, Norway: Association for Computing Machinery, 2020, pp. 273–279. ISBN: 9781450368742. DOI: 10.1145/3341525.3387408.

[34] Hezheng Yin, Joseph Moghadam, and Armando Fox. "Clustering Student Programming Assignments to Multiply Instructor Leverage". In: *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*. L@S '15. Vancouver, BC, Canada: Association for Computing Machinery, 2015, pp. 367–372. ISBN: 9781450334112. DOI: 10.1145/2724660.2728695.

A Content of the thesis archive

This thesis is available at `https://is.muni.cz/th/fefjq/` and contains the dataset and Jupyter notebooks we used in our research, together with the resulting visualizations. The archive has the following structure:

- `dataset`: Folder containing selected 29 log files from cybersecurity training in JSON format.
- `src`: Folder containing Jupyter notebooks created for our research. It also contains folder with their Python version.
  - `log_parser.ipynb`: A Jupyter notebook for parsing the log files.
  - `clustering_utilities.ipynb`: A Jupyter notebook containing functions for clustering, result extraction, and visualization.
  - `bag_of_words.ipynb`: A Jupyter notebook for clustering the bag of words matrix and results visualization.
  - `selected_features.ipynb`: A Jupyter notebook for clustering the selected features matrix and results visualization.
- `visualizations`: Folder containing visualizations of the results.
  - `bag_of_words`: Visualizations produced by the bag of words matrix.
  - `selected_features`: Visualizations produced by the selected features matrix.
- `LICENSE.md`: An MIT license for the scripts.
- `README.md`: A file containing archive information.
- `requirements.txt`: Python packages necessary to run the scripts.