

Using endpoints process information for malicious behavior detection

K.J. Wijnands

1228374

USING ENDPOINTS PROCESS INFORMATION FOR MALICIOUS BEHAVIOR DETECTION

by

K.J. Wijnands
1228374

in partial fulfillment of the requirements for the degree of

Master of Science
in Technology, Policy and Management

at the Delft University of Technology,
to be defended publicly on Thursday September 24, 2015 at 15:00.

Supervisor:	Prof. dr. ir. J. van den Berg	
Thesis committee:	Dr. ir. S. Verwer,	TU Delft
	Dr. V. Dignum,	TU Delft
	Dr. M. Warnier,	TU Delft
	M. Boone,	Fox-IT

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.



FOX IT



TU Delft

Delft
University of
Technology

Executive Summary

In the last years the impact of malware has become a huge problem. Each year, more and more new malware samples are discovered [2]. And the malware is becoming more sophisticated, for example ransomware. Ransomware encrypts personal documents, such as photos and word documents, and asks money to be able to decrypt these files, hence the name. Malware is not only used for financial gain at the backs of consumers. Sophisticated targeted attacks at enterprise is not uncommon, for example the Sony hack.

Although there are many security solutions which should protect endpoints, malware infections still occur. The reason for this has to do with the way current security solutions work. Most of these security solutions act upon known malware behavior and signatures. However when new malware is released and the behavior and signature is still unknown the security solutions cannot protect the endpoint against these infections.

To be able to overcome this problem a new method for malware detection should be developed. This detection method should be able to detect malicious behavior without prior knowledge. In scientific literature this type of detection is called anomaly detection [26, 38, 57, 58]. Anomaly detection uses the gathered data to construct a model for normal behavior. Any deviation from the defined normal behavior is seen as an anomaly.

At Fox-IT, an IT security company based in the Netherlands, a new security solution is developed, called FoxGuard. This security solution has the ability to block and allow process activity based on a set of rules. FoxGuard also has the ability to log very detailed low level information of all the processes running on a system. This information include actions such as filesystem actions and registry actions. For a more detailed explanation of the data FoxGuard can gather read section 4.1.

In this master thesis an explorative research is conducted on using anomaly detection to detect malicious process on an endpoint by using the detailed process information FoxGuard can collect. The main research question to be answered is:

How can anomaly based detection be used for detecting unknown malicious processes based on the detailed process information gathered on a single endpoint?

To answer this question first a literature research was conducted on the use of anomaly detection for detecting malicious process in scientific literature, see chapter 2. The main conclusion from the literature study is that using process information combined with tree based representations, large quantities of data can be stored in a compact representation. These compact representations can aid the security officer in graphically analyzing the processes on an endpoint and hereby possibly spotting deviations.

In chapter 3 the design requirements of the developed system are analyzed. The conclusion of this analysis is that the amount of data used should be reduced. Not only does it prevent the chances of generating a detection method in which overfitting occurs, reducing the data also reduces the need for huge amounts of memory, storage, processing power and network data send.

The collection and preparation of the data is discussed in chapter 4. We have collected four clean datasets, a complete dataset contains one complete bootcycle, and five malware datasets. To generate

the malware dataset the following malware was used: a banking malware, a Remote Access Trojan and a sample of Zeus.

The collected data is aggregated, such that a dataframe remains containing per process the number of times it triggered the following activities: filesystem, registry, process create, thread create, object callback and module load. Furthermore it contained the unique process id of the parent process.

As the difference between the number of times process activities were triggered the data was normalized between 0 and 10, such that the data of the process activities becomes comparable between each other. A k-means clustering algorithm was applied on the process activities to assign every process to cluster with likewise processes.

The aggregated and processed data is used to generate process trees, section 5.1 and heatmaps, section 5.3. These two tools provide a graphical representation of the processes. In a heatmap a security officer can easily spot the processes with high number of process activities per second compared to other processes.

In analyzing the process tree deviations were spotted in the top part of the tree, providing proof that an expert can use the process tree to easily spot deviations in the top level. However due to the huge number of nodes present in a tree and the difference in computer usage each day, finding deviations in the lower levels of the tree proofed to be difficult.

Analyzing the process trees from the malware sets proofed again that the process tree can help in finding deviations. The rat malware processes were clearly visible as deviations on the process tree. Further more the analysis showed that all malware samples ran could be found in the same part of the process tree.

Chapter 6 explains the three algorithms used to calculate the distances between processes in the clean and malware set. These calculated distance are used for marking a process malicious or benign. A process is marked malicious if it is above a set threshold value. To set these threshold values we used the mean and 75%, 80%, 85%, 90% and 95% quantile.

All threshold values and algorithms were test and the True Postive Rate, False Negative Rate and the Accuracy were calculated. The outcome of all experiments is shown in chapter 7. In figure 1 the True Positive Rate, False Positive Rate and Accuracy for all algorithms is shown. As can be seen in the figure the malicious processes of the banking malware and rat malware could partly be detect. The highest True Positive rate gained is 0.917 using algorithm 1 and 3 on the banking malware. However paired with this is a high False positive Rate. However the Zeus malware was not detect.

In chapter 8 the conclusion and recommendations of this thesis are presented. The main shortcoming for the conducted research is way in which the collection of the data was done. By using two different machines differences in processes from the same executable were noticeable. This had to do with the fact that the running times for these processes differs. For future research this experiment should be repeated by collecting data on one machine. Although the shortcoming had its effects on the collected data the proposed algorithms showed the ability to detect malicious processes from at least two out of the three malware types. Furthermore the analysis of the process trees showed us that, although limited, deviations can be detected.

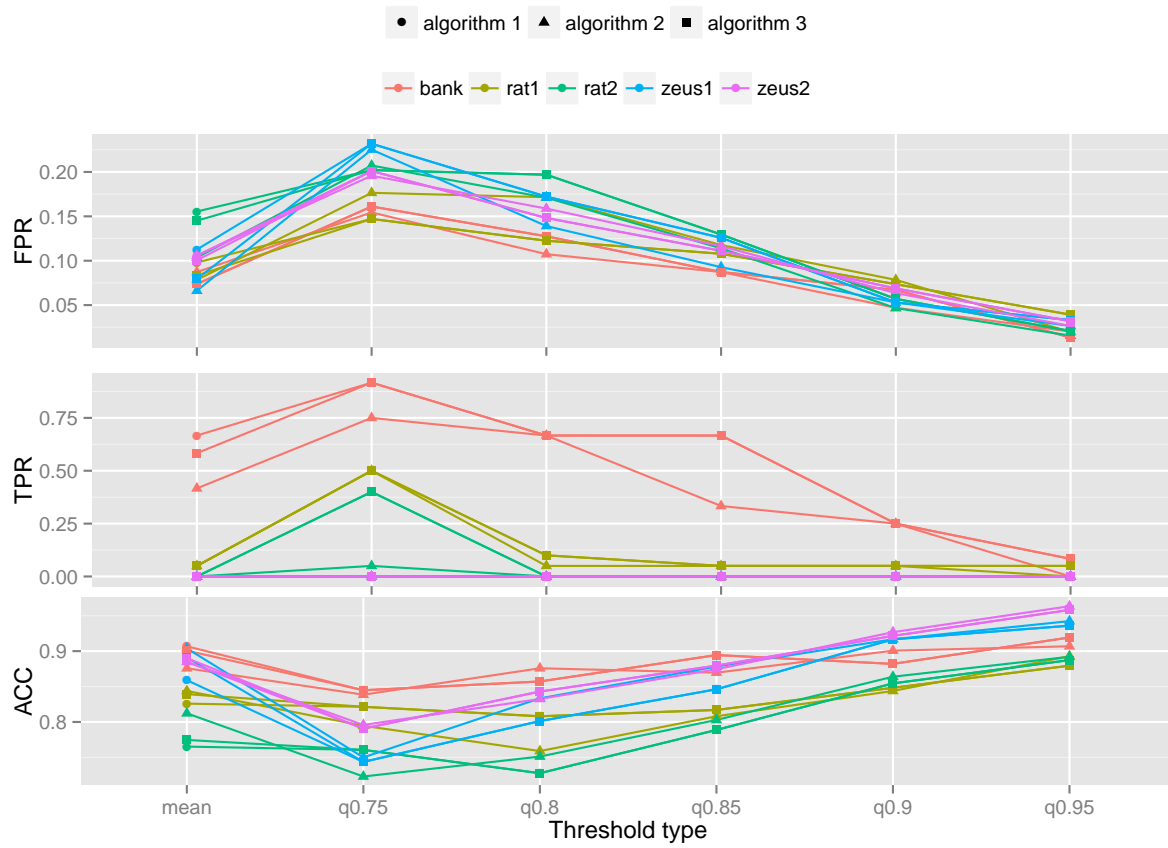


Figure 1: Plots of FPR, TPR and ACC of all malware and all methods

Contents

List of Figures	XI
List of Tables	XV
1 Introduction	1
1.1 Introduction	1
1.2 Malware detection in practice	2
1.2.1 Methods used by anti-virus solutions	3
1.2.2 Deceiving current anti-virus solutions	3
1.3 Malware detection in scientific literature	4
1.4 Solution direction	5
1.5 TPM relevance	5
1.6 Knowledge gap: Difference between science and practice	6
1.7 Research goal and questions	7
1.8 Research approach	8
1.9 Thesis outline	8
2 Literature research	9
2.1 Anomaly Detection	9
2.1.1 Disadvantages of anomaly detection	11
2.2 Graph methods	11
2.2.1 Graphs	11
2.2.2 Time based graphs	14
2.3 Graph comparison	18
2.4 Conclusion	19
3 Design requirements and feature selection	20
3.1 Design requirements based on stakeholder analysis	20
3.1.1 Stakeholder analysis	20
3.1.2 Implications for design requirements	20
3.2 Design requirements imposed by privacy law and regulation	22
3.3 Design requirements based on literature	23
3.4 Design requirements based on the data	23
3.5 Conclusion	23

4	Data collection and preparation	24
4.1	The data	24
4.1.1	Filesystem event	24
4.1.2	Registry event	24
4.1.3	Process create event	25
4.1.4	Process exit event	25
4.1.5	Thread create event	25
4.1.6	Thread exit event	25
4.1.7	Module load event	25
4.1.8	Object callback events	25
4.2	Collecting the data	25
4.2.1	Clean datasets	25
4.2.2	Malware datasets	26
4.2.3	Malware used	26
	4.2.3.1 Bank malware: Dridex	27
	4.2.3.2 Remote Access Trojan	27
	4.2.3.3 Zeus Trojan	27
4.3	Collected datasets	27
4.3.1	Collected clean datasets	27
4.3.2	Collected malware datasets	27
4.4	Data selection	27
4.5	Data preparation	30
4.6	Malware processes	31
4.6.1	Banking malware	31
4.6.2	Rat malware session 1	32
4.6.3	Rat malware session 2	32
4.6.4	Zeus malware session 1	32
4.6.5	Zeus malware session 2	36
5	Exploring the data	37
5.1	Process trees	37
5.1.1	Constructing the process trees	37
5.1.2	Process trees from the clean datasets	38
5.1.3	Process trees from the malware datasets	40
5.1.4	Slimming the process trees	40
5.1.5	Analyzing the process trees	40
	5.1.5.1 Analysing the malicious parts of the process tree	44
5.1.6	Conclusion	44
5.2	Process activities	54
5.2.1	Process activities clean data	54
5.2.2	Process activities malware data	55
5.2.3	Conclusion	57
5.3	Heatmaps	57
5.3.1	Heatmaps from the clean datasets	57
5.3.2	Heatmaps from the malware datasets	58
5.3.3	Analyzing the heatmaps	58
5.3.4	Analyzing the malware heatmaps	62
5.3.5	Conclusion	62
5.4	Possible benign process	65
5.5	Conclusion	66

6	Building the detection method	67
6.1	Comparing methods	67
6.1.1	Compare method 1	68
6.1.2	Compare method 2	68
6.1.3	Compare method 3	69
6.2	Ranking of malicious marked processes	69
6.3	Running times	69
6.4	Conclusion	70
7	Evaluation	71
7.1	Evaluation Set-up	71
7.2	Evaluation	72
7.2.1	Algorithm 1	72
7.2.1.1	Algorithm 1:Banking malware	72
7.2.1.2	Algorithm 1: Rat session 1	79
7.2.1.3	Algorithm 1: Rat session 2	85
7.2.1.4	Algorithm 1: Zeus session 1	89
7.2.1.5	Algorithm 1: Zeus session 2	93
7.2.1.6	Conclusion Algorithm 1	97
7.2.2	Algorithm 2	99
7.2.2.1	Algorithm 2: Banking malware	99
7.2.2.2	Algorithm 2: Rat 1	102
7.2.2.3	Algorithm 2: Rat 2	105
7.2.2.4	Algorithm 2: Zeus 1	108
7.2.2.5	Algorithm 2: Zeus 2	111
7.2.2.6	Conclusion Algorithm 2	114
7.2.3	Algorithm 3	115
7.2.3.1	Algorithm 3:banking malware	115
7.2.3.2	Algorithm 3: rat 1	116
7.2.3.3	Algorithm 3: rat 2	116
7.2.3.4	Algorithm 3: Zeus 1	118
7.2.3.5	Algorithm 3: Zeus 2	118
7.2.3.6	Conclusion Algorithm method 3	121
7.3	Conclusion	121
8	Conclusion	124
8.1	Reflection on research questions	124
8.2	Shortcomings and recommendations for future research	126
9	Bibliography	128
	Appendices	133
A	Collected data	134
A.1	File system event	134
A.2	Registry event	135
A.3	Process create event	136
A.4	Process exit event	137
A.5	Thread create event	137
A.6	Thread exit event	138
A.7	Module load event	138
A.8	Object callback events	138

B	Process trees	140
B.1	Cut Process trees from clean datasets	140
B.2	Cut process trees from malware datasets	144
B.3	Merged process trees	149
C	Process Activities	159
C.1	Process activities clean dataset	159
C.2	Process activities malware dataset	160
D	Heatmaps	168
D.1	Heatmaps from the clean datasets	168
D.1.1	Analyzing the heatmaps	172
D.2	Benign process analysis	182
E	INPUT IS HERE	185
F	Evaluation Algorithm 1	186
F.1	Banking malware	187
F.2	Rat malware session 1	188
F.3	Rat malware session 2	189
F.4	Zeus malware session 1	190
F.5	Zeus malware session 2	191
G	Evaluation Algorithm 2	192
G.1	Banking malware	193
G.2	Rat malware session 1	194
G.3	Rat malware session 2	195
G.4	Zeus malware session 1	196
G.5	Zeus malware session 2	197
H	Evaluation Algorithm 3	198
H.1	Banking malware	199
H.2	Rat malware session 1	200
H.3	Rat malware session 2	201
H.4	Zeus malware session 1	202
H.5	Zeus malware session 2	203
H.5.1	Malicious heatmaps	204
I	Ranked malicious processes	218
I.1	Algorithm 1: Ranked malicious marked processes	218
I.2	Algorithm 2: Ranked malicious marked processes	229
I.3	Algorithm 3: Ranked malicious marked processes	239
J	Algorithm 3: Process trees	249

List of Figures

1	Plots of FPR, TPR and ACC of all malware and all methods	V
1.1	New malware samples per year [2]	2
1.2	Showing the problem of finding the malicious processes	6
1.3	A graphical presentation of the thesis layout	8
1.4	Thesis outline	8
2.1	An example of a traffic-dependency graph [70]	12
2.2	An example of a process tree [66]	13
2.3	An example of a graph model [66]	14
2.4	An example of a honeypot hierarchical probabilistic automaton [63]	14
2.5	Aggregated Tree for a destination profile [64].	15
2.6	A traffic profile T_i , compared to T_{i-1}, T_{i-2} and T_{i-3} showing a DDoS UDP attack [64] .	15
2.7	Same DDoS UDP Flood attack with Phase Space Embedding Analysis [64]	15
2.8	A partial view of an aggregated tree with a traffic volume threshold of $\alpha = 5\%$ [35] . . .	16
2.9	The change in the entropy of H_{ip} of a tree without and with anomaly [34]	17
4.1	K-means plot	32
5.1	First 5 levels of win8 1604	39
5.2	First 5 levels of bank malware	42
5.3	Part of 1604 avond showing the mentioned processes which are only visible in the 1604 avond and 1804 datasets.	44
5.4	Part of process tree containing the malicious processes of the banking malware	45
5.5	Part of process tree containing the malicious processes of rat 1	46
5.6	Part of process tree containing the malicious processes of rat 2	47
5.7	Part of process tree containing the malicious processes of zeus 1	48
5.8	Part of process tree containing the malicious processes of zeus 2	49
5.9	Part of process tree containing the same part as malicious 1604 avond	50
5.10	Part of process tree containing the same part as malicious 1604	51
5.11	Part of process tree containing the same part as malicious 1704	52
5.12	Part of process tree containing the same part as malicious 1804	53
5.13	Part from the heatmap from dataset win8 1604	58
5.14	Another Part from the heatmap from dataset win8 1604	58
5.15	Part from the heatmap from dataset zeus session 2	59
5.16	Part 1 from the heatmap from 1604 avond with the malicious processes	59
5.17	Part 3 from the heatmap from banking malware heatmap with the malicious processes .	63
5.18	Part 5 from the heatmap from rat session 1 with the malicious processes	64
5.19	Heatmap showing process 243	65

5.20	Heatmap showing the processes	65
7.1	Plot of the ACC, FPR and TPR of banking malware	73
7.2	The process tree of the banking malware set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.	78
7.3	Plot of the ACC, FPR and TPR of rat session 1	79
7.4	The process tree of the rat session 1 set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.	84
7.5	Plot of the ACC, FPR and TPR of rat session 2	85
7.6	The process tree of the rat session 2 dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.	88
7.7	Plot of the ACC, FPR and TPR of Zeus session 1	89
7.8	The process tree of the Zeus session 1 dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.	92
7.9	Plot of the ACC, FPR and TPR of Zeus session 2	93
7.10	The process tree of the Zeus session 2 dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.	96
7.11	Plot of the ACC, FPR and TPR of all malware sets using method 1	98
7.12	Plot of the ACC, FPR and TPR of bank using method 2	99
7.13	The process tree of the banking malware dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.	101
7.14	Plot of the ACC, FPR and TPR of rat session 1 using method 2	102
7.15	The process tree of the rat session 1 dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.	104
7.16	Plot of the ACC, FPR and TPR of rat session 2 using method 2	105
7.17	The process tree of the rat session 2 malware dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.	107
7.18	Plot of the ACC, FPR and TPR of Zeus session 1 using method 2	108
7.19	The process tree of the Zeus session 1 dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.	110
7.20	Plot of the ACC, FPR and TPR of Zeus session 2 using method 2	111
7.21	The process tree of the Zeus session 2 dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.	113
7.22	Plot of the ACC, FPR and TPR of all malware sets using algorithm 2	114
7.23	Plot of the ACC, FPR and TPR of bank using method 3	115
7.24	Plot of the ACC, FPR and TPR of rat session 1 using method 3	117
7.25	Plot of the ACC, FPR and TPR of rat session 2 using method 3	118
7.26	Plot of the ACC, FPR and TPR of Zeus session 1 using method 3	119
7.27	Plot of the ACC, FPR and TPR of Zeus session 2 using method 3	120
7.28	Plot of the ACC, FPR and TPR of all malware sets using method 3	121
7.29	Plots of FPR, TPR and ACC of all malware and all methods	122
B.1	First 5 levels of win8 1604 avond	141
B.2	First 5 levels of win8 1704	142
B.3	First 5 levels of win8 1804 avond	143
B.4	First 5 levels of rat malware session 1	145
B.5	First 5 levels of rat malware session 2	146
B.6	First 5 levels of zeus malware session 1	147
B.7	First 5 levels of zeus malware session 2	148
B.8	1604 avond merged tree	150
B.9	1604 merged tree	151
B.10	1704 merged tree	152

B.11	1804 merged tree	153
B.12	bank malware merged tree	154
B.13	rat malware session 1 merged tree	155
B.14	rat malware session 2 merged tree	156
B.15	Zeus malware session 1 merged tree	157
B.16	Zeus malware session 2 merged tree	158
C.1	Boxplot win 8 1604	159
C.2	Boxplot win 8 1604 avond	160
C.3	Boxplot win 8 1704	161
C.4	Boxplot win 8 1804	162
C.5	Boxplot bank malware	163
C.6	Boxplot Zeus malware session 1	164
C.7	Boxplot Zeus malware session 2	165
C.8	Boxplot Rat malware session 1	166
C.9	Boxplot Rat malware session 2	167
D.1	Heatmap win8 1604 dataset	169
D.2	Heatmap win8 1604 dataset split test	170
D.3	Heatmap win8 1604 avond dataset	171
D.4	Heatmap win8 1704 dataset	173
D.5	Heatmap win8 1804 dataset	174
D.6	Part from 1604 avond	175
D.7	Part from 1604	176
D.8	Part from 1704	176
D.9	Part from 1804	177
D.10	Part from bank malware	178
D.11	Part from rat 1 malware	178
D.12	Part from rat 2 malware upper part	179
D.13	Part from rat 2 malware bottom part	179
D.14	Part from zeus 1 malware bottom part	180
D.15	Part from zeus 1 malware upper part	180
D.16	Part from zeus 2 malware bottom part	181
D.17	Heatmap showing the process	183
H.1	Part 1 from the heatmap from the banking malware with the malicious processes	204
H.2	Part 2 from the heatmap from banking malware with the malicious processes	205
H.3	Part 1 from the heatmap from rat session 1 with the malicious processes	206
H.4	Part 2 from the heatmap from rat session 1 with the malicious processes	207
H.5	Part 3 from the heatmap from rat session 1 with the malicious processes	208
H.6	Part 4 from the heatmap from rat session 1 with the malicious processes	209
H.7	Part 1 from the heatmap from rat session 2 with the malicious processes	210
H.8	Part 2 from the heatmap from rat session 2 with the malicious processes	211
H.9	Part 3 from the heatmap from rat session 2 with the malicious processes	212
H.10	Part 4 from the heatmap from rat session 2 with the malicious processes	213
H.11	Part 5 from the heatmap from from rat session 2 with the malicious processes	214
H.12	Part 1 from the heatmap from zeus session 1 with the malicious processes	215
H.13	Part 2 from the heatmap from zeus session 1 with the malicious processes	216
H.14	Part 1 from the heatmap from zeus session 2 with the malicious processes	217
J.1	The process tree of the banking malware dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.	250

J.2	The process tree of the rat session 1 malware dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.	251
J.3	The process tree of the rat session 2 malware dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.	252
J.4	The process tree of the zeus session 1 malware dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.	253
J.5	The process tree of the zeus session 1 malware dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.	254

List of Tables

3.1	Overview of stakeholders	21
4.1	Clean datasets	28
4.2	Malware datasets	29
4.3	Malicious processes banking malware	33
4.4	Malicious processes rat session 1	33
4.5	Malicious processes rat malware session 2	34
4.6	Zeus session 1 malware samples	34
4.7	Zeus session 2 malware samples	35
5.1	Number of nodes on each depth per clean dataset	38
5.2	Number of nodes on each depth per malware dataset	40
5.3	summary of strange pid 4 behaviour	43
5.4	Normalized number of events for process id 4	43
5.5	Some information on process 133 from the 1604 avond dataset. The comment line options show that it is the F-secure program	43
5.6	Showing the summary of the activities of the 1604 dataset	54
5.7	Showing the summary of the activities of the 1604 avond dataset	54
5.8	Showing the summary of the activities of the 1704 dataset	54
5.9	Showing the summary of the activities of the 1804 dataset	55
5.10	Showing the summary of the activities of the bank malware dataset	55
5.11	Showing the summary of the activities of the rat session 1 malware dataset	56
5.12	Showing the summary of the activities of the rat session 2 malware dataset	56
5.13	Showing the summary of the activities of the zeus session 1 malware dataset	56
5.14	Showing the summary of the activities of the zeus session 2 malware dataset	56
5.15	Starting of process 459 and 460 in win8 1604 avond	61
5.16	Process 459 and 460 activity	61
6.1	Running times (in seconds) of comparison methods using large dataset and banking malware	69
6.2	Running times (in seconds) of comparison methods using large datasets	70
7.1	Outcome using different threshold values for bank malware using method1	72
7.2	The five highest ranked benign processes for banking malware using algorithm 1.	73
7.3	Showing the processes from other datasets from the same executable as process 27 from the banking malware	74
7.4	Showing a selection the processes from other datasets from the same executable as process 119 from the banking malware	75

7.5	Calculate normalized values for the process activities for the same executable as process 119 in the banking malware set	76
7.6	Outcome using different threshold values for rat1 malware using method1	79
7.7	The five highest ranked benign processes for rat session 1 using algorithm 1.	80
7.8	The calculated normalized values of processes with the same executable as process 109 in the rat session 1 data	81
7.9	Summary of the process activities of the processes with the same executable as process 95 and 204 in the rat session 1 dataset	82
7.10	Showing the processes from the same executable as process 95 and 204 in the rat session 1 dataset	83
7.11	Outcome using different threshold values for rat2 malware using method1	85
7.12	The five highest ranked benign processes for rat session 2 using algorithm 1.	85
7.13	Summary of the process activities of the malicious processes of the rat session 2 dataset	86
7.14	Summary of the process activities of the malicious processes of the rat session 1 dataset	86
7.15	Process information from the executable belonging to process 161 of the rat session 2 malware dataset	87
7.16	Process information from the executable belonging to process 206 of the rat session 2 malware dataset	87
7.17	Outcome using different threshold values for zeus1 malware using method1	89
7.18	A summary of the process activities of the malicious processes of the Zeus session 1 malware dataset	90
7.19	Summary of the process activity of the clean 1604 dataset for comparison.	90
7.20	The five highest ranked benign processes for zeus session 1 using algorithm 1.	90
7.21	Outcome using different threshold values for zeus2 malware using method1	93
7.22	Summary of the process activities of the zeus session 2 dataset	94
7.23	The five highest ranked benign processes for zeus session 2 using algorithm 1.	94
7.24	Summary of process activity of process id 116 and 130 of the Zeus session 2 malware set	94
7.25	Summary of the process activities of the executable belonging to process 116 and 130 in the zeus session 2 data from all datasets	94
7.26	Outcome using different threshold values for bank malware using method2	99
7.27	The five highest ranked benign processes for banking malware using algorithm 2.	100
7.28	Outcome using different threshold values for rat1 malware using method2	102
7.29	The five highest ranked benign processes for rat session 1 using algorithm 2.	103
7.30	Summary of normalized process activities for process id 110 from the rat session 1	103
7.31	Summary of the process activities of all processes from the same executable as process 110 in rat session 1 dataset.	103
7.32	Outcome using different threshold values for rat2 malware using method2	105
7.33	The five highest ranked benign processes for rat session 2 using algorithm 2.	105
7.34	Overview of the normalized number of activities for the processes.	106
7.35	Outcome using different threshold values for zeus1 malware using method2	108
7.36	The five highest ranked benign processes for zeus session 1 using algorithm 2.	108
7.37	Outcome using different threshold values for zeus2 malware using method2	111
7.38	The five highest ranked benign processes for zeus session 2 using algorithm 2.	112
7.39	Outcome using different threshold values for bank malware using method3	115
7.40	The five highest ranked benign processes for banking malware using algorithm 3.	116
7.41	Outcome using different threshold values for rat1 malware using method3	116
7.42	The five highest ranked benign processes for rat session 1 using algorithm 3.	116
7.43	Outcome using different threshold values for rat2 malware using method3	117
7.44	The five highest ranked benign processes for rat session 2 using algorithm 3.	117
7.45	Outcome using different threshold values for zeus1 malware using method3	118
7.46	The five highest ranked benign processes for zeus session 1 using algorithm 3.	119
7.47	Outcome using different threshold values for zeus2 malware using method3	119

7.48	The five highest ranked benign processes for zeus session 2 using algorithm 3.	120
7.49	Overview of the number highest TPR and lowest FPRs	123
A.1	File system data fields	135
A.2	Registry data fields	136
A.3	Process create data fields	137
A.4	Process exit data fields	137
A.5	Thread create data fields	137
A.6	Thread exit data fields	138
A.7	Module load data fields	138
A.8	Object callbacks data fields	139
F.1	Outcome using different threshold values for bank malware using algorithm 1	187
F.2	Outcome using different threshold values for rat1 malware using algorithm 1	188
F.3	Outcome using different threshold values for rat2 malware using algorithm 1	189
F.4	Outcome using different threshold values for zeus1 malware using algorithm 1	190
F.5	Outcome using different threshold values for zeus2 malware using algorithm 1	191
G.1	Outcome using different threshold values for bank malware using algorithm 2	193
G.2	Outcome using different threshold values for rat1 malware using algorithm 2	194
G.3	Outcome using different threshold values for rat2 malware using algorithm 2	195
G.4	Outcome using different threshold values for zeus1 malware using algorithm 2	196
G.5	Outcome using different threshold values for zeus2 malware using algorithm 2	197
H.1	Outcome using different threshold values for bank malware using algorithm d3	199
H.2	Outcome using different threshold values for rat1 malware using algorithm 3	200
H.3	Outcome using different threshold values for rat2 malware using algorithm 3	201
H.4	Outcome using different threshold values for zeus1 malware using algorithm 3	202
H.5	Outcome using different threshold values for zeus2 malware using algorithm 3	203
I.1	Ranking of the processes based on the distance for banking malware using algorithm 1. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.	219
I.2	The five highest ranked benign processes for banking malware using algorithm 1. Also showing the clean datasets in which the executable is present.	220
I.3	Ranking of the processes based on the distance for rat session 1 using algorithm 1. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.	221
I.4	The five highest ranked benign processes for rat session 1 using algorithm 1. Also showing the clean datasets in which the executable is present.	222
I.5	Ranking of the processes based on the distance for rat session 2 using algorithm 1. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third q85, fourth the mean, the fifth q80 and the bottom line is q75.	223
I.6	The five highest ranked benign processes for rat session 2 using algorithm 1. Also showing the clean datasets in which the executable is present.	224
I.7	Ranking of the processes based on the distance for zeus session 1 using algorithm 1. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.	225

I.8	The five highest ranked benign processes for zeus session 1 using algorithm 1. Also showing the clean datasets in which the executable is present.	226
I.9	Ranking of the processes based on the distance for zeus session 2 using algorithm 1. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.	227
I.10	The five highest ranked benign processes for zeus session 2 using algorithm 1. Also showing the clean datasets in which the executable is present.	228
I.11	Ranking of the processes based on the distance for banking malware using algorithm 2. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third q85, fourth the mean, the fifth q80 and the bottom line is q75.	229
I.12	The five highest ranked benign processes for banking malware using algorithm 2. Also showing the clean datasets in which the executable is present.	230
I.13	Ranking of the processes based on the distance for rat session 1 using algorithm 2. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90 and the mean, third q85, the fourth q80 and the bottom line is q75.	231
I.14	The five highest ranked benign processes for rat session 1 using algorithm 2. Also showing the clean datasets in which the executable is present.	232
I.15	Ranking of the processes based on the distance for rat session 2 using algorithm 2. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.	233
I.16	The five highest ranked benign processes for rat session 2 using algorithm 2. Also showing the clean datasets in which the executable is present.	234
I.17	Ranking of the processes based on the distance for zeus session 1 using algorithm 2. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.	235
I.18	The five highest ranked benign processes for zeus session 1 using algorithm 2. Also showing the clean datasets in which the executable is present.	236
I.19	Ranking of the processes based on the distance for zeus session 2 using algorithm 2. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.	237
I.20	The five highest ranked benign processes for zeus session 2 using algorithm 2. Also showing the clean datasets in which the executable is present.	238
I.21	Ranking of the processes based on the distance for banking malware using algorithm 3. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.	239
I.22	The five highest ranked benign processes for banking malware using algorithm 3. Also showing the clean datasets in which the executable is present.	240
I.23	Ranking of the processes based on the distance for rat session 1 using algorithm 3. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.	241
I.24	The five highest ranked benign processes for rat session 1 using algorithm 3. Also showing the clean datasets in which the executable is present.	242

I.25	Ranking of the processes based on the distance for rat session 2 using algorithm 3. The first dark line shows the q95 border. The second line is the q90, third q85, fourth is the mean, the fifth q80 and the bottom line is q75.	243
I.26	The five highest ranked benign processes for rat session 2 using algorithm 3. Also showing the clean datasets in which the executable is present.	244
I.27	Ranking of the processes based on the distance for zeus session 1 using algorithm 3. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.	245
I.28	The five highest ranked benign processes for zeus session 1 using algorithm 3. Also showing the clean datasets in which the executable is present.	246
I.29	Ranking of the processes based on the distance for zeus session 2 using algorithm 3. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.	247
I.30	The five highest ranked benign processes for zeus session 2 using algorithm 3. Also showing the clean datasets in which the executable is present.	248

1

Introduction

1.1 Introduction

In the last few decades the rise of internet has had a huge impact on our global society. Today's world is largely dependent on the use of information technology and the internet. The amount and different types of devices interconnected by the internet is still rising. Not only our computer and mobile phone are connected to the internet, but devices as televisions, smart-meters (for electricity) and smartwatches as well. In the coming years more and more devices will be connected to the internet, for example your thermostat [8] or your fridge [12], creating the internet of things.

All these internet connected devices create huge potentials for new functions, it has its downside as well. All devices connected to the internet are the potential subjects of cyberattacks, which could end up in a loss of data, money or other inconveniences. These cyberattacks and infections of computers are a huge problem at the moment. This can be concluded from the vast amount of news headlines stating malware infections at municipalities, companies and personal users [14–16, 18, 19, 21].

Together with the rise of internet usage, the amount and characteristics of cyberattacks has changed. Nowadays these threats are becoming more complex and serious. Mostly used for financial or information gain. For example, in the last couple years malware threats were active, that used a compromised device for mining digital currencies, which could then be converted in money with a monetary value [50, 51].

Since 2013, a lot of *ransomware* attacks are present [47]. Ransomware is a malware that infects a users computer and persuades the owner to pay the malware author. This persuasion can be false claims against the user of copyright infringement or owning child pornography, but also, and the most common version, is to encrypt the users files. To be able to decrypt and gain access to the files, the victim has to pay money.

In the last few years, the frequency of cyberattacks and the cost of these cybercrimes have risen [6] and cybercrime has become a highly organized business [43]. Norton Security claims, that the damage of cybercrime aimed at consumers in 2012 was around \$113 billion [49]. In 2013 the economic damage caused by cybercrime worldwide, consumer damage as well as damage done to companies, was roughly estimated between \$375 billion to \$575 billion [11, pp. 6]. Moreover, the coming years the predictions are that malware and cyberattacks will become more sophisticated and probably incur more damage [5, 7].

Not only consumer products are connected to the internet, but also critical infrastructures, such as power and water plants. Disrupting these critical systems can have a huge disruptive effect on daily life. Examples of recent attacks on critical infrastructure are: to gather information about [52]

and disrupt [32] nuclear enrichment plants, gather information from the energy sector [59] or using sophisticated malware to access telecom providers [44, 46].

Although nowadays anti-virus and malware protection applications are commonly used, these types of headlines still appear often. There are several possible explanations for this:

1. Software is not always up to date and the unpatched vulnerabilities are used as an attack vector
2. A zero-day exploit is misused as an attack vector
3. The effectiveness of anti-virus and malware protection applications is depending on signatures and know behavior of malware attacks. So any new attack type should be encountered first, giving the malware exploiters a head start

1.2 Malware detection in practice

Although in 2014 more than 70% of the computers worldwide have real-time security software installed [10, p. 90] still a lot of malware infections take place. The reason for this is the fact that the current security products are one step behind the newest cyberthreats [45, 56]. And this problem will continue as the number of new malware samples seems to be rising each year. According to [13] 143 million new malware samples were discovered in 2014, while in 2013 the number of new malware samples was around 80 million (see Figure 1.1).

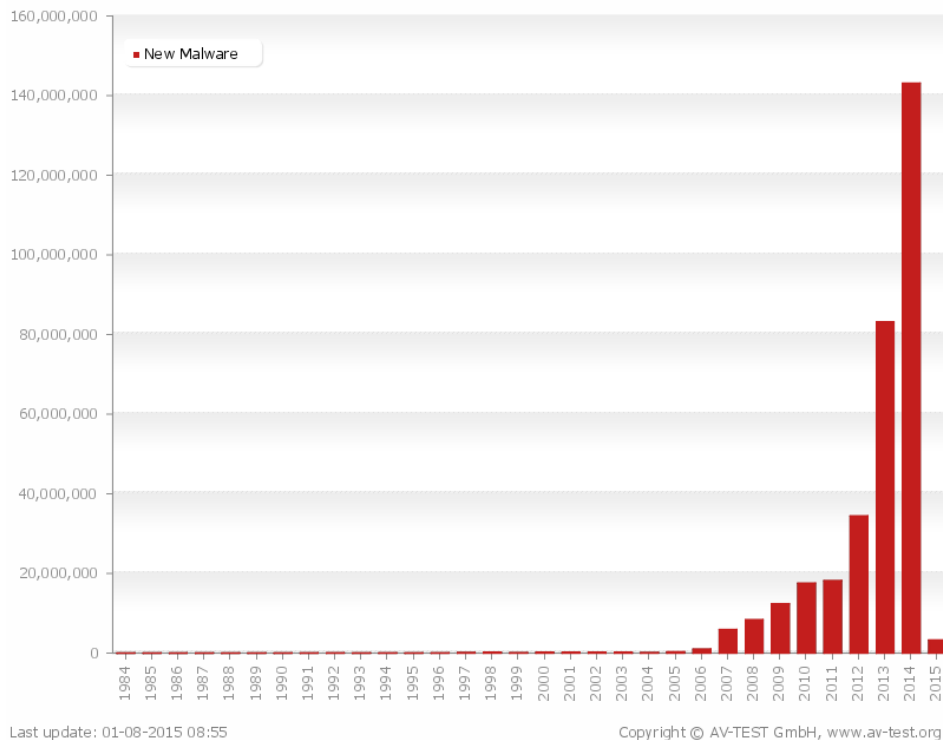


Figure 1.1: New malware samples per year [2]

In the next paragraphs we provide an overview of the current methods used in anti-virus solutions. This will give a better overview how current security solutions work, after which we will show, in section 1.2.2, some examples of how the current used methods can be deceived.

1.2.1 Methods used by anti-virus solutions

This section will explain several methods used in detecting malware. After gaining a better insight in how current security products work and what their weaknesses are, discussed in section 1.2.2, we will take a look at the methods proposed in scientific literature.

The most basic method for malware detection is the scanning of the files which reside on a computers hard disk and compare hashes of the files to known hashes of malware files. These known hashes of malware files are called signatures. Therefor this type of malware detection is called Signature-based Detection [61]. These signatures can be based either on a checksum [60, 72] and byte sequences [56, 60, 72].

A checksum signature is a value which is calculated based on the file. When changes are made to a file the checksum will change as well. In checksum signature based detection the checksums of known malware files are compared to the files scanned on a computer. The downside of this type of signature based scanning is the fact that a small change in the file will change the checksum. To circumvent this problem fuzzy hashing is used. Fuzzy hashing looks for hashes that are almost identical but not completely [36].

To overcome this problem byte sequences signatures are created. In this type of detection the byte sequence performing the malicious behavior is used to create a signature. In doing so changes to a file will not fool the detection method as long as the malicious byte sequence is still present in the file.

Furthermore in [56] the following detection methods for malware detection are explained:

- **Emulation** is now a days used in many major anti-virus solutions. With the emulation method the program's code is broken down into commands. These commands are then launched in a fully contained and controlled virtual environment to observe the programs behavior. This virtual environment emulates a real operating system. The malware will think it runs on the real operating system and will execute its malicious code.
- **Sandbox** is an extension of emulation. In the sandbox method, the executable is run on the operating system. However the interaction the executable has with the operation system is limited by strict rules to safeguard the security of the operating system. For example malware in the sandbox gives the command to write a certain file and execute it. The sandbox will receive this command and will acknowledge the command is successfully performed with executing it.
- **Monitoring** system events is a method which is under rapid development. Statistics are collected on the actions performed by the operation system components.
- **Scanning** for system anomalies is a classified and still emerging method. In this method the system's known status is used as a reference for the future status of a system. If the status of the system has changed too much this might be an indicator of a malware infection.

As can be concluded from the above statement, the state-of-the-art malware detection method at the moment is anomaly detection. This type of detection is still under heavy development.

The next section will explain how it is possible to deceive the currently used detection methods, namely signature checking, emulation and sandboxing.

1.2.2 Deceiving current anti-virus solutions

Current malware has to possibility to circumvent many of these detection mechanisms. In [48] is shown which techniques can be used to bypass a huge amount of the anti-virus software used today. It discusses three types of anti-virus analysis methods: Static signature analysis, static heuristic analysis and dynamic analysis.

Signature-based detection can be deceived by using code obfuscation or encryption techniques [28, 48, 61, 62]. Static signature analysis makes use of blacklisting signatures of known malware and is in use since the first anti-virus solution. The signature of the malware is mostly based on the first executed bytes of malicious code. The downside of this method is the malware must be known to

create a signature, implying that it cannot detect new malware. Bypassing signature based analysis can be done by making changes to the code so the signature will be different when calculated. Oligomorphic, polymorphic and metamorphic viruses use this weakness by automatically changing their code or encrypt parts of their code to change their signatures [60]. In [61] the following obfuscation techniques, which are used to change the signature of the malware to defeat signature-based detection, are described:

- dead-code-insertion: Insert code that does nothing, so the signature will change.
- code transportation: shuffle the code and add jumps in the code, to the appropriate location in the to keep the original flow of the program equal.
- register renaming: the instruction which normally is loaded into a register is replaced with another instruction
- instruction substitution: replace an instruction with another instruction which has the same effect but will create another signature

Static Heuristic analysis is based on checking for code patterns to be known as malicious behavior found in malware. Hereby static heuristic analysis can detect new malware, as long as the malware makes use of already known code patterns. Using these code patterns for detecting malicious behavior can generate false positives. To bypass heuristic analysis the malicious part of the code should be hidden, which can be done by the use of encrypting the code.

Dynamic analysis, according to [48], is used by most anti-virus solutions nowadays. This approach will run an executable in a sandbox environment hereby performing signature-based and heuristic analysis and thus combines several detection methods. In running executables in a sandbox environment possible malicious behavior can become evident without infecting the rest of the system.

Summarizing the information above it can be concluded that today's anti-virus solutions use several methods of detection. These methods are signature-based, heuristic-based, behavioral-based and sandbox detection. However these detection methods can be circumvented. To overcome this problem there is a trend in developing system monitoring methods which uses anomaly detection to detect deviations from the known or expected state. However at the moment such method of anomaly detection is still in development.

In the next section an introduction on malicious behavior detection in scientific literature will be presented. By looking at scientific literature on this subject we will get an insight of what is the current state-of-the-art on malware detection being developed.

1.3 Malware detection in scientific literature

In scientific literature a lot is written about detecting malicious behavior on computers and networks. There are two main types of detection methods present in scientific literature: *misuse detection* and *anomaly detection*. As stated in [26], the difference between *misuse detection* and *anomaly detection* can be defined as: misuse detection detects what is known and anomaly detection detects what differs from what is known as normal. Misuse detection, mostly used in Intrusion Detection Systems (IDS) and anti-virus software, makes use of data of known malicious behavior and categorizes the provided data into the known subsets. New data will be compared to these subsets. Misuse detection has good detection rates for known malicious behavior, but it is depending on this attack behavior information. Therefor misuse detection is not sufficient for detecting new kinds of malicious behavior or 0-day exploits [26, 57, 58].

Misuse detection works by comparing measurable quantities, such as network traffic, to known attack patterns. While misuse detection can quite successfully detect and prevent known attacks, it could already fail to detect slight modifications to existing attacks. And the chance of detecting unseen attacks is minimal [67].

In anomaly detection, sometimes referred to as outlier detection, the data is used for building a model describing normal behavior. This model is used for identifying deviations from this normal behavior [26, 38, 57, 58]. These deviations from normal behavior can be an indicator of malicious behavior.

An overview of anomaly detection methods is presented in [27]. It gives the following definition for anomaly detection: *"Anomaly detection refers to the problem of finding patterns in data that do not conform expected behavior"*. The paper does not only focusing on the techniques used, e.g. classification; clustering or nearest neighbor based, but on the possible applications as well. For example, cyber-intrusion detection, image processing and fraud detection.

To be able to detect new kind of attacks, anomaly detection should provide a better detection rate of unknown malicious behavior compared to misuse detection. As this master thesis focuses on detecting unknown malicious behavior, anomaly detection shall be researched in depth in chapter 2.

1.4 Solution direction

At Fox-IT a new security solution is developed, to overcome the problem needing to know signatures and behavior patterns from malware. Fox-IT is a Dutch based cybersecurity company developing and using innovative solutions to prevent, solve and mitigate cyber threats [1]. This new type security solution, called FoxGuard and is initially intended for enterprise environments. The main difference between FoxGuard and current anti-malware solutions is the fact that FoxGuard does not scan files for malicious characteristics or known signatures, but has the ability to block or allow behavior from applications based on predefined rules. These rules can be used to prevent malicious malware behavior from executing. The main advantage is that it does not need to know what malware is available as it blocks predefined actions.

Because of how FoxGuard is programmed, it has the ability to log very detailed information of the processes running and actions their actions triggered on an endpoint¹ and send this information to a central server. These actions include events on the filesystem and registry or which processes or threads are started by which process. For example FoxGuard is able to log that a process from a browser is reading files on the filesystem. It does not only log if it is reading a file, but also what kind of access it requested, was granted and more. A detailed explanation of the information logged will be given in chapter 4.

As the endpoint application is capable of logging a huge amount of information on the behavior of processes, this could provide new possibilities for developing an anomaly based malicious behavior detection method.

In this masters thesis an explorative research is conducted to develop a proof of concept, of a generic detection system which can detect unknown malicious processes, without prior knowledge on how these malicious processes tend to behave. The problem hereby is, how to find and select these processes from the huge amount of data. In figure 1.2 the problem is graphically presented. We need to find a method to separate the malicious (blue dots) from the benign processes (red dots).

The outcome of the detection system should be list of processes, which perform malicious behavior. Furthermore it should provide graphical tools to help the security officer in analyzing these malicious marked processes.

1.5 TPM relevance

The detection of unknown malicious processes has several implications. The first implication is that a working detection system can be implemented in security solutions, such that these malicious processes can not execute their harmful code. This would prevent damage done by malware. As a result less

¹In this thesis, endpoint refers to an employees work computer, either desktop or laptop, running windows. However an endpoint can be any smartphone, laptop or desktop.

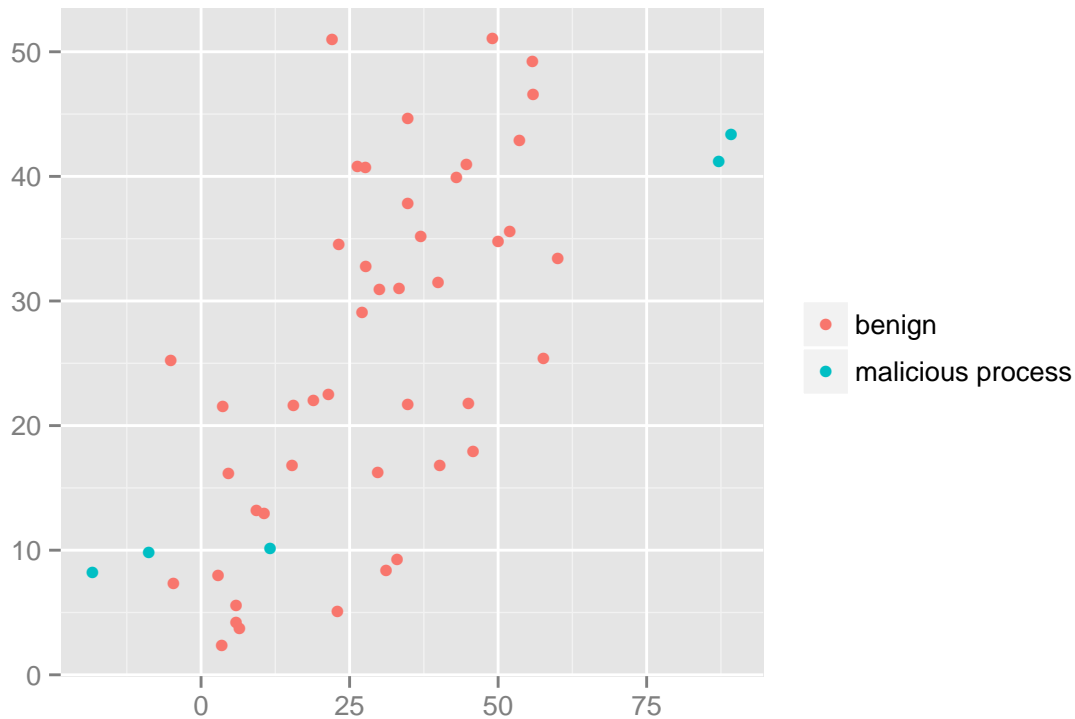


Figure 1.2: Showing the problem of finding the malicious processes

money is spend on restoring backups, cleaning up systems or paid to ransomware owners to decrypt the encrypted files.

Implementing such a detection system in an enterprise environment will provide the security officer with a tool to early detect deviating processes. Together with the graphical tools he is able to analyze these deviating processes. The early detection of possible malware processes, gives the security officer the possibility to prevent damage, or prevent the spreading of the malware on the enterprise network. Hereby preventing more damage done by the malware to computers and indirectly creating monetary loss.

If the detection system is implemented in a system, such as FoxGuard, in such a way that a security officer could request all logged data belonging to the malicious marked process, we have to keep in mind privacy regulations and laws. These laws and regulations are different for each country, but in the Netherlands the type of information stored by FoxGuard are considered personal data, as it can be traced back to a person [23]². This implies that the *Wet Bescherming Persoonsgegevens* should be enforced. This will be discussed in more detail in section 3.2.

An in depth analysis on the stakeholders is presented in 3.

1.6 Knowledge gap: Difference between science and practice

As stated the complexity of malware and attacks has risen in the last years. Although the anti-virus vendors are developing and deploying new methods for detecting new malware, a lot of computers get still infected by malware. One of the reasons for this is the dependence on malware signatures and

²Artikel 1b WBP

malware behavior. Malware which utilizes unknown methods for infecting an endpoint are mostly only identified after the damage is done.

To overcome this problem it is important to find a method which can detect these new and complex malware behavior, based on the actions it performs. Because misuse detection only works based on known malware the focus will be on finding an anomaly detection method. This method should be able to detect malicious behavior based on the hosts process information. This implies that a method is researched, which can model normal behavior of system processes and detect deviations from this normal behavior. These deviations should be an indication of malicious behavior. However using this method of detection brings the problem of false positives into play.

As the detection method is developed to be used in FoxGuard, it is important to take into account what the requirements are for implementing such a detection method in an enterprise solution. Hereby taking into account the requirements the stakeholders of FoxGuard have. And what implications these requirements have on developing such a detection method.

The knowledge gaps can be summarized by the following questions:

- Which detailed process information is usable for identifying malicious behavior?
- Which methods can be used for modeling normal behavior of the system processes?
- Is it possible to detect malicious behavior by deviations from the normal behavior model?
- How can we detect these deviations?

1.7 Research goal and questions

This master thesis is focused on exploring the possibilities of detecting unknown malicious behavior on computers by analyzing detailed process information. As stated in previous sections, anomaly detection is the method of choice. However using anomaly detection introduces the False Positives problem. In exploring the possibilities of using the process information, we should keep in mind how to reduce these false positives.

The problem statement is as follows:

At the moment most malicious behavior is detected based on known behavior. This imposes the problem of not being able to automatically detect unknown malicious behavior and hereby preventing damage done by new malware.

To be able to solve this problem the following research question will be answered in this master thesis:

How can anomaly based detection be used for detecting unknown malicious processes based on the detailed process information gathered on a single endpoint?

The research question can be split into the following sub-questions:

1. What are the current state-of-the-art anomaly detection methods for malicious behavior?
2. Which design requirements should be taken into account when developing an anomaly detection method?
3. Which data can be used for modeling benign process behavior?
4. How well does the constructed method detect malicious behavior?
5. What graphical presentations of the malicious marked processes can aid a security officer?

1.8 Research approach

The first step in answering the main research question is to conduct a literature study on the current state of scientific research concerning detection of unknown malware. Concluding from the literature research the design requirements will be drafted. These design requirements will help in selecting the appropriate features for the data to be collected. The collected data will then be used to explore the possibilities of using the process information in malicious behavior detection. A proof of concept will be developed and will then be tested and evaluated, after which the conclusion can be drawn and recommendations for further research can be given.

1.9 Thesis outline

To answer the main research question the following outline will be used. In the next chapter a scientific literature research will be conducted. This chapter is split into four sections. In the first section anomaly detection will be discussed, after which research on the use of graphs in anomaly detection will be explained. In chapter three, the design requirements will be discussed. First, the design implications from a stakeholder analysis will be stated. Afterwards the design requirements extracted from the literature will be presented. Chapter four will explain the process of collecting and preparing the data to be used in building and testing the malicious behavior detection method.

In chapter five, the prepared data will be used to create process graphs. These process graphs show the connection between processes and provide information on process characteristics. Thereafter the data will be used in creating heatmaps. The heatmaps can be used to spot processes with deviating process activities. Furthermore the heatmaps are used to proof the clustering possibilities and the assumption that process from the same executable tend to behave the same.

In chapter six possible detection algorithms are introduced. In chapter seven, the detection algorithms build in the previous chapter shall be tested on data containing malicious behavior. The setup of testing will be presented after which the outcome of these tests is discussed. Chapter eight will conclude this master thesis and provide recommendations for future future research. IN figure ?? a graphical presentation to the thesis outline is given.

Figure 1.3: A graphical presentation of the thesis layout

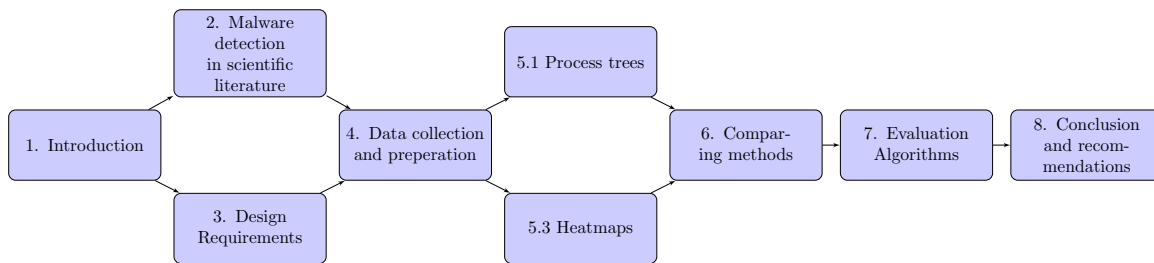


Figure 1.4: Thesis outline

endpoint, log sensor events is collected, is taken through the researched methods, delivers heatmaps and process trees for security officer to easily spot deviations. And is put through compare method and delivers processes marked as malicious, non-malicious and is "delivered" to the security officer.

2

Literature research

In this chapter the literature research conducted is presented. The first section will provide a summary of the scientific literature on anomaly detection. In this part of the research we searched especially for anomaly detection in the IT field. During the search for anomaly detection we found the use of graphs in anomaly detection on network traffic. As the interaction between the processes can be drawn as graphs, the next part of the literature research was based on using graphs in anomaly detection.

Besides the possibility of building process graphs we would have access to very detailed information per process. As we would look at the events triggered per process our graph would be extended with additional process information. As we would need to compare the graphs from a clean and malicious datasets a research was conducted on graph comparison.

The last section is the conclusion which will summarize this chapter.

2.1 Anomaly Detection

As stated in the previous chapter, in order to detect unknown malicious behavior on end-points, anomaly detection is necessary. In this section the current state of research on anomaly detection will be described. Although this master thesis focuses on finding malicious behavior by analyzing information from the processes running on end-points, the conducted literature research will start from a broader perspective. First we will start to look at anomaly detection in IT environments, such as malware and intrusion detection. Thereafter we will narrow down our scope.

In [38] several machine learning techniques for anomaly detection are discussed. Anomaly based detection systems are defined as systems analyzing current activity against a "baseline" of normal behavior and searching for deviations outside of the normal considered behavior. The advantage of anomaly detection is the ability to detect unknown misuse types. However the disadvantage is the number of false alarms that can be triggered by anomaly detection.

The following machine learning techniques for anomaly detection in intrusion detection systems are summarized:

- **Fuzzy Logic** is used in anomaly detection as often the features used can be seen as fuzzy variables. Fuzzy logic is proven to be effective in anomaly detection against port scans and probes. However the resource consumption in using it is high.
- **Bayesian Network** uses probabilistic relationships between the variables of interest. It is generally used for intrusion detection as it has the ability to incorporate prior knowledge.
- **Genetic Algorithms** are inspired on biological principles, such as inheritance, selection, mutation and crossover.

- **Neural Networks** can be used on noisy and incomplete data. This algorithm tries to generalize the data and has the ability to recognize future unseen patterns.

Based on this assumption and the assumption that malware will phone home, a technique is proposed in [24,25], in which they inspect the mouse and keyboard activity from a user and correlate this with the network traffic. The time between an user event and traffic flow is evaluated, whenever a traffic flow appears without an user event happening shortly before it is classified as an anomaly. This concept is also used in [29] and is called BINDER. BINDER consists of four components: User Monitor, Process Monitor, Network Monitor and Extrusion Detector. The first three components collect information on user activity, keyboard and mouse usage, network activity and process information. This collected information is send to the extrusion detector, which should be able to detect malware. The downside of BINDER is that it cannot detect malware when malware is hidden inside another process.

In [41] uses a collection of system calls of processes and convert these input sequences into a bag of system calls hereby removing the sequence of the system calls but preserving the frequency of the system calls. For the unsupervised machine learning k-means with k set to two for clustering in normal behavior and intrusion behavior.

OS-level information is used for clustering the objects of a program together in a cluster n [55]. After which the composed clusters will be compared with predefined behavior templates for malicious behavior. If the cluster matches with these templates a process will be defined as malicious. The problem is the fact that still known malicious behavior is used to detect malicious behavior and does not suffice in detecting new kind of malicious behavior.

In [53] the following information is used for their classification of malware behavior: changes to the file system, changes to the Windows registry, infection of running processes, creation and acquiring of mutexes, network activity and transfer, starting and stopping of Windows services. This data was collected using CWSandbox and after classifying, 88% of the malware was assigned to the correct malware family. As well in [30] they make use of a sandbox to monitor program executions and train different machine learning classifiers for detecting unknown malware instances. However the use of a sandbox might not provide the correct data, as some malware has the capability to detect if it is ran inside of a sandbox and will not run its malicious code [48].

Another method for malicious behavior detection described in scientific literature is making use of the call stack [33]. The call stack stores the information about the active subroutines of a process and provides information to which subroutine it should return control after running. In [33] the return address of the system calls from a process are analyzed to detect possible malicious behavior. The training here for is done by running the processes many times and building a list with the return addresses used by the process.

Unsupervised, supervised and semi-supervised machine learning approaches are used to detect anomalous data in system log files using outlier detection with classification algorithms in [42].

In [65] a method of anomaly detection on network traffic is discussed. The network data used is gathered by using Netflow. This data can be collected from networking equipment and provides the IP flow information going through this equipment [3]. The IP flow contains information such as: source and destination IP addresses, packets and byte counts, timestamps, Type of Service, application ports, input and output interfaces.

From the Netflow data, which will be divided in time windows of 5 seconds, the IP addresses and number bytes are used are converted to metrics used in anomaly detection. The source and destination IP addresses are converted to the Classless Inter-domain Routing (CIDR) format¹. Defining $IP = (prefix, suffixlength)$ where the prefix is the longest common sequence of two IP addresses and the

¹The CIDR format is a standard system for IP address allocation and IP packet routing. The IP address consists of two blocks of bits, the most significant bits and the least significant. The most significant bits identifies the network block. The number of these bits are appended behind the IP address with a slash [4]. For example the CIDR notation of a host IP address is 192.168.1.10/32. The 32 defines that all the 32 bits in the IP address are significant, so only one address is available in this IP block. An CIDR notation of 192.168.1.0/24 defines an IP subnet from all IP addresses in the range of 192.168.1.0 - 192.168.1.255.

suffixlength is the remaining part of the IP addresses. This will define an IP flow as $f_i = (prefix(src)_i, suffixlength(src)_i, prefix(dst)_i, suffixlength(dst)_i, vol_i)$. And every time window will be defined as $W = \{f_1, \dots, f_n\}$.

The constructed time windows will be used in a kernel function which compares two sequential time windows and calculates the similarity of these time windows. A higher value means the more similar the two windows are. On the calculated similarity values the One-Class SVM (OCVSM) algorithm is applied. The reason for using the OCSVM algorithm is the ability to detect unknown anomalies.

The proposed setup is evaluated with several different types of attacks, e.g. Netbios scan and DDOS TCP flood. The results are promising with an average accuracy of 92% and a False Positive rate between 0% and 3.3%.

2.1.1 Disadvantages of anomaly detection

Although anomaly detection seems very promising in scientific literature in practice it is still not that commonly used [57]. One of the main problems with anomaly detection is the introduction of false alarms, or better known as false positives [38]. If the test data does not have enough similar normal instances in the training data, the false positive rate becomes much higher [27].

Another disadvantage is that the strength of an anomaly detection model is depending on the input data. The training data needs to be attack free, otherwise the detection model will learn anomalies behavior as normal behavior [57]. Gathering attack free data, especially in the IT field is quite difficult.

2.2 Graph methods

As stated earlier, the data used contains process properties. Computer processes interact with each other and invoke other system actions. This information might provide useful insights into process behavior. For example one type of process, a file explorer, might perform a lot of file system actions, whilst a internet browser would trigger a significant lower amount of file system actions but will interact more often with other processes. In this section scientific literature will be discussed, in which these process interactions are used for detection of malicious behavior. In the next subsection the usage of graphs and trees in anomaly detection will be discussed.

2.2.1 Graphs

In this part the use of trees and graphs in malicious behavior detection is discussed. The scientific information on this subject is not limited to the use of process information. Literature in which malicious behavior in internet data is detected is presented as well.

An approach of finding unknown malware without any prior knowledge is described in [70]. The information of a user's input activities and the host's outbound network activity is combined, to detect malicious traffic. Using the combined information a traffic-dependency graph is constructed, as shown in Figure 2.1. A traffic-dependency graph consists of several trees of an undefined depth. In which the root of each tree is an user event, user's input to the application through an input device, and the internal and external nodes consists of traffic events. The node with traffic events corresponding directly to the user's input is the subroot. The subroot can trigger other traffic events, for example images loaded on the webpage.

In this model legitimate traffic events are traffic events which are triggered by a legitimate user event. To build the traffic dependency graph, breadth-first search algorithm is used. This algorithm helps in quickly identifying the parent node of a new network request, starting from the most recent sub-root. In the case no dependence is found with the sub-root, the sub-root's childs will be compared.

Using process related information, including relationships among processes, for malicious behavior detection, is done in [66]. The use of this information can help in building a better real time host-level intrusion detection. In the first part of the paper is discussed how the collected process information

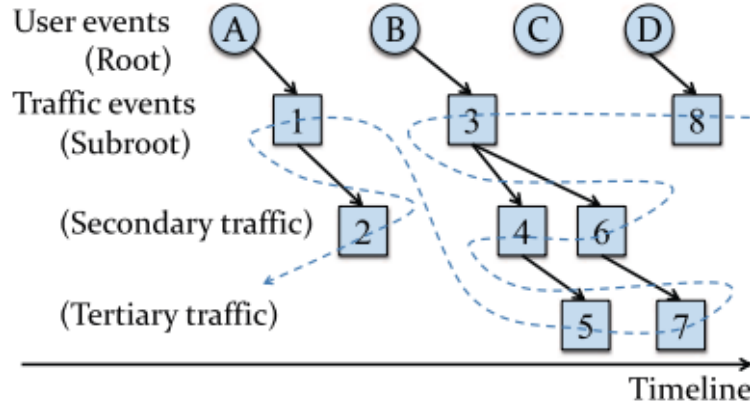


Figure 2.1: An example of a traffic-dependency graph [70]

can be modeled into tree-based structures and how to evaluate the constructed process trees by using tree-based kernels. The second part of the paper describes how to build a labeled graph.

In the constructed process trees the following nodes can be present: *PID*, *Process name (pn)* and *system call (sc)*. The *PID* is a numerical process identifier used in Linux, assigned when the process is started. A *system call* refers to the number assigned by the Linux kernel to a specific system call type. System calls are used to by processes to request a service from the systems kernel. If a process wants access to a file it will use a system call to request the kernel access to the file. The *process name* is derived from the first argument of the system call, which is the name of the program.

Between these three different node types the following types of edges are possible:

- A *PID-to-PID* edge, which presents the creation of one process by another process. The edge indicates the time difference between the two nodes. The monitoring of this is possible by the PPID (parent process identifier) in linux systems, which is an identification of the source process.
- A *PID-to-pn* edge, showing the program started by the process. Here the edge indicates the time difference as well.
- A *PID-to-sc* edge, referring to the frequency the system call is called.

With these nodes and edges a process tree T can be constructed which consists of and ordered pair $T = (V, E)$ where the set of nodes is described by $V = (p_1, \dots, p_n)$. The set of edges is represented by $E = (t_1, \dots, t_j)$. An example of the process tree is shown in Figure 2.2.

The second model discussed in this paper is a labeled graph G which is described as a pair $T = (V, E)$. And where V consists of a set of nodes p which is composed by two parameters: *name_proc*, the process name and *stat_d* the probability distribution of system calls, where *Stat_d* is *stat_d* = (x_1, \dots, x_n) . So V can be presented as $V = (p_1(\text{name_proc}, \text{stat_d}), \dots, p_i(\text{name_proc}, \text{stat_d}))$. See Figure 2.3 for an example of a graph model. By using *stat_d* the distance between the nodes can be calculated. This distance will provide the ability to compare how similar or distinguishable the processes are. When a process with the same name has a higher distance it could be an indication of a malicious processes using a legacy name.

The proposed techniques is evaluated by collecting data through a SSH² honeypot, because SSH is a popular attack vector. The data collected consisted of 75% attack related data and 25% ordinary behavior data. In the training data, the attack data was labeled positive and the normal behavior

²SSH (Secure Shell) is used for a secure connection to a remote machine and execute commands on the remote machine.

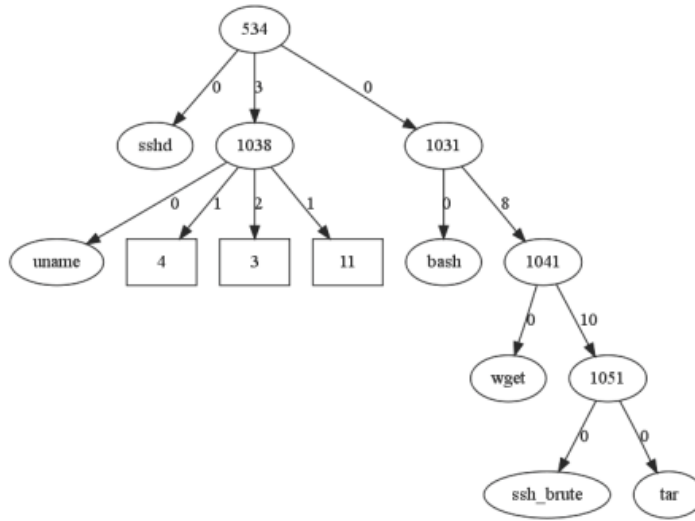


Figure 2.2: An example of a process tree [66]

The figure above shows an example of a constructed process tree. The process with PID 534 starts the program `sshd` and process 1031 within zero seconds. After three seconds it starts the process 1038. The process 1038 invokes the system calls 4, 3 and 11, respectively one, two and one time.

data was labeled negative. To train the supervised SVM with tree kernel, 75% from the data was randomly selected. The remaining 25% is used for testing the prediction accuracy. The experiment was conducted nine times with an accuracy between 0.71 and 0.87.

The use of SSH honeypots for collecting data and creating a process tree, are proposed in [63] as well, although this information is used for a game theory concept to keep the attackers as long as possible busy. In the process tree presented in this paper the *system calls* are replaced by *command line arguments*. In Linux on the command line interface programs can be started with an argument, to invoke a specific function of the program, e.g. the `ls` command is used to show an overview of the files in a directory. By adding the `-l` argument, the ownership and user access rights are presented as well.

As the PID for a program is different the next time the process is started the process tree is converted to a process vector. This vector describes the sequence of the programs executed, e.g. `</bin/bash, /usr/bin/wget, /usr/bin/tar >`. Using these process vectors enables the comparison of different attacks recorded on the honeypot system.

The work is extended by introducing a honeypot hierarchical probabilistic automaton. The probabilistic is based on the argument that attackers will mostly follow a certain pattern of running processes. If an attacker gains access to SSH it will mostly start `bash` or run `uname`, which provides system information such as the running kernel. These patterns of program usage during attacks can have a certain probability. For example after the attackers gained access the chance of starting `bash` or `uname` is respectively 50% and 50%. These probabilities are shown on the edges, see Figure 2.4.

These probabilities are used to create the *honeypot game*. During this game, executions of the attackers will be blocked on a predefined probability. If an action is blocked the attacker might retry the execution or try another kind of attack and hereby providing extra information for analyzing the types of attacks used.

Although the information presented above is not used for real-time detection of malicious behavior, it provides some useful insights how to model and use system process information.

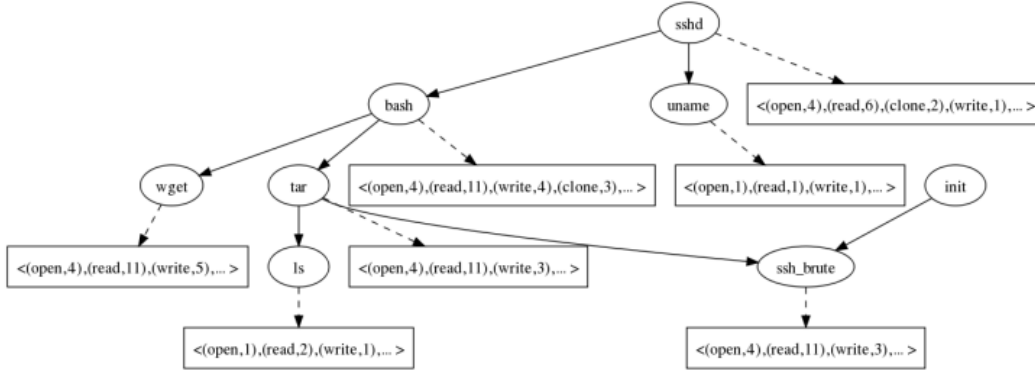


Figure 2.3: An example of a graph model [66]

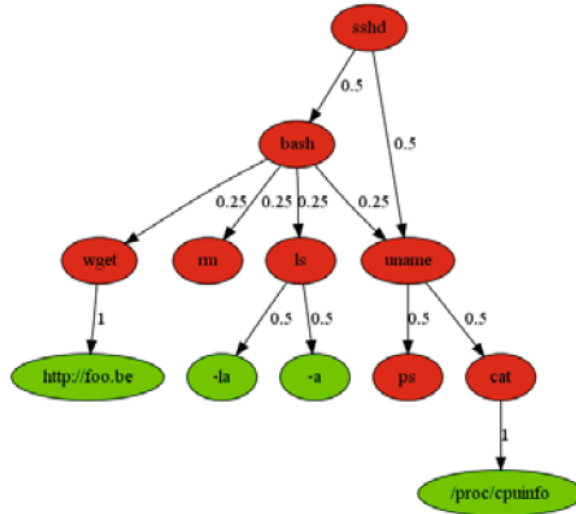


Figure 2.4: An example of a honeypot hierarchical probabilistic automaton [63]

2.2.2 Time based graphs

In [64] the method discussed in [65] in section 2.1 is extended. Based on the extended method, a monitoring framework, called DANAK (Detecting Anomalies in Netflow records by spatial Aggregation and Kernel methods), is proposed.

The first addition is spatial aggregation. Collected data is now aggregated on a spatial and temporal level, which will create a tree-like overview based on aggregated IP prefixes instead of complete IP flows, see figure 2.5. Per time window separate profiles will be created for source and destination data. The CIDR notation is used, grouping IP addresses from the same subnet together, as well subnets on higher levels, only showing nodes with a volume higher than a threshold of α . For each sequential time windows the similarity value (s) is calculated, as discussed before, however the similarity value is now based on the similarity of the source trees from both time windows and the destination trees from both time windows.

The second addition in this paper is the Phase Space Analysis with delayed coordinates. As the time windows are set at 5 seconds and attacks mostly have a longer duration, useful information can be missed. To overcome this problem Phase Space Analysis creates a three-dimensional representation

```

Total = 11008757
0.0.0.0/0 540220 (4.91% / 100.00%)
└─96.0.0.0/3 560312 (5.09% / 27.21%)
    └─101.0.0.0/8 550880 (5.00% / 22.12%)
        └─101.138.64.0/20 754834 (6.86% / 11.99%)
            └─101.138.74.115/32 564682 (5.13% / 5.13%)
                └─101.176.128.0/19 564851 (5.13% / 5.13%)
144.0.0.0/4 771170 (7.01% / 67.88%)
144.115.176.0/20 552664 (5.02% / 5.02%)
145.213.132.0/22 590328 (5.36% / 5.36%)
145.213.144.0/20 712268 (6.47% / 6.47%)
147.186.128.0/18 737222 (6.70% / 17.24%)
    └─147.186.144.0/21 599586 (5.45% / 5.45%)
        └─147.186.160.0/21 561074 (5.10% / 5.10%)
            └─147.186.192.0/18 559543 (5.08% / 26.78%)
                └─147.186.192.0/20 629860 (5.72% / 5.72%)
                    └─147.186.208.0/21 561773 (5.10% / 5.10%)
                        └─147.186.240.0/21 608873 (5.53% / 5.53%)
                            └─147.186.248.0/21 588617 (5.35% / 5.35%)

```

Figure 2.5: Aggregated Tree for a destination profile [64].

of the one-dimensional data s . The equations for a sample n are:

$$x[n] = s[n - 2] - s[n - 3] \quad (2.1)$$

$$y[n] = s[n - 1] - s[n - 2] \quad (2.2)$$

$$z[n] = s[n] - s[n - 1] \quad (2.3)$$

Where $s[n]$ is the similarity value of $Tree_n$ and $Tree_{n-1}$. The effect of the Phase Space Analysis is shown in figures 2.6 and 2.7.

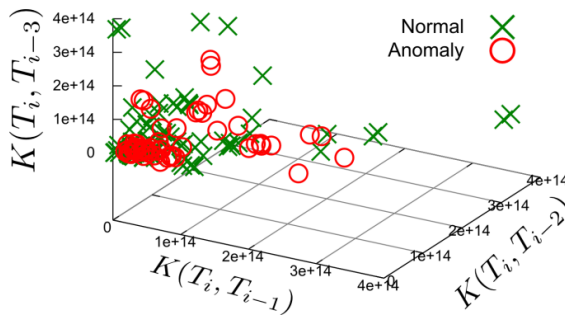
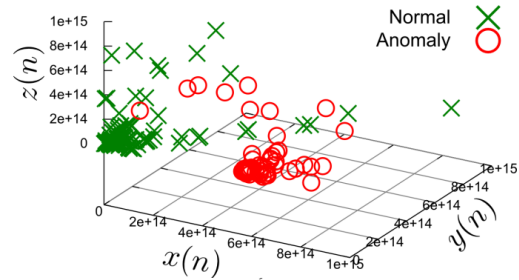
Figure 2.6: A traffic profile T_i , compared to T_{i-1}, T_{i-2} and T_{i-3} showing a DDoS UDP attack [64]

Figure 2.7: Same DDoS UDP Flood attack with Phase Space Embedding Analysis [64]

The classification is performed by using the Classification and Regression Trees (CART) algorithm. This supervised algorithm uses a binary tree building algorithm to find an optimal splitting parameter

for binary splitting. Training the model is done on the attack free data sets. To assess the performance the different data sets with different attack type data injected were used. Performing the classification resulted in a True Positive Rate between 88% and 94%, with a False Positive Rate of 3% and 36%.

In this paper is shown that aggregating huge volumes of data still can be used for anomaly detection. The advantage of aggregating the collected data is the reduced amount of computation needed. This advantage creates the possibility of building a near real-time anomaly detection.

In [35] a prototype of a malicious behavior detection on network traffic is discussed. The prototype is called SAFEM, which is an abbreviation for Scalable Analysis of Flows with Entropic Measures. Here again Netflow data is used.

The SAFEM prototype consists of three components, respectively the S-aggregation module, the Entropy module and the Classifier module. In this paper only the first two modules, S-aggregation module and entropy module, are discussed. The first module will aggregate the Netflow data on the space and time dimensions. The spatial aggregation is done on the IP addresses from the Netflow records. The IP addresses is extracted together with the amount traffic generated, in- and outbound, as a proportion from the total volume of traffic. The collected IP addresses are presented in a hierarchical tree according to the CIDR format. Using the CIDR notation IP addresses from the same subnet are grouped together, as well subnets on higher levels, only showing nodes with a volume higher than a threshold of α . Creating an hierarchical aggregation tree as shown in Figure 2.8.

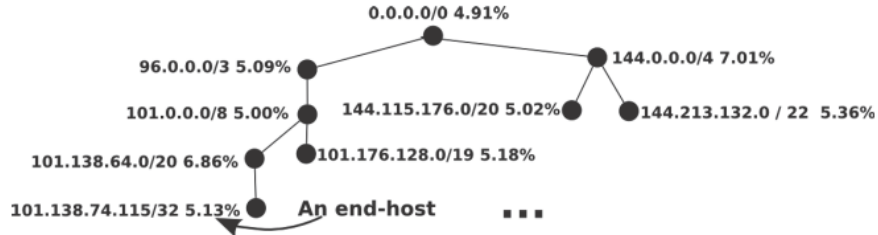


Figure 2.8: A partial view of an aggregated tree with a traffic volume threshold of $\alpha = 5\%$ [35]

The temporal aggregation is performed to create time windows, with a time size specified by β . Within these time windows the data will be aggregated to the spatial dimension described in the previous paragraph. Splitting the time into periods enables the ability to track changes of behavior over sequence of time windows.

After aggregating the data profiles can be constructed. There are two possibilities for constructing these profiles:

1. Observe anomaly free traffic for a limited period of time to construct a profile.
2. Make use of moving profiles, in which the data is compared to recent past time windows.

However the drawback of the first option is, the need for constructing a profile for every potential pattern and keeping these patterns up to date as user activities might change over time. These potential patterns are related to human activity patterns such as night or day time and weekdays or weekend. The use of moving profiles removes the need for creating and updating all the potential patterns and does not need anomaly free data to construct these initial profiles.

Although a tree based representation allows for a compact presentation of a large quantity of data, it lessens the ability to provide useful and relevant information. Therefore the entropy module of the SAFEM prototype is used to calculate the Shannon entropy of the subnets present in the tree. The entropy provides a measure for the quantity of information in the data and the dispersion of the data distribution. If an attack to a host is present in the data the amount of traffic to this host will change and so will the entropy change as well. In the conducted experiments the use of the aggregated trees and entropy proved to be able to detect anomalies in network traffic.

The scientific research on using aggregated trees and entropic metrics in anomaly detection is extended in [34]. This paper builds upon the concept of spatial and temporal aggregated Netflow data as presented in [35]. Using spatial aggregation is useful, as it discards small proportions of traffic that are highly variable and temporal aggregation avoids biased results from short-duration traffic patterns, which are not representative for the overall traffic. Although both aggregations are useful using the wrong level of aggregation could impose a bias due to localized behavior or scattering of local behavior of attacks.

The construction of the aggregated trees is extended by using pre-order traversal to find the position in the tree where an IP address has the highest similarity to other IP addresses, in other words to find the subnet to which the IP belongs in the tree. Furthermore they introduce the usage of Patricia trees in which the tree size has a fixed number of nodes. When all nodes are used and a new node is need, the least recently used (LRU) node is replaced. The use of Patricia trees does not have a significant impact on the quality of aggregation and reduces the post processing analysis. This produces a set of N nodes with $T = \{n_1, \dots, n_N\}$ and $n_i = \langle prefix_i, prefix_length_i, vol_i \rangle$. Here again the aggregation threshold α , the minimum percentage of traffic (vol_i) from total volume of traffic generated by the node, is used to control the granularity of the traffic profiles. And β is used to define the time window size in seconds, creating the following traffic profiles over time: $\{ \langle T_1^{src,byte}, T_1^{dst,byt}, T_1^{src,pkt}, T_1^{dst,pkt} \rangle, \dots, \langle T_M^{src,byte}, T_M^{dst,byt}, T_M^{src,pkt}, T_M^{dst,pkt} \rangle \}$. Where $T_i^{src,byte}$ and $T_i^{dst,byt}$ are for the % of bytes based on respectively the source and destination information and $T_i^{src,pkt}$ and $T_i^{dst,pkt}$ are used for the number of packets.

To be able to perform pair-wise comparison between time windows this paper suggest two different types of entropic metrics to be used, eliminating the variable number of nodes or flows. The two entropic metrics are:

1. IP address distribution entropy
2. Markov Chain entropy

The IP address distribution entropy is again based on the Shannon Entropy, however adopted to take only into account the subspace of IP addresses contained in the tree. Eliminating the use of the whole range of available IP addresses. The Shannon Entropy based on the IP distribution is: H_{ip} . See Figure 2.9 for an example of change in entropy with change in the IP distribution in this case labeled as an anomaly. However the IP address distribution entropy fails to detect IP traffic differences, due to the fact that the underlying IP distribution is discarded by the aggregation.

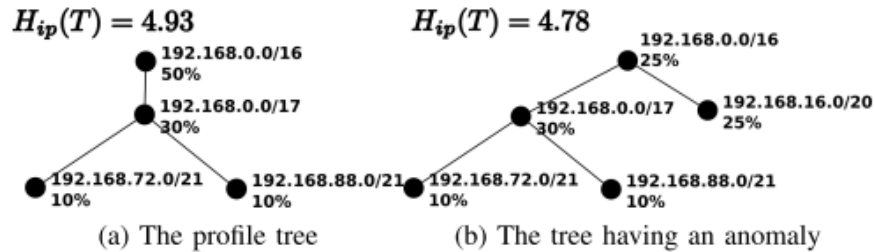


Figure 2.9: The change in the entropy of H_{ip} of a tree without and with anomaly [34]

To solve this the failed detection of IP traffic differences the Markov Chain Entropy is used. The Markov Chain Entropy is the entropy of the tree structure itself and defines the transition probability between to nodes. The Markov Chain entropy is named $H_{tree}(T)$. By using both entropies the data can be described as a set $H = \{h_1, \dots, h_M\}$ in which h_i is a single time window and is defined as an 8-tuple representing the different entropies:

$$h_i = \langle H_{ip}(T_i^{src,byte}), H_{tree}(T_i^{src,byte}), H_{ip}(T_i^{dst,byte}), H_{tree}(T_i^{dst,byte}), H_{ip}(T_i^{src,pkt}), H_{tree}(T_i^{src,pkt}), H_{ip}(T_i^{dst,pkt}), H_{tree}(T_i^{dst,pkt}) \rangle. \quad (2.4)$$

An One Class SVM (OCSVM) was used on the data, as OCSVM performs better on no prior knowledge. The OCSVM was trained on attack free data and tested on several different data sets containing different attacks. The result true positive rate was between the 84.6% and 92.3% and the false positive rate was between the 0.2% and 5%. Although in this experiment they used supervised data, the concept of building trees and using entropies to minimize the amount of data to be fed into a machine learning algorithm proves to be promising.

In [31] the previous explained concept of an aggregated tree is used in the detection of malicious DNS³ names. They assume that malicious domain names are used with a high intensity for a short period of time and the associated IP addresses are highly variable.

The following tuples are collected: $\langle timestamp, FQDN, IP\ address \rangle$ ⁴. The collected data is divided into time windows, specified by η in seconds. The data within each time window is aggregated into a tree-based view of the DNS data. The aggregated tree is limited by 3000 nodes, using Least Recently Used, to keep memory consumption low. Each node contains:

- the domain name (e.g.: tudelft.nl)
- the ip block (e.g.: 145.94.0.0/22)
- percentage of aggregated activity for the current node, defined as volume
- cumulative percentage of activity of the current node and its subtree, defined as accumulative volume

Where only the nodes with a volume higher than α are retained. Nodes with a volume lower than α are aggregated into their parents.

To be able to perform anomaly detection, a steadiness metric is used. This metric characterizes the steadiness of the mapping between IP and DNS spaces over several time windows. To evaluate the steadiness metric, the most similar node to node n_1 from tree T_i will be searched for in tree T_{i-1} . The similarity is calculated by using the IP, DNS and volume features.

The proposed method was tested on a data from which they knew the malicious domains and benign domains and their corresponding IP spaces. Several different datasets were created containing 100%, 10%, 0.1%, 0.01% and 0% malicious domains. Concluding from several tests malicious domains appear to have a steadiness around 0.5, whilst benign domains have a steadiness of around 0.75. Using the 0.5 as an threshold for malicious domains 73% of the malicious domains could be detected with a false positive rate of 1.6%.

Although the percentage of malicious domains detected is not really high, the proposed method shows that using aggregated trees combined with time sequences proves to be useful for anomaly detection.

2.3 Graph comparison

As stated before we would like to explore the possibility of building a process tree combined with the additional process information. In [74] comparing graphs with additional information is discussed. However the additional information connected with the nodes in the graph was pre-labeled information. As our data does not consists of pre-labeled information, the proposed comparing algorithm is not that useful for our purposes.

The tree edit distance is discussed in [68]. The idea behind this method is that every change needed to graph B to get to graph A will have its edit cost. The higher the edit cost, the bigger the difference between both graphs.

³Domain Name System is used for translating domain names, e.g. www.google.com to an IP address.

⁴The fully qualified domain name (FQDN) is the complete domain name specifying a specific host e.g. web-mail.tudelft.nl

Although a lot of literature is written on different methods of matching on different types of graphs, for example, finding the best matching nodes in phylogenetic trees [71] and comparing taxonomy trees [73]. However either the comparison of graphs with additional data is of the pre-labeled kind, or the tree comparing methods are not suitable for the type of tree and additional information we will construct in chapter four.

2.4 Conclusion

In this literature research a lot of interesting concepts of anomaly detection and graph comparison have been discussed. In several papers malicious behavior was detected using process information [53, 66], however they needed labeled data to train their detection method. In [30] process behavior was recorded in a sandbox environment to train machine learning classifiers to detect unknown malware. It can be concluded that process information can be used for detecting malicious behavior.

In [35] a tree based representation is used as it allows to store a large quantity of data into a compact representation. This reduces storage needs and processing time of the data. The use of graphs to detect unknown malicious behavior is also done in [70]. A combination of using graphs and process information to detect malicious behavior can be found in [66]. Our work will be based on these presented methods.

The main conclusion from the literature study is the fact that using process information in detecting malicious behavior is viable. If it is combined with tree based representations large quantities of data can be stored in a compact representation.

3

Design requirements and feature selection

In this chapter the requirements for the design of the malicious behavior detection are collected. To compose these requirements several aspects of the system and environment in which it is operating, should be analyzed. First the design requirements based on a stakeholder analysis are introduced. In the second section the design requirements based on the literature discussed in the previous chapter will be presented. Hereafter the selection of features, taking into account the design requirements, are presented after which the chapter is concluded with a summary of the design requirements.

3.1 Design requirements based on stakeholder analysis

In this section the design requirements from the stakeholders perspective will be identified. As this detection method is developed for incorporating into an endpoint security application to detect real-time unknown malicious behavior in an enterprise environment, the system can be constraint by enterprise environments. In the next section the stakeholder analysis will be conducted.

3.1.1 Stakeholder analysis

As stated in the previous section the system is constraint by enterprise environments. The stakeholders and their implications and requirements are presented in table 3.1.

In the next section the implications of the stakeholders requirements on the design requirements will be explained.

3.1.2 Implications for design requirements

If we take into account the requirements of the end user it means the designed detection model should have a minimal impact if run on the end users computer. When performing the detection model does have a noticeable impact on the end users system, the detection model should be performed on an external system.

To safeguard the privacy concerns of the end user should make sure that the data that could trace back to the end-user is never send to an external server. So either the detection should be performed on the end users computer or when sending data to a server the privacy-sensitive data should be discarded.

Table 3.1: Overview of stakeholders

Stakeholder	Role	Implications	Requirements
<i>End user</i>	Employee of an enterprise using the computer with FoxGuard installed on it.	The end user's data is collected and send to the server. Data could contain information such as file paths or file names.	<ul style="list-style-type: none"> • Privacy safeguarded • Low impact on system usability
<i>IT Manager</i>	Responsible for managing the IT systems and applications	New system, sends information, needs storage	Does not want to buy expensive servers for performing the real-time detection
<i>System Admin</i>	Responsible for installing and maintaining applications	Install and maintain endpoint applications and server	Low impact of the systems of the end users and low server and network requirements
<i>CISO</i>	Is responsible for the IT security of a company. Prevent, detect and solve security incidents.	Should provide the CISO with useful information on possible malware infections	<ul style="list-style-type: none"> • Fast and reliable detection of malicious behavior • Comply with privacy laws and regulations
<i>Security Officer</i>	Monitor the FoxGuard system to see if any	Will see the possible detections of malicious behavior	<ul style="list-style-type: none"> • Clear information on detected malicious behavior. • Should be able to react to a possible infection. • Does not want to be bothered with false alarms

The IT manager does not want to spend a lot of money to be able to run real-time detection. This implicates that both the processing power as well as the required storage should be minimized as much as possible.

For the system admin using real-time detection should not have a noticeable impact the current infrastructure. This implies that the data send to a server should be held at a minimum to unburden the network. Furthermore the storage needs and processing power needed to perform the real-time detection should be kept at a minimum.

The CISO wants a secure IT environment as possible. To be able to deliver this the detection model should detect all malicious processes.

The security officer wants to get clear information on detected malicious behavior but does not want to be bothered with false positives. This implies that when malicious behavior is detected we should be able to pin point which process it is that is malicious and on which host.

Further more the number of false positives must be very low, even with a 0.001% false positive rate the security officer will receive a high number of false positives a day in an enterprise environment. For example 500 end users which have about 800 processes running a day means still 400 false positives are shown.

3.2 Design requirements imposed by privacy law and regulation

As mentioned in section 1.5 storing and using the information that can be logged by FoxGuard has to confirm to privacy law and regulations. The laws and regulations which need to be enforced do depend on the country in which the system is used. In this section the applicable Dutch laws and regulations will be discussed.

According to *Artikel 8 WBP* [?] personal data can only be processed in the following cases:

- (a) The concerned has agreed to the processing of his data
- (b) The processing of the data is required for the execution of an agreement in which the concerned is involved
- (c) Processing the data is required to comply with a lawfully obligation
- (d) Processing the data is a necessity to protect a vital interest of the concerned
- (e) Processing the is needed to fulfill a public task
- (f) Processing the data is required to fulfill a justified interest

According to item f the processing of personal data is allowed when it fulfills a justified interest. In the case of malware detection and preventing malware infections, this can be justified. If the detection system makes use of stored personal data it has to comply with the following rules as well:

1. The personal data must be stored in compliance with the law and should be processed carefully¹
2. Personal data can only be used for justified cases which should be described beforehand ²
3. The person from whom the personal data is processed, should be notified who is processing the data and why ³
4. The processing of the data should be done secure ⁴

This implies for our design requirements, when data which can be identified as personal data is used the right security measures should be taken to safeguard the data and the concerned should be notified about why this data is collected and processed.

¹Artikel 6 WBP

²Artikel 9 WBP

³Artikel 33 WB

⁴Artikel 13 WBP

3.3 Design requirements based on literature

A tree based representation allows to store large quantities of data, which reduces the storage needs. Furthermore aggregating data will reduce this even more. But not only the storage needs will be reduced [35]. If less data is used to perform the detection model, less processing power is needed, so the calculations can be done faster. This enables the option to create a real-time detection model.

3.4 Design requirements based on the data

The amount of data generated by one host on a normal working day is huge. In eight hours about 20 million events are created. When using R to perform computations this takes up a lot of memory. Therefore selecting only the needed data will not only reduce the memory usage but will decrease processing time and storage needs.

3.5 Conclusion

One of the main themes present in all design requirement analysis is the need to reduce the amount of data. Reducing the data needed to perform the malicious behavior detection, will reduce the amount of memory, storage and processing power needed. Furthermore if the data is sent over the network sending less data will reduce the chance of congestion on the network. This will reduce the burden on the IT manager in buying new equipment to perform real-time detection.

To be able to reduce the data needed we will construct process trees, as is suggested in [35]. Furthermore the amount of data every event type sends, is not all useful. When all this data would be used in building a detection model the chances of over fitting are becoming larger and hereby losing the generic character of the detection method.

4

Data collection and preparation

In this chapter the process of collecting and preparing the data for further analysis is explained. The first section will explain how the data is collected and what type of information is included in the collected data. Section two will explain which data will be used and section three will explain what actions are performed to get the desired data.

4.1 The data

As stated earlier the data collected by the FoxGuard endpoint is very detailed and low level system information. The ability to record this low level system information is called sensor log. The type of information collected by the sensor log is divided into eight different event types, namely: registry, file system, process create, process exit, thread create, thread exit, module load and object callbacks. Besides the time and sensor event type field, each sensor event type has different required and optional data fields defined. In the following paragraphs each event type will be described shortly. For a more detailed overview of what is included see appendix A.

As the process id and thread assigned by Windows can be recycled after a process or thread is closed, FoxGuard will assign an unique id to each process. This unique process id or thread id will not be recycled during a complete cycles. After rebooting the system, the unique ids will be assigned again, starting from 1. Furthermore is each sensor event contains a timestamp.

4.1.1 Filesystem event

Filesystem events provide detailed information on the actions performed on the filesystem. Such as, what kind of filesystem action, the process triggering the event and the size of the performed action. After the registry event, the filesystem events produces the most sensor events, around 5 to 10% of the data.

4.1.2 Registry event

A registry event is triggered when a process tries to perform an action on the Windows registry. The Windows registry is mostly used for storing configuration settings of Windows and applications. Examples of registry events are, when a process tries to read a certain registry entry the desired access

mask, the granted access mask, the type of operation and the result of the operation is logged. Besides the information on the registry action performed information is included on which source process and source thread did trigger the event. About 80 to 90% of the logged events are registry events.

4.1.3 Process create event

In the process create event, information can be found on the starting of a new process. This information contains the source process starting the new process, a tokenized string of the location of the executable executed, called the executable path hash. If a process has the same executable path hash, it means the same executable is started. This information can be used during the analysis to see which processes are originating from the same executable. Furthermore optional command line parameters are logged as well.

4.1.4 Process exit event

Besides the usual information such as the timestamp, the process exit event only provides the process id and unique process id of the process closed.

4.1.5 Thread create event

A thread create event provides the information of the source process and thread starting a thread, into which process the thread is started and the location of memory in which the thread is loaded.

4.1.6 Thread exit event

The same as the process exit event, it provides the thread id and unique thread id of the thread closing.

4.1.7 Module load event

When a DLL is load a module load event is triggered. Module load events contains the size of the loaded module, the start address of the memory into which the module is load as well as the path location where the module can be found.

4.1.8 Object callback events

An object callback is an executable code which is passed into another program to be executed. The object callback event contains information on the source process and thread id as well, as the unique source process and thread id, executing this event type. Furthermore it contains the process id and unique process id of the target which must execute the given code.

4.2 Collecting the data

Before any research can be done on finding an anomaly detection method for malicious processes, data needs to be collected. The next two subsections will be described how the clean datasets and infected datasets were collected. Furthermore possible implications of the manner of data collecting on the collected data are explained.

4.2.1 Clean datasets

To collect the clean datasets FoxGuard was installed on the laptop, running Windows 8.1, of an employee of a small and medium-sized business (SMB). By turning on the capability of logging the sensor events all process information is gathered and send to the server. The clean data collected consists out of four different datasets. Two of the datasets are gathered during a workday, the other

two datasets are collected during some evening and weekend usage of the laptop and are of a shorter duration.

As the laptop was freshly installed the week before collecting the data and the needed anti-virus and anti-malware software was installed and running on the laptop, the laptop is considered malware free.

4.2.2 Malware datasets

Unfortunately the collection of the malware data could not be performed on the same machine as the clean data. The reason for this, the possible access to personal and company information, such as documents or passwords. To be able to collect sensor log data with malware a virtual machine running Windows 8 was set-up. On this virtual machine Office 2013, Firefox and Chrome was installed in an attempt to simulate a comparable environment as in which the clean data was collected. During the logging of the data, Chrome and/or Firefox were used for browsing and Word and Excel were used to perform office tasks in simulating a working employee.

Most malware will try to phone home after being installed. This functionality is used to send over collected data, receive commands to perform or receive and install new malware. To ensure the installed malware could call home, the VM was connected to the internet without a firewall.

For every type of malware collected, the clean virtual machine was first used normally. This usage means using internet browsers, doing some office work and browsing the computer. After a certain period, the malware was executed from a folder containing the executable.

Infecting the VM this way is comparable to receiving an e-mail with a malicious attachment which needs to be extracted and executed. However opening a zip file from outlook and executing the malicious file, might appear on a different location in the process tree compared to the above described manner.

Another popular method of infecting a computer is done by exploiting vulnerabilities in the installed software, e.g. Adobe Flash, and downloading and executing malicious software, such as done with malicious advertisements [9, 20, 22].

As this type of attack is difficult to set-up it is not taken into account during this thesis research. For further research we would advice to look into the different methods of how the malware is installed and include these in testing and evaluating.

After the execution of the malware some more simulated work was done, before restarting the computer. When rebooted the computer was again used for some time to perform office tasks and browse the internet. Subsequently the computer was rebooted again and cleaned.

Due to the fact the machine was infected with malware, complete and long term usage of the virtual Windows 8.1 machine was not feasible, as no real work could be done on the machine, as well long term running of the malware on the company network was not desirable. Therefore the running time of the collected datasets with malware infections are shorter than the datasets collected on the SMB's employee's computer. More information on the duration of both the clean and infected datasets can be found in section 4.3.

4.2.3 Malware used

For the creation of the malware data samples three different types of malware were used: a banking malware, a Remote Access Trojan (rat) and a Zeus malware. These malware types are, together with cryptolocker/ransomware, the most common malware infections. Although we would like to have tested the cryptolocker malware, as we suspect that it generates a lot of filesystem activity, however due to the fact the malware encrypts files on the computer, we were not sure if we could extract the log files. Together with the time constraints we did not test it.

The banking malware and Remote Access Trojan were both discovered in the week before running the malware. In this subsection the three different malware samples used will be discussed.

4.2.3.1 Bank malware: Dridex

The first malware was a banking malware called Dridex, however it is capable of more than only stealing bank account information. It is capable of uploading, downloading and executing files, communicating with peer to peer networks to receive new commands or be added to a botnet [54]. As stated in the previous section the running time of the malware was limited. This could imply that some of the possible actions of the malware were not performed as it might not have received any commands from the central server.

4.2.3.2 Remote Access Trojan

The second malware is a Remote Access Trojan (RAT). This malware installs a backdoor on the computer giving the attacker administrative control. This can be used to control the computer, distribute the RAT to other victims or install other malware. Here again the elapsed time while running the malware was limited, so it might not have performed any additional operations to installing itself.

4.2.3.3 Zeus Trojan

The aim of a Zeus Trojan is to steal personal data from the computer, such as log in data from online accounts. The collected information is sent to remote servers. The Zeus malware makes use of stealth techniques to impede detection and removal. For this malware sample the same limitation of time applies and therefore might not have shown any additional activity besides installing itself.

4.3 Collected datasets

In the first subsection the collected clean data is presented. Hereby giving the datasets name, which will be used for further reference in this thesis, a short description of the dataset, the number of minutes the dataset encloses and the number of events present in the dataset per event type.

4.3.1 Collected clean datasets

Table 4.1 gives an overview of the four clean datasets collected. The overview includes information such as the time between start and shutdown of the computer and the number of events for each event type.

4.3.2 Collected malware datasets

In table 4.2 an overview is given of the malware datasets. In the description is some additional information about the time the malicious file was opened. Hereby providing some reference points for finding the possible malicious processes by hand. As mentioned in 4.2.2, the running times are short compared to the clean datasets. For future recommendations we would advise to try to create larger datasets containing malware.

4.4 Data selection

The previous section provided an overview of all the collected datasets. As can be seen in the tables 4.1 and 4.2 the number of events, especially for the working day datasets, are rather high. For a full working day around 30 million events are logged. These amounts of data are a heavy strain on the memory usage of the computer.

Concluding from the literature study and the design requirements we do not need all available data. First of all the use of a lot of data creates a high chance of introducing overfitting. Furthermore for

Table 4.1: Clean datasets

Dataset	Description	Running time	Event type	Event numbers
win8_1604	The data logged from a full working day.	7h 49 min	file system	3545608
			registry	26556541
			process create	924
			process exit	884
			thread create	61912
			thread exit	61377
			module load	39551
			ob	492528
win8_1604_avond	Data from the usage of the computer during off-work hours	2h 6 min	file system	542707
			registry	11105607
			process create	461
			process exit	422
			thread create	18936
			thread exit	18406
			module load	17080
			ob	141110
win8_1704	Another full day of work data logged	7h 52 min	file system	2128619
			registry	25644005
			process create	880
			process exit	839
			thread create	58928
			thread exit	58331
			module load	35971
			ob	447948
win8_1804	The computer used during the weekend	48 min	file system	495925
			registry	7097579
			process create	331
			process exit	257
			thread create	8861
			thread exit	7826
			module load	12150
			ob	83304

Table 4.2: Malware datasets

Dataset	Description	Running time	Event type	Event numbers
malware_bank	The VM was started clean. At 16:44 the banking malware Didrex was started	32 min	file system registry process create process exit thread create thread exit module load ob	183405 2067666 160 139 3786 3494 6794 30791
malware_rat_session1	Started at 13:37. At 14:03 the machine was infected by running the rat malware.	44 min	file system registry process create process exit thread create thread exit module load ob	173334 4683261 223 202 4827 4524 8959 97871
malware_rat_session2	After reboot the machine is still infected	19 min	file system registry process create process exit thread create thread exit module load ob	211686 2153338 212 190 3083 2779 7846 52508
malware_zeus_session1	The machine was cleanly started. Around 16:50/16:55 the Zeus malware was started.	17 min	file system registry process create process exit thread create thread exit module load ob	100213 1842066 155 135 2907 2613 6579 44059
malware_zeus_session2	Run after reboot with the Zeus malware	19 min	file system registry process create process exit thread create thread exit module load ob	153253 1815547 190 170 3498 3207 9230 97084

constructing process trees and describing the processes by the number of events per second a process triggers we do not need all the data as well.

To construct the process trees we will need the unique source process id and the unique process id present in the process create sensor event type. Each of these rows represents a parent node with its child node. Furthermore the executable path token will be selected. With the help of the executable path token it is possible to find the processes which are started by the same executable.

For the second data selection we do not need all the additional information from the sensor event types, such as filesystem, registry, module load, ob process create and thread create to calculate the number of events per second trigger by the process. The next chapter will explain how the wanted data is extracted, created and prepared.

4.5 Data preparation

To be able to use the data, it first needs to be altered and prepared. This section will explain the steps taken to get from the raw data to the desired datasets containing all the information needed.

The raw data is extracted from the server in Protocol Buffer format. Protocol Buffer is a mechanism for serializing structured data into a small footprint [37]. First the data is convert from Protocol Buffer format to JSON format. After which the data can be loaded into R. As not all data is needed, while importing the not needed variables will be discarded. The variables that are imported are:

sensor_event_type, time, source_process_id.id, source_process_id.unique_id, process_id.id, process_id.unique_id, process_create_event.command_line.data, process_create_event.command_line.token, process_create_event.process_exe_path.token

During the import the variable time is converted to a human readable time annotation in *time.date*, as the time variable is in filetime, which is the number of 100 nanoseconds since first of january 1601.

After importing the data it can be used to construct for every dataset a dataframe with the needed information. In every data set there are two process_id and process_id unique_id which have an unknown unique ID. These id's are assigned 9999, the kernel process and 9998, the user land process. Hereafter the source_process_id.id and process_id.id can be discarded as these values are reusable by Windows and will only cause confusion.

Next for every process ran in the dataset we will count the number of events, filesystem; registry; process create; thread create; module load and object callback (OB), they have triggered and calculate the number of events per second. The reason here for is the fact that not every process is running the same amount of time. This calculation is done on the time provided by the process create events and the process exit events. If no process create event can be found the start time of the dataset will be used. In the case no process exit can be found the end time of the dataset will be used.

The events per seconds variables will be normalized between 0 and 10, see equation 4.1. In which x is the value to be normalized, A and B are the minimum respectively maximum value of the variable to be normalized and a and b provide the range for the normalization.

$$\frac{(x - A) * (b - a)}{(B - A)} \quad (4.1)$$

As the number of filesystem and registry events are a lot higher during the same period, scaling will provide values which are comparable with each other. To normalize all the datasets with same min and max values, all datasets will be combined first, normalized and split again. Now for every dataset the normalized values for the events per second are the same.

The used method for normalizing the data, creates implications for usage in a malicious process detection system, as for every new data collected all datasets need to be combined to normalize the data. To avert this, the maximum and minimum non-normalized values for every event type variable of the already collected data can be stored. The newly collected data can then be checked to see if new minimum or maximum values are present. If not the case, the data can be normalized by using

the known values for A and B. When a new maximum or minimum is present in the newly collected data, all data, used for the detection, should be normalized again with the new A and B values.

In this research we chosen to use to normalize the data between 0 and 10. However using other methods for normalization should be tested in future research, such as for example Z-scores.

In the dataframes we have the six event type variables which provide information on the behavior of the process. Further more the unique process ids and unique ids the source process, the executable path token, commandline data and command line token are included.

As will be discussed in chapter 5 the processes with similar process activities, such as filesystem actions per second, tend to group together. As k-means is an often used clustering method in anomaly detection, to group similar entities together. As we assume that malicious processes perform behavior deviates from normal process behavior we expect that the processes will have a larger distance to these cluster centers.

First all the clean datasets are combined into one dataset. To find the appropriate amount of clusters we analyze the within group sum of squares for the number of clusters from two to fourteen. As there is a elbow at the cluster size of eight, see figure 4.1, this number will be used for clustering. K-means "Hartigan-Wong" algorithm is used to cluster the clean dataset with a cluster size of eight. The clustering is done on the six event variables by minimizing the within-cluster sum of squares, see equation 4.2 [39]. In which $i = 1, 2, \dots, n$, with n defining the number of events, so the number of processes in the dataset. j is defined as $j = 1, 2, \dots, p$, in which p is the number of variables, so in our case six. $\bar{x}(k, j)$ is the mean of the variable j of all elements in a cluster k .

$$Sum(k) = \sum_{i=0}^n \sum_{j=0}^p (x(i, j) - \bar{x}(k, j))^2 \quad (4.2)$$

The cluster centers found are now used to cluster the malware datasets. Due to not every cluster being present in the malware datasets the k-means function in R ¹ could not be used. A own written function was used to cluster each processes to the closest cluster center using the euclidean distance measure.

The data is now ready to be used for further analyses and to test the possible detection methods. In the next section we will identify the malicious processes for each malware dataset.

4.6 Malware processes

To be able to evaluate the detection methods, the malicious processes need to be identified. In this section we try to identify for every malware dataset the malicious processes. These will be used in chapter 7 during the evaluation.

Identifying the malicious processes is done based on the known starting time of the malware executable and inspecting the command line options and the processes started by the malware process.

4.6.1 Banking malware

After analyzing the process create events from the banking malware twelve malicious processes where found. The starting of the malicious executable has the process id 111 after which the processes 112, 116, 117, 118, 120, 121, 122, 123, 124, 125, 126 are started.

By taking the path tokens of the processes marked as malicious and check if these path tokens are found in any of the other datasets, we found that the processes 117 and 121 can be found all the collected datasets. This means the process itself is not a malicious executable however it is used to perform actions with malicious intent. A further investigation into these processes will be done in section 5.4.

¹The statistical software used for working with the collected data in this Master Thesis. R is a language for statistical computing and graphics [17]

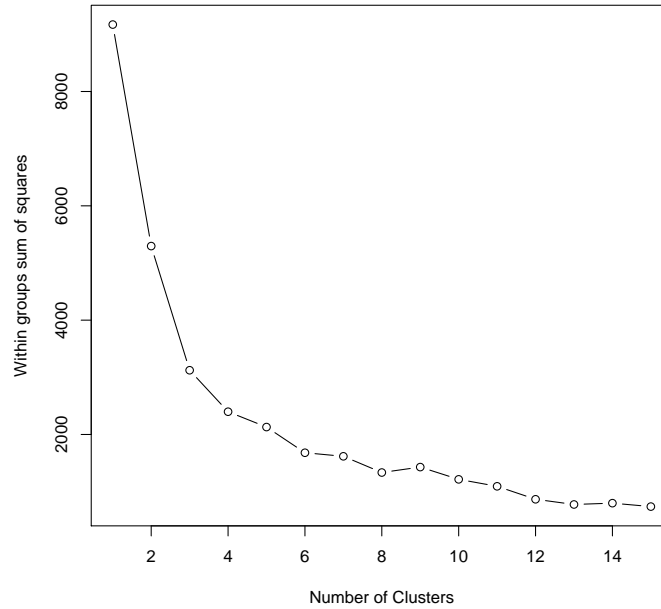


Figure 4.1: K-means plot

4.6.2 Rat malware session 1

In the rat malware session 1 there are twenty processes found which are marked as malicious. Their corresponding process ids are: 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185 and 186. In table 4.4 we can see that process 167 starts process 168, which spawns the other eighteen processes.

4.6.3 Rat malware session 2

If we take the path tokens from session 1 and look for these path tokens in session two we find eighteen processes which match. These processes ids are 64, 66, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84 and 85. However if we look at which process has started these processes we find that process 62 spawns all these processes. Process 62 is started by process 60 which has the same path token. This executable is written to disk before reboot and set as start process after reboot. Here again we have marked twenty processes as malicious. See table 4.5 for more in depth information.

4.6.4 Zeus malware session 1

In the first session of the Zeus malware, the session where the malware executable is executed, the process with id 114 is the starting of the malware. Hereafter four other processes are started, namely 116, 122, 125 and 126, making a total of five processes marked as malicious. See table 4.6 for detailed information on the processes.

Here again there is a process present with a path token which can be found in all other datasets. As stated in the previous section this will be discussed in section 5.4.

Table 4.3: Malicious processes banking malware

source process unique id	time date	time run	process unique id	command line token	process exe path token	command line.data
30	2015-06-11 16:44:49	0.156	111		6, 774, 935, 858, 042, 816, 512	
111	2015-06-11 16:44:49	0.031	112	1, 668, 767, 127, 550, 293, 248	2, 853, 641, 700, 518, 833, 664	/Q "C:\USERS\LAB2\APPDATA\LOCALLOW\NGPBJPQG.SDB"
111	2015-06-11 16:44:50	0.453	116	1, 668, 767, 127, 550, 293, 248	2, 853, 641, 700, 518, 833, 664	
116	2015-06-11 16:44:50	0.500	117	-9, 086, 252, 108, 385, 885, 184	-5, 932, 391, 742, 018, 334, 720	
111	2015-06-11 16:44:50	0.547	118		3, 371, 955, 335, 670, 766, 592	
111	2015-06-11 16:44:50	0.188	120		3, 371, 955, 335, 670, 766, 592	
120	2015-06-11 16:44:50	0.203	121	-9, 086, 252, 108, 385, 885, 184	-5, 932, 391, 742, 018, 334, 720	
120	2015-06-11 16:44:50	0.297	122	4, 619, 302, 901, 630, 733, 312	5, 992, 314, 292, 155, 399, 168	/C C:\USERS\LAB2\APPDATA\LOCALLOW\NGPBJPQG.BAT
122	2015-06-11 16:44:50	0.366	123	-847, 390, 535, 022, 933, 504	2, 230, 221, 228, 479, 690, 752	C:\USERS\LAB2\DOWNLO\1\BANKING\DRIDEX-1.EXE 3
122	2015-06-11 16:44:50	0.377	124	-6, 720, 181, 090, 770, 503, 680	2, 853, 641, 700, 518, 833, 664	/Q /U "C:\USERS\LAB2\APPDATA\LOCALLOW\NGPBJPQG.SDB"
122	2015-06-11 16:44:51	0.703	125	841, 275, 868, 143, 678, 208	1, 113, 762, 613, 334, 646, 528	DELETE "HKLM\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\APPCOMPATFLAGS\CUSTOM\JSCSICL1EXE" /F
122	2015-06-11 16:44:51	0.844	126	-1, 979, 144, 317, 536, 490, 752	1, 113, 762, 613, 334, 646, 528	DELETE "HKLM\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\APPCOMPATFLAGS\INSTALLEDSD\{F48A0C57-7C48-461C-9957-AB25DDC986E}" /F

Table 4.4: Malicious processes rat session 1

source process unique id	time date	time run	process unique id	command line token	process exe path token	command line.data
38	2015-06-11 14:03:21	0.110	167		-477, 687, 000, 000, 000, 000	
167	2015-06-11 14:03:22	0.227	168		-477, 687, 000, 000, 000, 000	
168	2015-06-11 14:03:22	0.436	169		1, 941, 070, 787, 904, 570, 112	
168	2015-06-11 14:03:22	0.580	170		-8, 392, 900, 000, 000, 000, 000	
168	2015-06-11 14:03:23	1.693	171		-8, 392, 900, 000, 000, 000, 000	
168	2015-06-11 14:03:23	1.795	172		-8, 392, 900, 000, 000, 000, 000	
168	2015-06-11 14:03:24	2.398	173		-8, 392, 900, 000, 000, 000, 000	
168	2015-06-11 14:03:24	2.500	174		-8, 392, 900, 000, 000, 000, 000	
168	2015-06-11 14:03:24	3.103	175		-8, 392, 900, 000, 000, 000, 000	
168	2015-06-11 14:03:24	3.205	176		-8, 392, 900, 000, 000, 000, 000	
168	2015-06-11 14:03:25	3.808	177		-8, 392, 900, 000, 000, 000, 000	
168	2015-06-11 14:03:25	3.910	178		-8, 392, 900, 000, 000, 000, 000	
168	2015-06-11 14:03:26	4.513	179		-8, 392, 900, 000, 000, 000, 000	
168	2015-06-11 14:03:26	4.615	180		-8, 392, 900, 000, 000, 000, 000	
168	2015-06-11 14:03:27	5.218	181		-8, 392, 900, 000, 000, 000, 000	
168	2015-06-11 14:03:27	5.320	182		-8, 392, 900, 000, 000, 000, 000	
168	2015-06-11 14:03:27	5.923	183		-8, 392, 900, 000, 000, 000, 000	
168	2015-06-11 14:03:27	6.025	184		-8, 392, 900, 000, 000, 000, 000	
168	2015-06-11 14:03:28	6.629	185		-8, 392, 900, 000, 000, 000, 000	
168	2015-06-11 14:03:28	6.732	186		-8, 392, 900, 000, 000, 000, 000	

Table 4.5: Malicious processes rat malware session 2

source process unique id	time date	time run	process unique id	command line token	process exe path token	command line.data
39	2015-06-11 14:23:04	1.844	60		-775,034,000,000,000,000	
60	2015-06-11 14:23:06	3.891	62		-775,034,000,000,000,000	
62	2015-06-11 14:23:07	0.438	64		1,941,070,787,904,570,112	
62	2015-06-11 14:23:07	0.625	66		-8,392,900,000,000,000,000	
62	2015-06-11 14:23:08	0.203	70		-8,392,900,000,000,000,000	
62	2015-06-11 14:23:08	0.312	71		-8,392,900,000,000,000,000	
62	2015-06-11 14:23:09	0.938	72		-8,392,900,000,000,000,000	
62	2015-06-11 14:23:09	1.047	73		-8,392,900,000,000,000,000	
62	2015-06-11 14:23:10	1.672	74		-8,392,900,000,000,000,000	
62	2015-06-11 14:23:10	1.781	75		-8,392,900,000,000,000,000	
62	2015-06-11 14:23:11	2.406	76		-8,392,900,000,000,000,000	
62	2015-06-11 14:23:11	2.516	77		-8,392,900,000,000,000,000	
62	2015-06-11 14:23:11	3.141	78		-8,392,900,000,000,000,000	
62	2015-06-11 14:23:11	3.250	79		-8,392,900,000,000,000,000	
62	2015-06-11 14:23:12	3.875	80		-8,392,900,000,000,000,000	
62	2015-06-11 14:23:12	3.984	81		-8,392,900,000,000,000,000	
62	2015-06-11 14:23:13	4.609	82		-8,392,900,000,000,000,000	
62	2015-06-11 14:23:13	4.719	83		-8,392,900,000,000,000,000	
62	2015-06-11 14:23:14	5.344	84		-8,392,900,000,000,000,000	
62	2015-06-11 14:23:14	5.453	85		-8,392,900,000,000,000,000	

Table 4.6: Zeus session 1 malware samples

source process unique id	time date	time run	process unique id	command line token	process exe path token	command line.data
30	2015-06-08 16:59:33	20.411	114		-4,288,820,000,000,000,000	
114	2015-06-08 16:59:44	1.547	116		-8,945,540,000,000,000,000	
116	2015-06-08 16:59:55	1.422	122		8,283,374,322,564,070,400	
114	2015-06-08 17:00:25	4.097	125	5,736,196,424,232,519,680	5,992,314,292,155,400,192	/C "C:\USERS\LAB2\APPDATA\LOCAL\TEMP\TMP8597B6A9.BAT"
125	2015-06-08 17:00:25	4.456	126	-9,086,250,000,000,000,000	-5,932,390,000,000,000,000	

Table 4.7: Zeus session 2 malware samples

source process unique id	time date	time run	process unique id	command line token	process exe path token	command line.data
30	2015-06-08 17:06:24	0.281	69		8,945,540,000,000,000,000	
69	2015-06-08 17:06:41	6.656	71		8,283,374,322,564,070,400	

4.6.5 Zeus malware session 2

To find the malicious processes in this dataset we used the path tokens of the malicious processes from the session 1 dataset. This gives us the following process ids 37 , 69, 71, 104 and 113. However process 37, 104, and 113 are coming from the path token present in all datasets. As these are not connected in anyway, by starting or getting started, by malicious processes we exclude them from the identified malicious processes.

5

Exploring the data

As the data is prepared, we are able to perform our exploratory research. The aim of this analysis is to confirm if the use of a process tree, combined with the events per second triggered by a process could provide insight in the behavior of a process.

This chapter will describe the steps taken and the conclusions that can be drawn from the conducted research. In the previous chapters is mentioned that the behavior of starting processes can be modeled in a process tree, this will be done in section 5.1. The next section, Process activities, an in depth look will be given on the characteristics of the activities a process performs. After which in section 5.3 heatmaps are constructed, to provide a graphical overview of the possibility of grouping together same type of processes based on the process properties. Furthermore we look into the process appearing in all datasets but marked as possible malicious in the previous chapter. Section 5.5 will provide the conclusion of the explorative data analysis.

5.1 Process trees

As stated before we are able to construct process trees from the collected datasets. These trees provide a graphical representation of which processes started an other process or processes. In constructing the trees, we aim to find certain regularities or matching patterns. By constructing the trees we find the depth of each node. This information is added to all the processes in each dataset and shall later be used in the comparing methods, explained in chapter 6.

First we will explain how the trees are constructed. In subsection 5.1.2 the process trees from the clean datasets are shown and analyzed, after which we will look at the trees from the malware sets. This section will be finalized in subsection 5.1.6.

5.1.1 Constructing the process trees

To be able to construct the process trees, we will use the process create event type from the datasets. As explained in chapter 4 this event type contains information on the starting of a new process. The information used to build the trees are the unique source process id, the unique process id and the executable path token. As the names suggests, the unique source process id is the identification number of the parent process. The unique process id provides the identification number of the process started. This relationship can be transformed to the edges of the process tree.

The executable path token is a tokenized string of the path from which the executable, and thus the process, is started. As explained previously the processes with the same executable path token are in the same executables being executed. The token is used to color the outside of every process,

a node in the tree. Each process with the same executable path token has the same outer color. In doing so we can easily see which processes are started from the same executable.

The inside of the node will be colored by the cluster it is belong to. In doing so we can possibly identify same executables with same or different cluster color.

5.1.2 Process trees from the clean datasets

First we will analyze the process trees constructed from the four clean datasets. As the amount of nodes per dataset are rather high, ranging between 332 and 925 nodes, we only present the first six depths in figures 5.1 B.1, B.2 and B.3. The first figure can be found in this chapter, the latter can be found in appendix B.1. In table 5.1 is an overview of the number of nodes per depth of the tree. Analyzing this table we can conclude that the number of nodes per level of the tree starts to rise significantly from depth 5 until depth 9.

Table 5.1: Number of nodes on each depth per clean dataset

Depth	win8 1604 avond	win8 1604	win8 1704	win8 1804
0	1	1	1	1
1	1	1	1	1
2	3	3	3	3
3	4	4	4	4
4	9	12	12	9
5	39	62	73	41
6	218	366	373	163
7	110	132	108	76
8	56	256	210	14
9	6	77	86	7
10	2	11	10	3
11	3	0	0	4
12	4	0	0	6
13	6	0	0	0

In figure 5.1 the first node, at the top of the tree, is the kernel process from Windows. This process will start the userland process, which will spawn another three processes. If we look at the colors of the nodes we can see that the processes with unique ids 9998, 2 and 4 have the same executable path token. If we compare the four process trees we see that the first four levels of the tree are the same. From the fifth level onward differences between the trees start to occur. However on every level of the presented parts of the trees in this section we see comparable behavior. For example in all four trees, the node with unique id 7 starts most of the processes in last shown level.

Analyzing the presented data, we can conclude there are similarities present between the different process trees, the first four levels are completely the same if we look at the has. However going deeper down the tree the differences start show. This can be concluded from the graphical presentation as well from table 5.1 showing the number of nodes per level.

5.1.3 Process trees from the malware datasets

After analyzing the process trees from the clean datasets, we will now analyze the process trees generated from the malware datasets. In figure 5.2 and the figures in section B.1 the upper part of the trees are shown.

Here again the first four levels of the process tree are exactly the same. From the fifth level onward differences start to occur. In table 5.2 the number of nodes per level of the process tree are shown. Here again the number of nodes rise significantly from depth 5 onward. However the significantly rise of number of nodes per level stops at depth 8 instead of depth 9 as could be seen in trees from the clean datasets.

Table 5.2: Number of nodes on each depth per malware dataset

Depth	bank	rat session 1	rat session 2	zeus session 1	zeus session 2
0	1	1	1	1	1
1	1	1	1	1	1
2	3	3	3	3	3
3	4	4	4	4	4
4	7	6	8	6	6
5	26	31	32	26	27
6	74	105	81	75	59
7	12	16	23	14	16
8	29	57	51	26	74
9	4	0	9	0	0

5.1.4 Slimming the process trees

As can be seen in the previous sections, the process trees tend to grow very big. To create a better overview, nodes with similar cluster, executable path token, at the same depth and the same parent will be merged together. By doing this the size of the trees is considerably reduced. The number of nodes merged together is put into the edge. See figures B.8, B.9, B.10, B.11, B.12, B.13, B.14, B.15 and B.16 in appendix B.3.

5.1.5 Analyzing the process trees

The first difference that is visible is the blue node with process id 4 in the malware trees, which is brown in the clean datasets trees. This indicates this process is fit to another cluster in the malware sets compared to the clean datasets.

In table 5.3 the process activities and additional information for process id 4 in all datasets is presented. As can be seen the values on the process activities variables for the malware sets are higher than the clean datasets. This means that the number events per second is higher. If we look at the the running time for the processes it can be concluded that process 4 has a shorter running time in the malware sets.

By using the running time, it is possible to calculate a normalized value for the number of actions performed by the processes. In table 5.4 this calculated value is shown. If we look closely at this we can see that the normalized values are the same for all. It performs the same actions however, probably due to different "machines", the running time is different and hereby generating different values for events per second.

Although there is a difference between the machines on which the clean and malware datasets are recorded this shows a shortcoming for the method used for showing and using the process activities.

In this specific case the process in both datasets performed the exact same number of activities and so does not show any deviating behavior except for the running time.

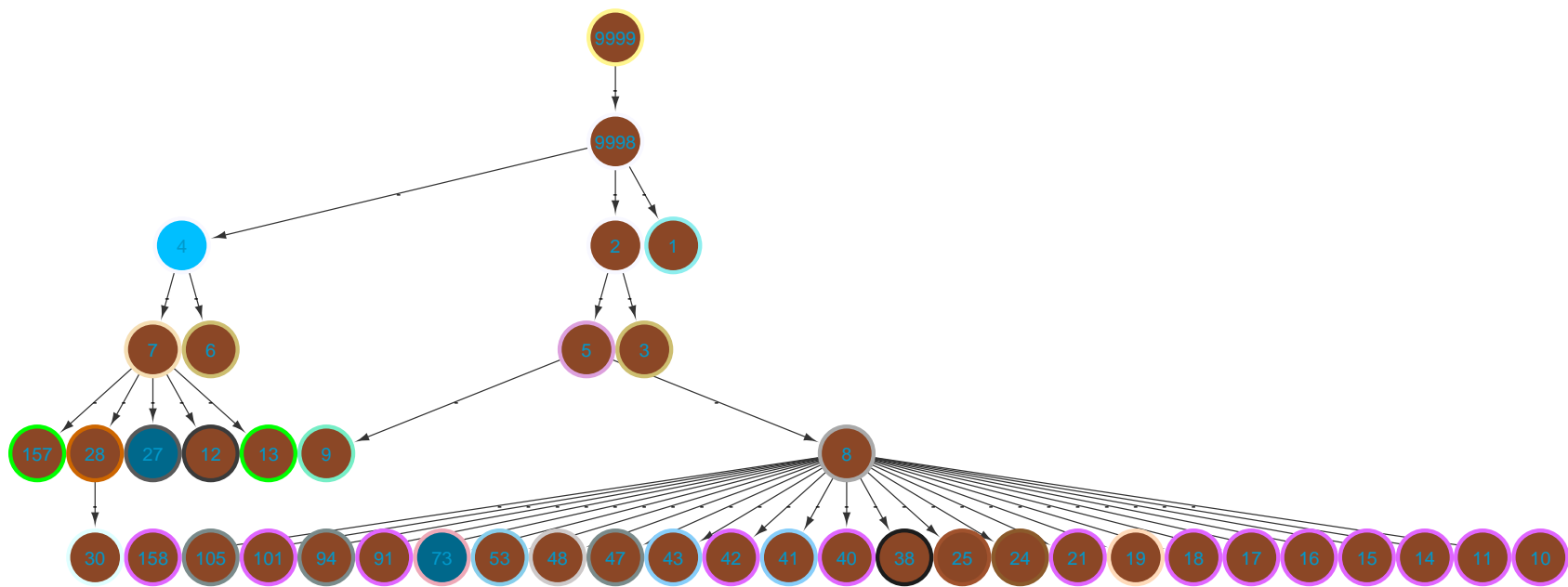


Figure 5.2: First 5 levels of bank malware

Table 5.3: summary of strange pid 4 behaviour

filesystem	registry	process create	thread create	module load	ob	unique ids	df	path token	parent unique id	parent path token	depth	nr childs	fit cluster	running time
0.195	0.209	6.656	1.664	0.070	1.743	4	malware bank event	-2,571,170 * 10 ¹²	9,998	-2,571,170 * 10 ¹²	2	2	8	0.047
0.117	0.125	3.995	0.999	0.042	1.046	4	malware rat session1 event	-2,571,170 * 10 ¹²	9,998	-2,571,170 * 10 ¹²	2	2	8	0.078
0.146	0.156	4.992	1.248	0.052	1.307	4	malware rat session2 event	-2,571,170 * 10 ¹²	9,998	-2,571,170 * 10 ¹²	2	2	8	0.062
0.194	0.209	6.653	1.663	0.070	1.742	4	malware zeus session1 event	-2,571,170 * 10 ¹²	9,998	-2,571,170 * 10 ¹²	2	2	8	0.047
0.194	0.209	6.653	1.663	0.070	1.742	4	malware zeus session2 event	-2,571,170 * 10 ¹²	9,998	-2,571,170 * 10 ¹²	2	2	8	0.047
0.008	0.008	0.266	0.067	0.003	0.070	4	win8 1604 avond event	-2,571,170 * 10 ¹²	9,998	-2,571,170 * 10 ¹²	2	2	3	1.172
0.007	0.007	0.230	0.057	0.002	0.060	4	win8 1604 event	-2,571,170 * 10 ¹²	9,998	-2,571,170 * 10 ¹²	2	2	3	1.359
0.007	0.007	0.227	0.057	0.002	0.059	4	win8 1704 event	-2,571,170 * 10 ¹²	9,998	-2,571,170 * 10 ¹²	2	2	3	1.375
0.007	0.008	0.241	0.060	0.003	0.063	4	zz_win8_1804_event	-2,571,170 * 10 ¹²	9,998	-2,571,170 * 10 ¹²	2	2	3	1.297

Table 5.4: Normalized number of events for process id 4

df	filesystem_calc	registry_calc	process.create_calc	thread.create_calc	module.load_calc	ob_calc
malware_bank_event	0.009	0.010	0.312	0.078	0.003	0.082
malware_rat_session1_event	0.009	0.010	0.312	0.078	0.003	0.082
malware_rat_session2_event	0.009	0.010	0.312	0.078	0.003	0.082
malware_zeus_session1_event	0.009	0.010	0.312	0.078	0.003	0.082
malware_zeus_session2_event	0.009	0.010	0.312	0.078	0.003	0.082
zz_win8_1604_avond_event	0.009	0.010	0.312	0.078	0.003	0.082
zz_win8_1604_event	0.009	0.010	0.312	0.078	0.003	0.082
zz_win8_1704_event	0.009	0.010	0.312	0.078	0.003	0.082
zz_win8_1804_event	0.009	0.010	0.312	0.078	0.003	0.082

Table 5.5: Some information on process 133 from the 1604 avond dataset. The comment line options show that it is the F-secure program

source process id unique id	time date	time run	process id unique id	process create event process exe path token	process create event command line data
48	2015-04-16 21:44:35	0.832	133	-8,264,910,000,000	/APP=FW /R "/DISP=F-SECURE CLIENT SECURITY 11.60" "/REMEXE=C:\PROGRAM FILES (X86)\F-SECURE\FWES\PROGRAM\FSFWWSCH.EXE"

By comparing the process trees the following that catches our attention is a pattern of light blue and pink processes in the 1604 avond and 1804 dataset which are not present in the 1604 and 1704 datasets. See figure 5.3 for a part of the 1604 avond process tree containing these processes. If we analyse one the process present in the screenshot, for example process 133 we see in the command line options that it is F-secure, table 5.5.

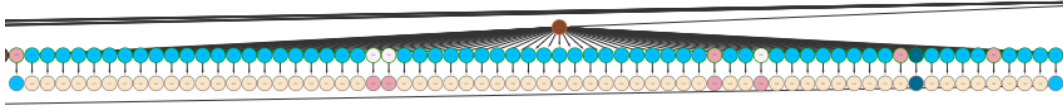


Figure 5.3: Part of 1604 avond showing the mentioned processes which are only visible in the 1604 avond and 1804 datasets.

During further analyzing of the merged process trees it was difficult to find any notable difference. The process tree shows that it is useful in graphically presenting the starting of and parent - child relation of the processes and some striking difference can be spotted by humans. It also proves that differences are present, as shown by the process id 4 discussed above, and including this information in the detection models could be useful.

5.1.5.1 Analysing the malicious parts of the process tree

In this section we will show the part of the process tree in which the malicious processes are present. First we will present the parts from the five malware datasets, see figures 5.4 to 5.8.

Looking at these figures we see that the malware in every dataset is visible in a certain part of the tree, namely the left side of the tree. To check if it is possible for an expert to spot anomalies in the process trees the same part of the process trees from the clean datasets are shown in figure 5.9 to 5.12.

If we compare the parts from the malware trees with the parts of the clean process trees, the process trees from both rat datasets catch our attention. The process 38 in the rat session 1 process tree spawns a lot of new processes. The same beholds for process 39 in the rat session 2 process tree. This type of behavior is not spotted in any of the other trees.

The malicious processes from the banking malware do attract attention however if we compare the banking malware processes see a comparable structure in the process tree of 1604 avond dataset.

5.1.6 Conclusion

In this section we analyzed the created process trees from the clean and malware datasets. By looking at the process trees we easily spotted the difference in behavior for process id 4 between the malware datasets and the clean datasets. As the fill color is based upon the cluster of the process a difference in color indicates different process behavior. After investigating this behavior we concluded that process 4 performs a certain number of activities. In the clean dataset the running time is longer than the malware dataset hereby creating another value for the number of process activities per second.

Likewise we spotted a difference in between the clean datasets 1604 avond and 1804 and 1604 and 1704. In the datasets 1604 avond and 1804 F-secure performed some actions spawning a lot of processes.

By looking at the process trees it can be concluded that from all presented trees the first four levels are exactly the same structure. Thereafter deviations start to occur. These deviations can be explained by the difference in computer usage per day. A colored node present in process tree A but not in process tree B can be explained by the fact a program was not used during that day or a new application was installed, as explained above with the F-secure application.

Due to the differences present between the process trees, because of different computer usage every day it is difficult to spot small anomalies. The small anomalies that can easily be detect by human eye, are those present in the first four levels of the tree, as shown with process 4, which got another cluster

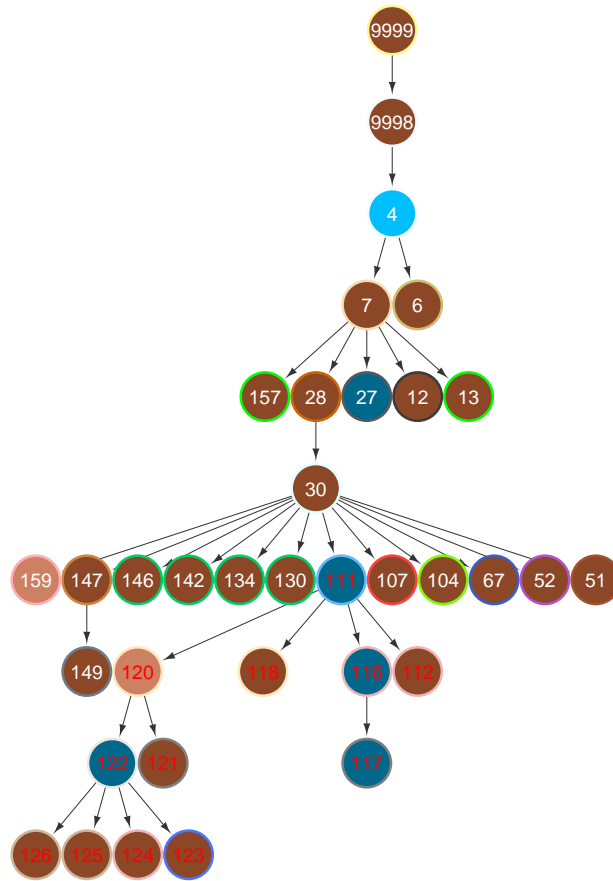


Figure 5.4: Part of process tree containing the malicious processes of the banking malware

As explained in the previous chapter process 111 is the malicious executable being executed. The child processes and their child processes all belong to the malware being executed. The process ids of these processes are 112, 116, 117, 118, 120, 121, 122, 123, 124, 125 and 126

color. However on the lower levels the number of nodes, processes, rise significantly and contain more differences.

However from analyzing the malware parts of the processes trees to their corresponding parts of the clean dataset we easily spotted the rat processes in both datasets.

Additionally the analysis of the malware parts of the process trees provided us with the insight that all malicious processes can be found at the same point in the process trees. However we must note that this might also be the effect of how we infected the computers as explained in section 4.2.2.

The main conclusion is that the process trees can be used by experts to spot anomalies, however in large trees small anomalies are rather difficult to detect.

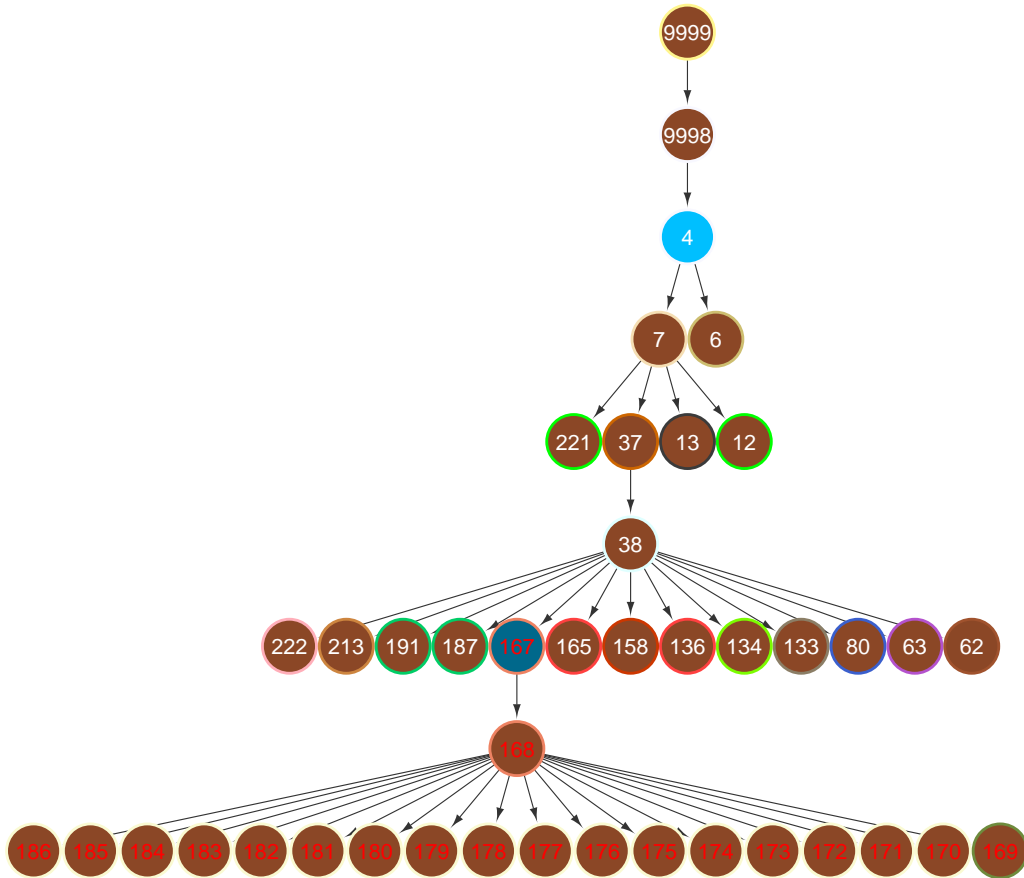


Figure 5.5: Part of process tree containing the malicious processes of rat 1

The malware executing is shown by process id 167. The processes with ids 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185 and 186 are all child processes of the malicious executable.

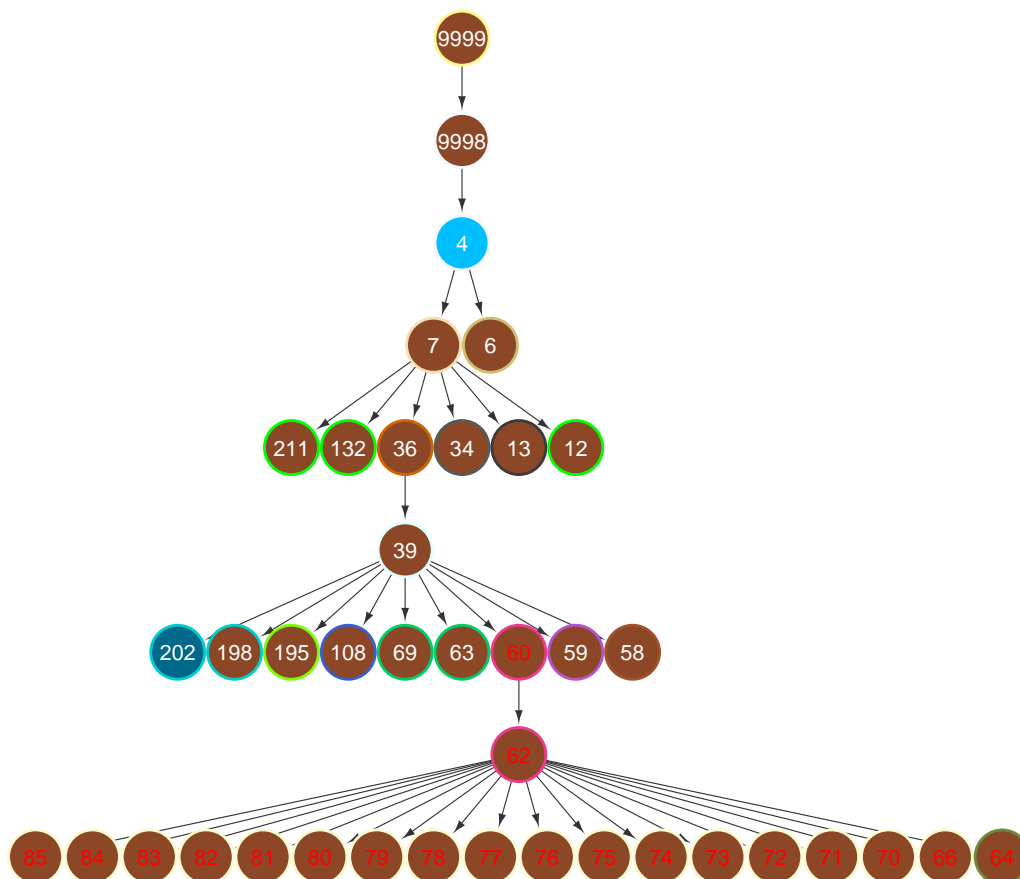


Figure 5.6: Part of process tree containing the malicious processes of rat 2

When the infected computer is restarted the malicious process has an id of 60, which starts the following child processes 62, 64, 66, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84 and 85.

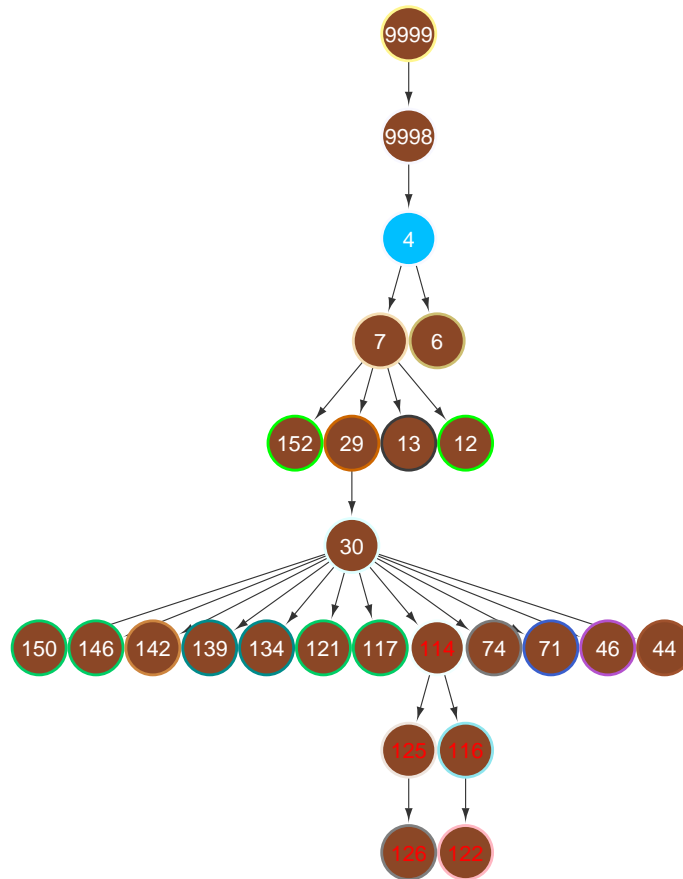


Figure 5.7: Part of process tree containing the malicious processes of zeus 1

Process id 114 is the malicious executable started, which spawns the following child process: 116, 122, 125 and 126.

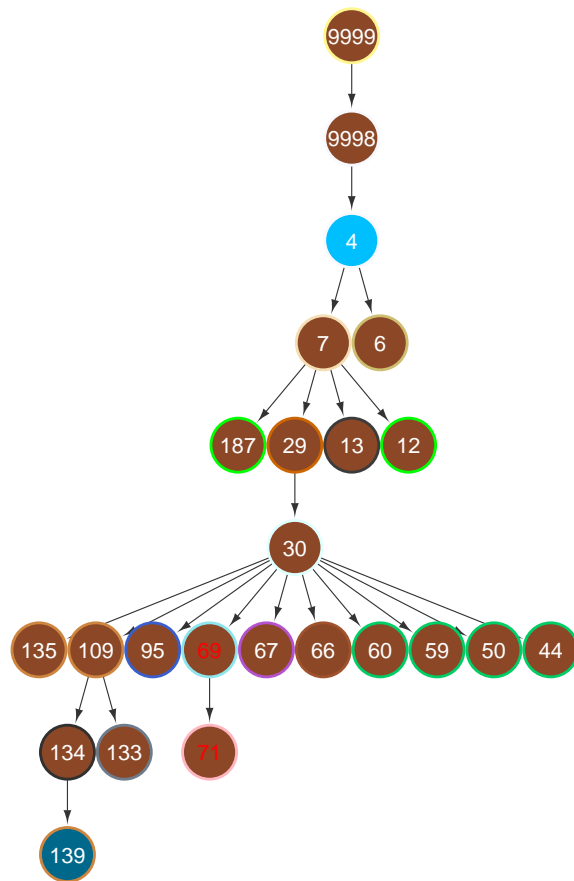


Figure 5.8: Part of process tree containing the malicious processes of zeus 2

Restarting the infected computer the Zeus malware is started by processes 69 after which the malicious process 71 is started.

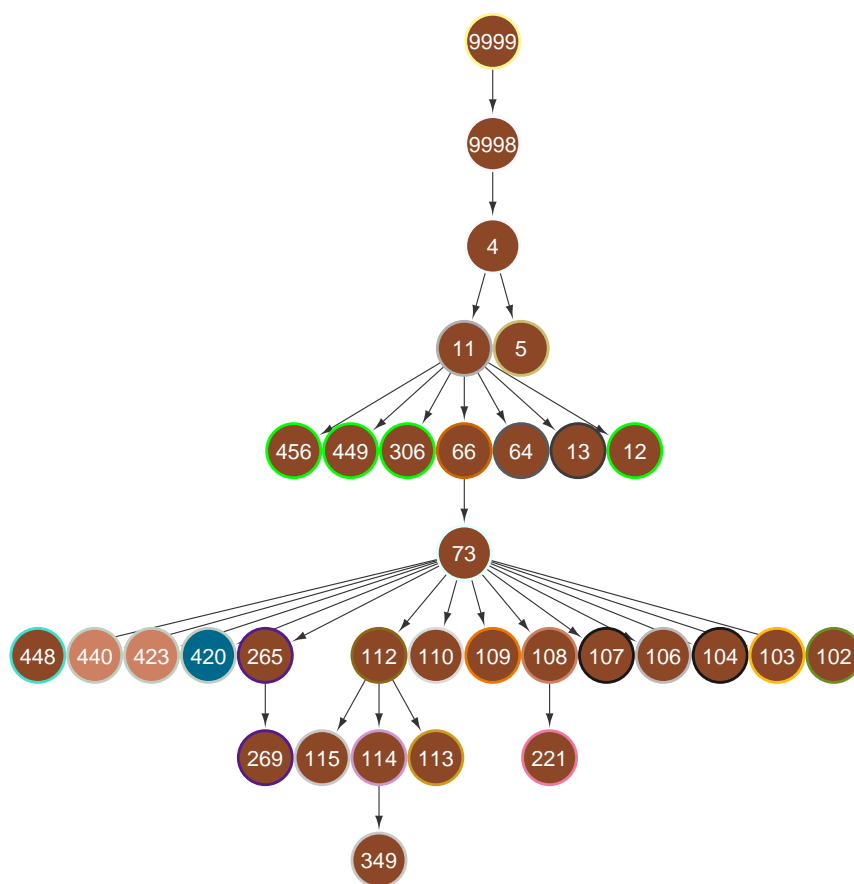


Figure 5.9: Part of process tree containing the same part as malicious 1604 avond

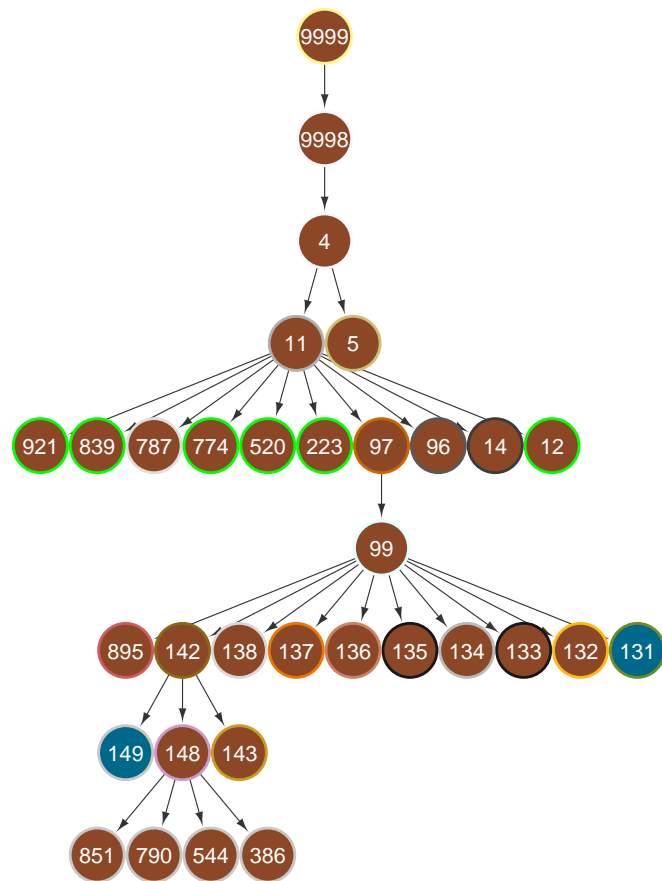


Figure 5.10: Part of process tree containing the same part as malicious 1604

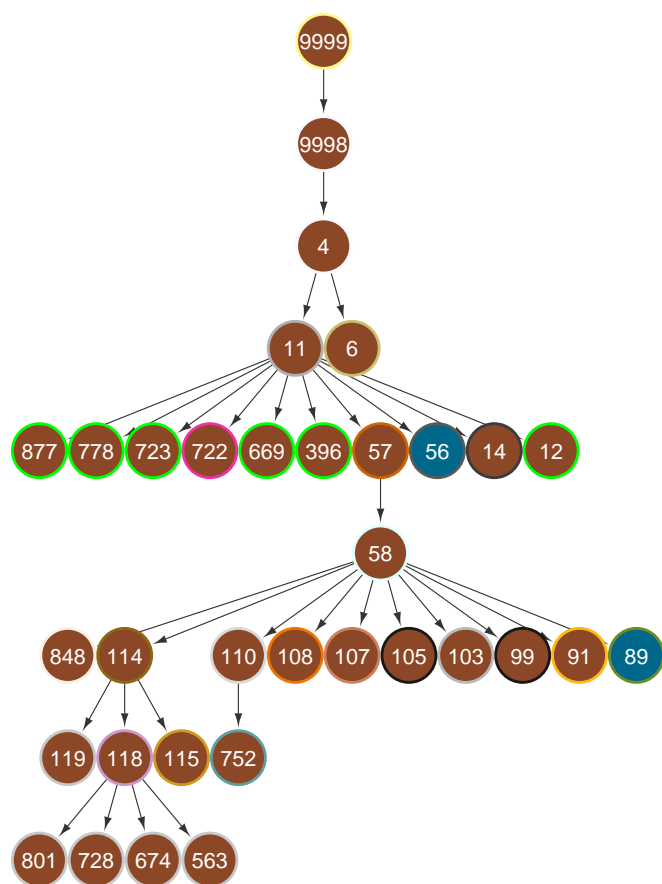


Figure 5.11: Part of process tree containing the same part as malicious 1704

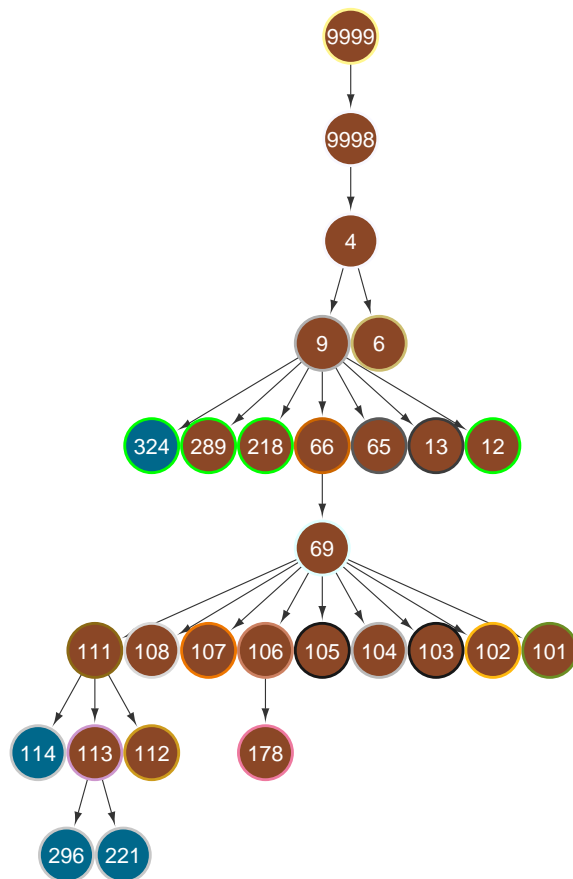


Figure 5.12: Part of process tree containing the same part as malicious 1804

5.2 Process activities

In this section a closer look will be taken on the event types triggered by the processes. These event types are described in section 4.1. Although eight different event types were discussed there, as stated in section 4.4 only six of these event types are used in the analysis, namely registry, filesystem, process create, thread create, module load and object callback events. In section 4.5 Data preparation we explained that all the data event types are normalized from zero to ten.

The first subsection will present an analysis on the clean dataset. Subsection 5.2.2 will provide an analysis on the malware datasets. The last subsection will conclude this section.

5.2.1 Process activities clean data

In the tables 5.6 to 5.9 a summary is shown of the normalized event types per second performed by the processes. The reason for looking at this data, is that we believe that the same processes will behave more or less the same. This will be explored more in depth in the next section, 5.3. In this subsection and the next the focus is on looking at the spread of the data and the presence of outliers.

Table 5.6: Showing the summary of the activities of the 1604 dataset

summary	filesystem	registry	process create	thread create	module load	ob
Min. :	0.000000	0.000000	0.000000	0.000000	0.0000000	0.000000
1st Qu.:	0.002168	0.000988	0.000000	0.000976	0.0005982	0.003218
Median :	0.024149	0.006895	0.000000	0.023733	0.0072140	0.031365
Mean :	0.328462	0.142610	0.08256	0.225443	0.1223209	0.292571
3rd Qu.:	0.303608	0.137681	0.000000	0.218688	0.1121780	0.264174
Max. :	4.676923	6.980502	5.000000	5.000000	2.8284983	5.485232

Table 5.7: Showing the summary of the activities of the 1604 avond dataset

summary	filesystem	registry	process create	thread create	module load	ob
Min. :	0.000000	0.000000	0.000000	0.000000	0.0000000	0.000001
1st Qu.:	0.002056	0.001266	0.000000	0.001362	0.0004488	0.002659
Median :	0.083853	0.033217	0.000000	0.030531	0.0159009	0.043198
Mean :	0.369895	0.178338	0.419629	0.688484	0.2329262	0.701378
3rd Qu.:	0.442886	0.217050	0.000369	1.085580	0.3225061	1.188556
Max. :	8.080727	3.118524	10.000000	10.000000	3.1384615	10.000000

Table 5.8: Showing the summary of the activities of the 1704 dataset

summary	filesystem	registry	process create	thread create	module load	ob
Min. :	0.000000	0.000000	0.0000	0.000000	0.000000	0.000000
1st Qu.:	0.003239	0.001653	0.0000	0.001297	0.000927	0.005414
Median :	0.058352	0.018427	0.0000	0.025952	0.009187	0.031685
Mean :	0.438194	0.201396	0.1177	0.278942	0.166397	0.360093
3rd Qu.:	0.566566	0.184636	0.0000	0.285714	0.151903	0.370418
Max. :	10.000000	6.980502	5.0000	4.984153	10.000000	6.566857

Table 5.9: Showing the summary of the activities of the 1804 dataset

summary	filesystem	registry	process create	thread create	module load	ob
Min. :	0.000000	0.000000	0.000000	0.000000	0.0000000	0.000003
1st Qu.:	0.002356	0.001566	0.000000	0.001269	0.0002632	0.003285
Median :	0.099642	0.041555	0.000000	0.040091	0.0293409	0.063676
Mean :	0.386103	0.199861	0.485468	0.760402	0.2491352	0.764166
3rd Qu.:	0.482791	0.202927	0.000274	0.997774	0.2507580	0.950261
Max. :	4.560000	3.226149	10.000000	10.000000	3.1384615	10.000000

If we look at the data presented in the summarizing tables we can conclude that three out of the four datasets contain processes that have a maximum value two or three of the event types. Both the datasets with a shorter time span, win8 1604 avond and win8 1804 contain a maximum value of ten on the process create, thread create and object callback event types. If we look at the boxplot of both datasets in appendix C.1 it can be seen that these values are outliers compared to the rest of the datapoints.

Dataset win8 1704 contains the maximum values on the event types filesystem and module load. Here again we can conclude from analyzing the boxplot from this dataset that these values are outliers as well, compared to the remaining datapoints. In finding an explanation for these high values we looked into the belonging processes. From the analysis we could conclude the one of the possible reasons here for, might be the very short running time, smaller than 0.1 seconds, of these processes.

Although the values can be considered outliers, they are still included in our further analysis. The reasoning behind this decision, is the fact that these kind of processes are present in three out of the four datasets and could be considered normal behavior. However we must note that these kind of values might trigger false positives.

5.2.2 Process activities malware data

The summary of the event types per second of the malware datasets are presented in the tables 5.10 to 5.12. The corresponding boxplots can be found in appendix C.2.

Table 5.10: Showing the summary of the activities of the bank malware dataset

summary	filesystem	registry	process create	thread create	module load	ob
Min. :	0.000000	0.000000	0.00000	0.0000000	0.000000	0.0000039
1st Qu.:	0.001297	0.000630	0.00000	0.0007964	0.000140	0.0016364
Median :	0.005858	0.002136	0.00000	0.0025946	0.001486	0.0053005
Mean :	0.128868	0.116939	0.08236	0.0720551	0.058645	0.0874600
3rd Qu.:	0.031734	0.007615	0.00000	0.0234175	0.008731	0.0391851
Max. :	2.780663	10.000000	6.65597	2.9041070	1.984029	2.9238629

Investigating these tables, only the bank malware of the five datasets contains a sensor event type with a maximum value of ten, namely the registry event type. If we compare this to the highest values on the registry event type from the other malware dataframes, it can be concluded it is significantly higher. The maximum values of the registry event type on the other malware dataframes ranges from 1.35 up to 1.54. Comparing it to the maximum registry values from the clean dataframes the difference is a lot smaller, as their maximum values range from 3.12 up 6.98. Looking into the data the process creating this spike in registry events we find the following command line option "AEINV.DLL,UPDATESOFTWAREINVENTORY", this process is part of Application Experience

Table 5.11: Showing the summary of the activities of the rat session 1 malware dataset

summary	filesystem	registry	process create	thread create	module load	ob
Min. :	0.000000	0.0000000	0.00000	0.000000	0.0000000	0.000003
1st Qu.:	0.000944	0.0003643	0.00000	0.000565	0.0002172	0.001801
Median :	0.005635	0.0022398	0.00000	0.003090	0.0026207	0.008487
Mean :	0.124576	0.0452931	0.05884	0.088486	0.0608884	0.106372
3rd Qu.:	0.036205	0.0155300	0.00000	0.031225	0.0169519	0.050645
Max. :	3.830400	1.3770067	5.20000	3.490977	2.1685129	3.514725

Table 5.12: Showing the summary of the activities of the rat session 2 malware dataset

summary	filesystem	registry	process create	thread create	module load	ob
Min. :	0.000000	0.0000000	0.00000	0.0000000	0.0000000	0.000007
1st Qu.:	0.000719	0.0002402	0.00000	0.0002582	0.0001877	0.001417
Median :	0.003840	0.0017536	0.00000	0.0017400	0.0015653	0.005850
Mean :	0.111748	0.0509413	0.05237	0.0633187	0.0445139	0.096124
3rd Qu.:	0.033066	0.0157437	0.00000	0.0111227	0.0168763	0.038554
Max. :	1.909957	1.4669178	4.99206	2.1839720	0.6778102	4.167604

Table 5.13: Showing the summary of the activities of the zeus session 1 malware dataset

summary	filesystem	registry	process create	thread create	module load	ob
Min. :	0.000000	0.000000	0.0000	0.000000	0.0000000	0.0000075
1st Qu.:	0.001549	0.000641	0.0000	0.001198	0.0001521	0.0019734
Median :	0.008152	0.003062	0.0000	0.003672	0.0020162	0.0083775
Mean :	0.143989	0.058154	0.1123	0.105147	0.0685082	0.1254814
3rd Qu.:	0.065687	0.016291	0.0000	0.035294	0.0149633	0.0415372
Max. :	4.852308	1.511173	6.6526	2.500000	1.8307692	3.0952381

Table 5.14: Showing the summary of the activities of the zeus session 2 malware dataset

summary	filesystem	registry	process create	thread create	module load	ob
Min. :	0.000000	0.0000000	0.00000	0.000000	0.0000000	0.0000066
1st Qu.:	0.001816	0.0008168	0.00000	0.001389	0.0003483	0.0027285
Median :	0.018244	0.0050704	0.00000	0.007808	0.0025791	0.0118176
Mean :	0.216552	0.0556587	0.08997	0.107069	0.0669981	0.1193144
3rd Qu.:	0.122717	0.0464325	0.00000	0.060274	0.0313125	0.0742206
Max. :	4.836681	1.3515012	6.66667	2.491949	1.9380000	3.0852703

Program Inventory Component from Microsoft.

If we look closer to the differences in the maximum values from the clean datasets and the malware sets, the following can be concluded: filesystem ranges from 1.91 up to 4.85 in the malware set and 4.56 up to 10 in the clean set. The process create range is 4.99 to 6.67 versus 5 to 10 in the clean datasets. In the clean dataset the maximum thread create value ranges from 4.98 to 10 while in the malware set it ranges from 2.18 to 3.49. For the module load it respectively 0.68 to 2.17 and 3.14 up to 10. The lowest value for the object callback in the malware sets is 2.92 and the highest is 4.17, while in the clean sets it ranges from 5.49 to 10.

5.2.3 Conclusion

By examining the summarized information in the tables and the graphical presentation in the boxplots it is shown that the events per second on each event type are unevenly spread. Although we would have expected more high values on the events in the malware datasets, only the registry event type has the maximum normalized value in the malware datasets. However these differences might be explained by the limitation we had in collecting the malware datasets, as stated in section 4.1. As we could not run the malware samples on the same computer as the clean datasets were collected, the lack of some of the used applications could explain these deviations.

Although the malware datasets do not contain as many outliers as the clean datasets. Analyzing this data shows that there are deviations in the number of events triggered per process. In this analysis a summarized view was presented and it was not possible to see if the processes from the same executable, show comparable number of events per second. To be able to analyze this, the next section will focus on constructing heatmaps, to give a graphical presentation of which processes are sharing almost the same number of events per second.

5.3 Heatmaps

In the previous section an overall analysis on the event types per second was conducted. The conclusion from this analysis was that there is quite large spread between the processes. There for in this section the focus will be on finding possible grouping of processes based on the events per second. This will be done by creating for each dataset a heatmap. In a heatmap the processes will be grouped together based on the distances between the events per second of the processes. To calculate these distances the Euclidean method is used.

To be able to see if processes with the same executable path are being grouped together the colors for every unique executable path token will be used again. At the left side of the heatmaps a matching color means it has the same executable path.

The heatmap can be used to easily spot processes which have high values on certain process variables. In a heatmap the coloring will be done based on the values present in the column. The higher the value the whiter the cell becomes, the lower the value the more yellow a cell is colored. This can be used to easily spot the processes with high process activities.

However a high process activity does not imply that a process is malicious, as it could be normal behavior for such a process.

5.3.1 Heatmaps from the clean datasets

In figure 5.13 and 5.14 a small part from the heatmap from the win8 1604 dataset is shown. The full heatmaps can be found in appendix C.1. The first figure shows that the processes, at least some, with the same executable path seem to group together with processes that have a similar events per second characteristic. However the second figure shows a part from the same heatmap where processes with the same executable token don not group as nicely together as in the first figure.

Examining the heatmaps from all clean datasets, it can be concluded that from the process executable tokens that appear several times are grouping together. However not all similar tokens are

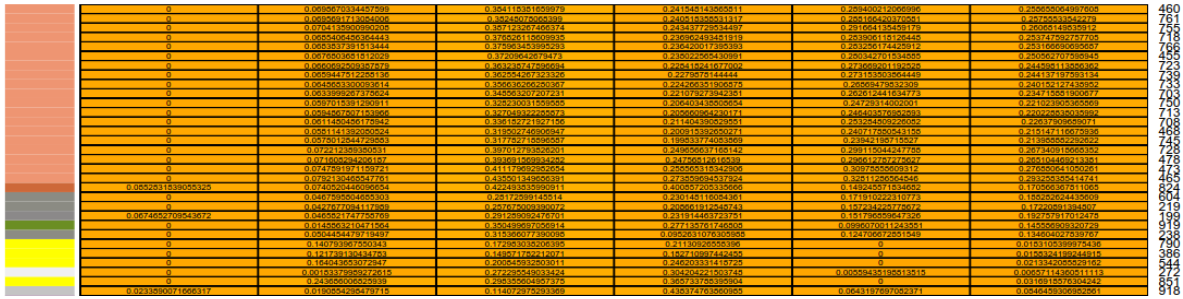


Figure 5.13: Part from the heatmap from dataset win8 1604

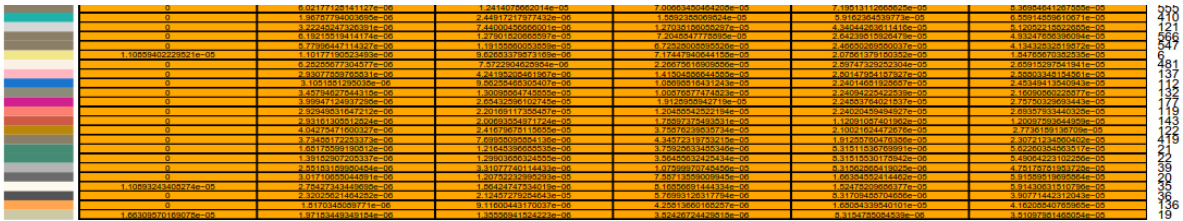


Figure 5.14: Another Part from the heatmap from dataset win8 1604

grouped together. In the heatmap from win8 1704 dataset, for example, several groups of white process executable tokens can be found on different places of the heatmap.

5.3.2 Heatmaps from the malware datasets

In figure 5.15 a cut from the malware dataset zeus session2 is shown. The heatmaps from the malware datasets do perform according to the same description as provided for the clean datasets. As can be seen, a rise in the normalized values changes the color of the cell. The higher the value the more yellow to white the cells become. Better examples will be shown in the next section.

5.3.3 Analyzing the heatmaps

In appendix D.1.1 several cutouts from the created heatmaps are shown. These figures show clearly that processes with almost the same normalized values for the process events tend to group together. For example figure D.6. In figure D.7 we can clearly see that processes that have a deviation in the events per second value are clearly visible. Furthermore the early explained outliers on process create, thread create and ob can clearly be seen in figure D.9. Also the dataset from the banking malware shows some deviations in the process activities, see figure D.10, as well the zeus session 1 malware set, figure D.14.

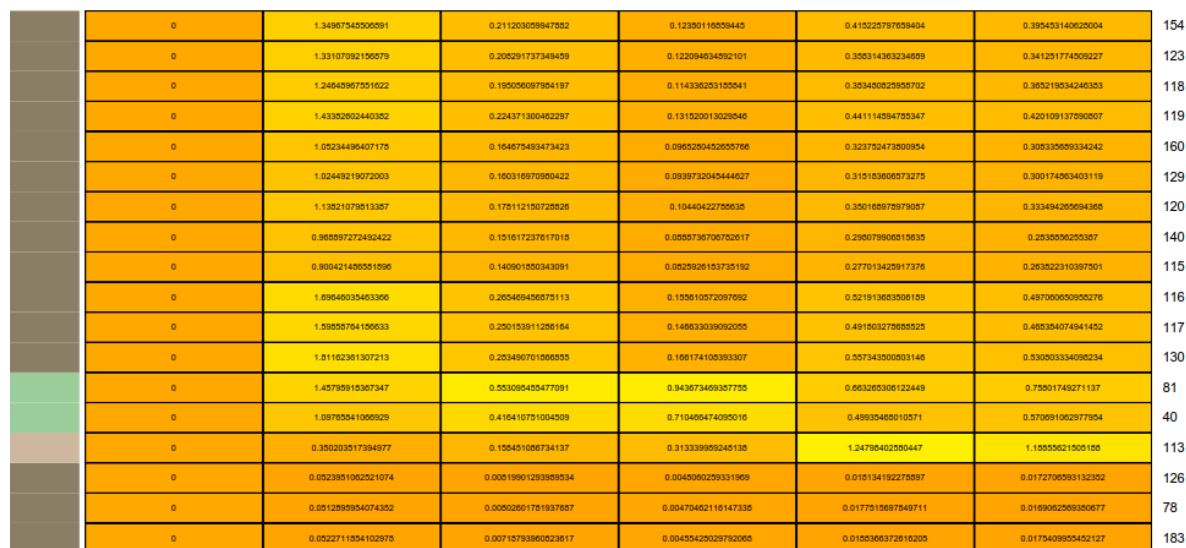


Figure 5.15: Part from the heatmap from dataset zeus session 2

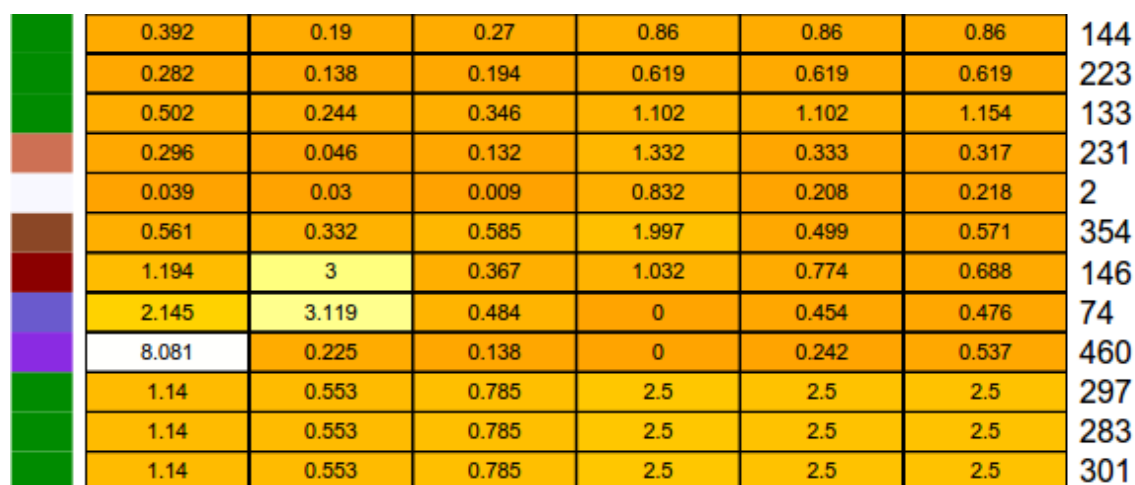


Figure 5.16: Part 1 from the heatmap from 1604 avond with the malicious processes

filesystem, registry, module load, process create, thread create, ob

In figure 5.16 we see that filesystem for process 460 is really different from other processes around. Comparable process is 459 see table 5.15 and 5.16.

Table 5.15: Starting of process 459 and 460 in win8 1604 avond

source process unique id	time date	time run	process unique id	process exe path token	command line data	command line token
47	2015-04-16 23:48:09.108863	2.641	459	1,294,089,667,656,760	-U -P 2080 -S 716	5,042,687,691,843,479,552
47	2015-04-16 23:48:09.124435	2.656	460	1,294,089,667,656,760		5,042,687,691,843,479,552

Table 5.16: Process 459 and 460 activity

filesys- tem	reg- istry	pro- cess create	thread create	mod- ule load	ob	unique ids	df	path token	parent unique id	parent path token	depth	nr childs	fit.clusterun- ning time
2.105	0.533	0	0	1.569	1.66	459	win8 1604 avond event	1,294,089,667,	47	7,932,894,714,	6	0	7 0.031
8.081	0.225	0	0.242	0.138	0.53	460	win8 1604 avond event	1,294,089,667,	47	7,932,894,714,	6	0	7 0.484

5.3.4 Analyzing the malware heatmaps

Analyzing the heatmap cutouts with the malware present, visible by the red process ids, we see in the following heatmaps processes start to show deviating behavior: Bank 5.17, rat 1 5.18. However still other processes there are interesting as they show deviating values. Important to note is that in all the malware heatmaps process id 4 is showing deviating behavior. However this could be connected to the behavior explained in 5.4. More heatmap cutouts can be found in appendix H.5.1

5.3.5 Conclusion

In this section a graphical presentation was given of the possibility of the processes grouping together. To present this heatmaps were used. It can be concluded that there is tendency of similar processes and processes with the same executable path token to group together. However not all processes with the same token tend to cluster in one group. It still provides evidence that clustering of the data might provide new insights.

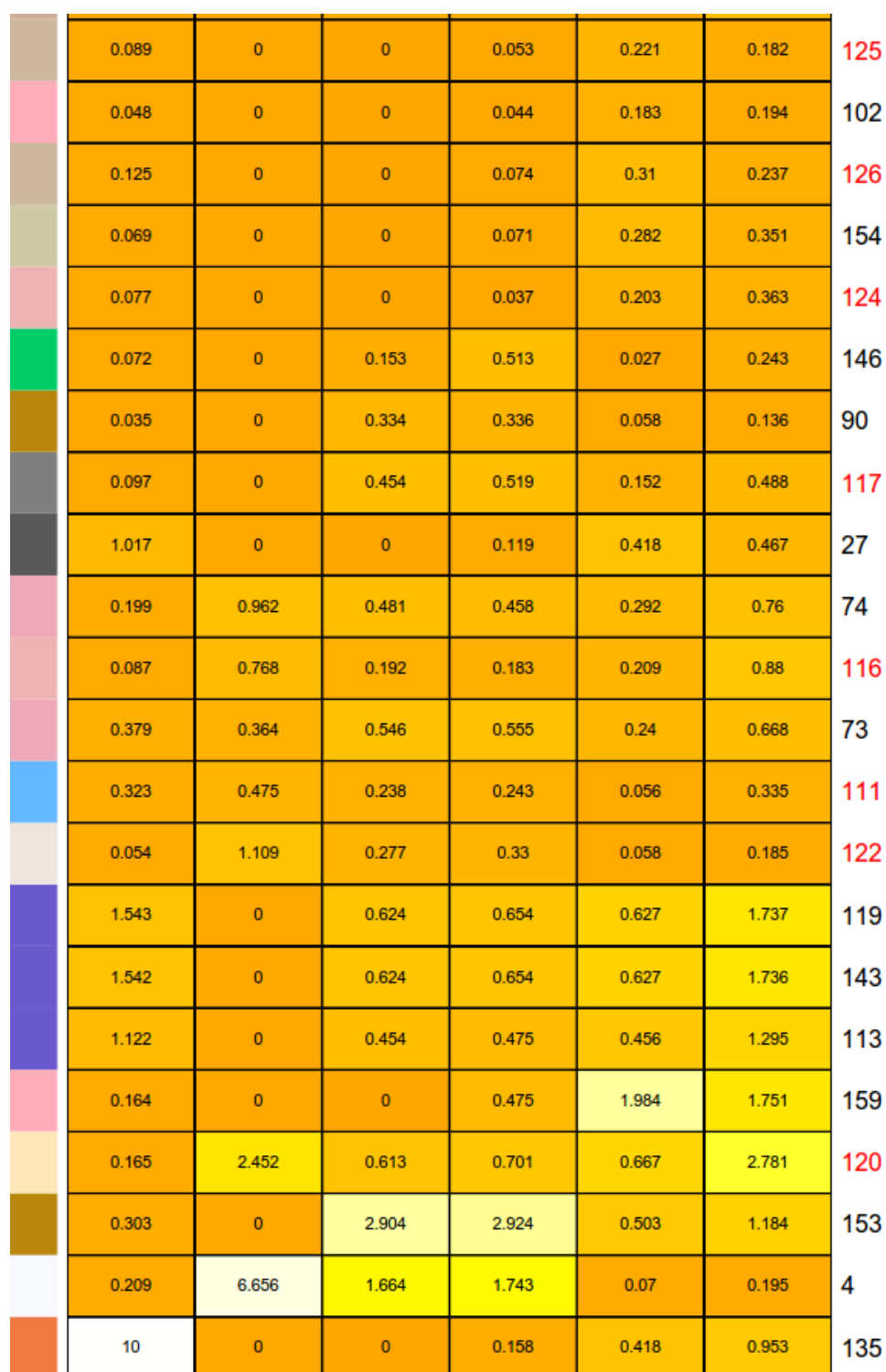


Figure 5.17: Part 3 from the heatmap from banking malware heatmap with the malicious processes

With the following column order:Registry, process create, thread create, ob, module load, filesystem

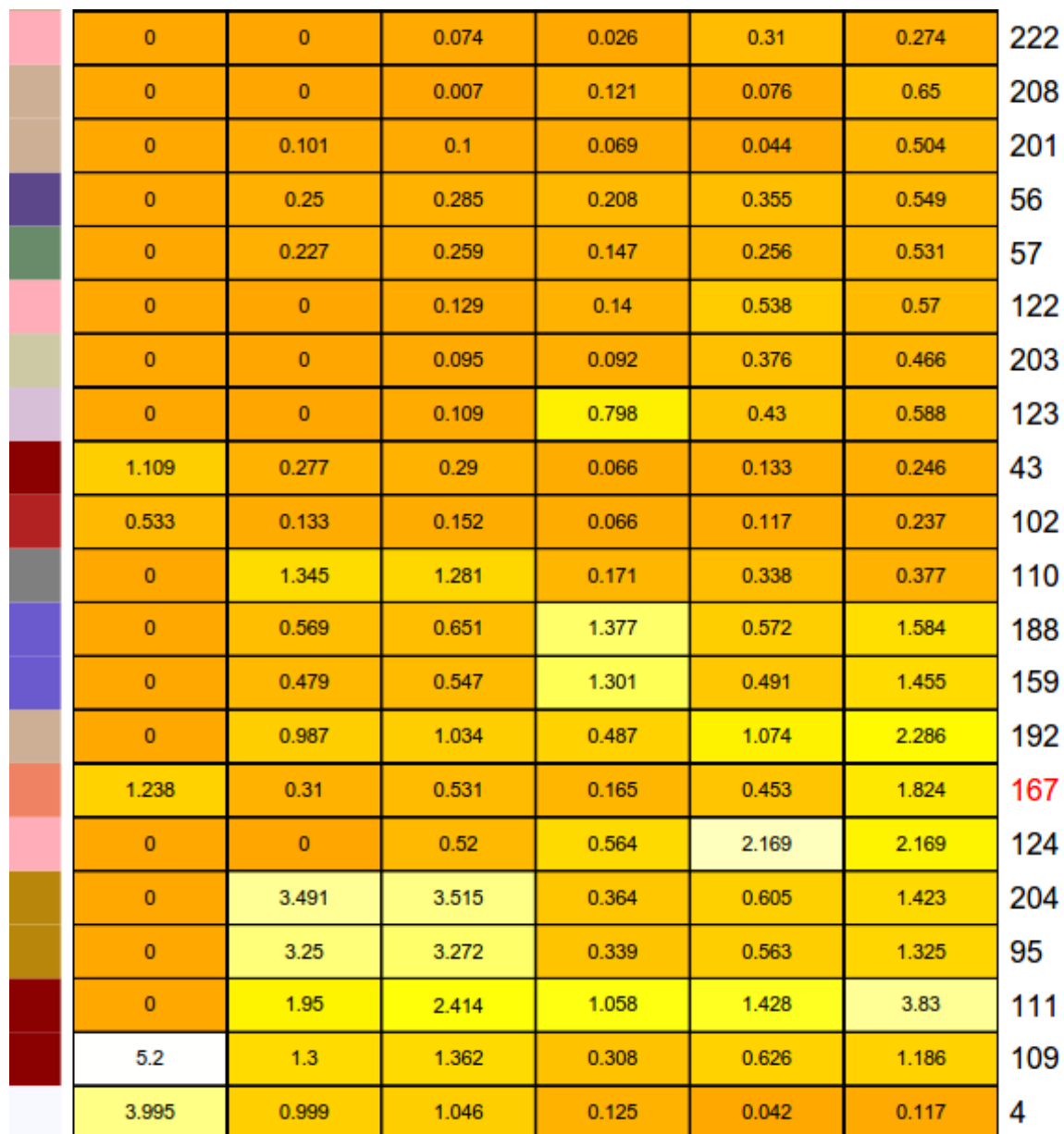


Figure 5.18: Part 5 from the heatmap from rat session 1 with the malicious processes

With the following column order: process create, thread create, ob, registry, module load, filesystem

5.4 Possible benign process

As stated in the previous chapter a path token was marked as malicious, which is present in every dataset. In this section we will explore if the malicious marked processes in the Zeus session 1 dataset shows a deviation from the other processes. Furthermore we will check if the processes in zeus session 2 show a deviation.

In figure D.17 in appendix D.2 the heatmap is shown of the path token in all the datasets. The first striking color is for the process create event of process 243 in the win8 1704 dataset, see figure 5.19. This is the only process that creates another process. Furthermore at the bottom of the heatmap a change in color is detectable, however all these processes belong to clean datasets, figure 5.20. Concluding from this analysis the process marked as malicious in zeus session 1 does not show any noticable deviations from the other process. What is striking is that only the processes from the clean datasets seem to show a deviation.

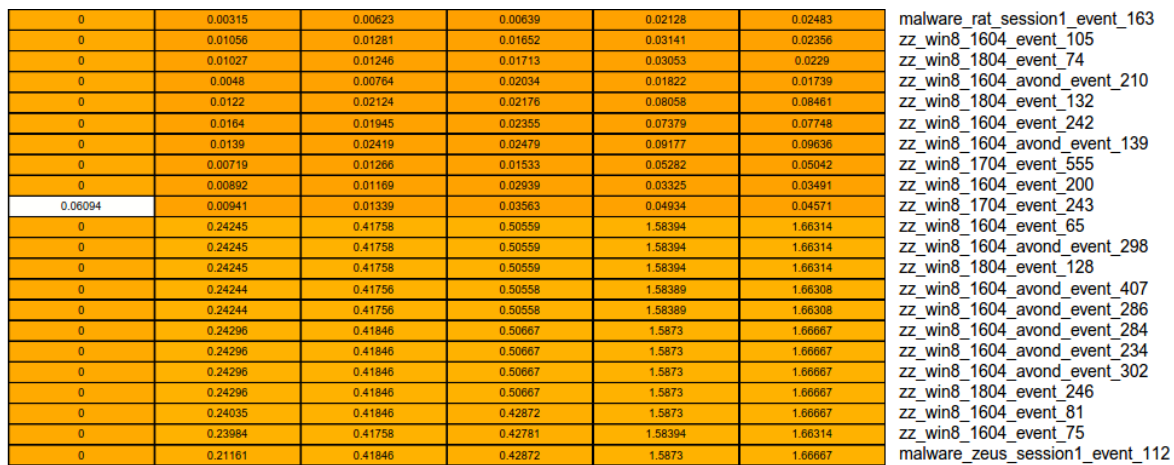


Figure 5.19: Heatmap showing process 243

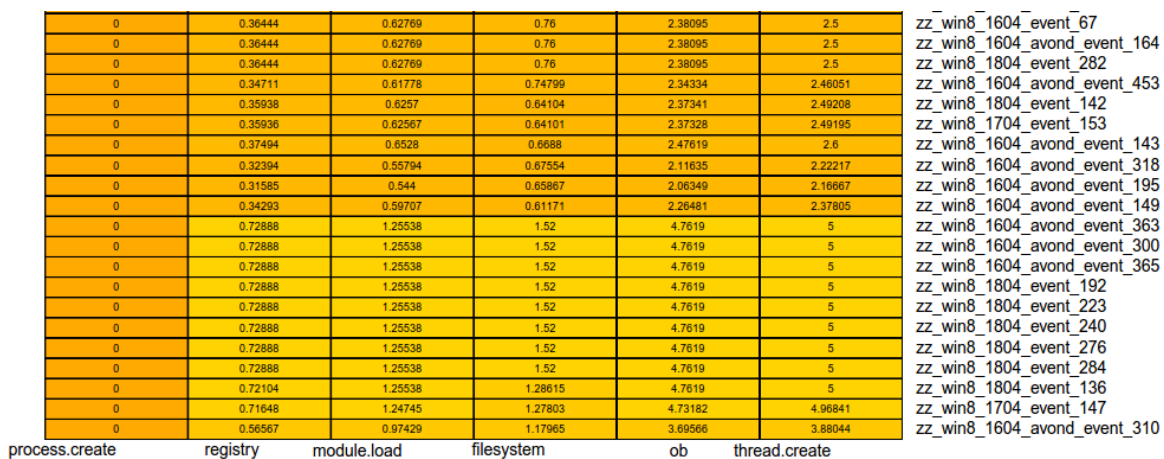


Figure 5.20: Heatmap showing the processes

5.5 Conclusion

In this chapter an explorative analysis was performed. Hereby we analyzed whether process trees and the activities of processes per second could provide useful information to characterize a process. The process tree analysis showed that there are recognizable patterns present in the trees. In all constructed trees the first four levels of the tree matched with each other, except for the color of process 4 in the malware datasets. From there on the trees started to branch widely and dissimilarities could be found. These differences between trees can probably be ascribed to the fact of different usage of applications between the datasets. However these different usage patterns will mostly be the case.

The second section of the chapter showed an in depth look on the process activities. The conclusion from this analysis was that the clean datasamples showed a wider range in the normalized events per second values in comparison to the malware datasets. However both collections of datasets contained outliers in the data, this shows the possibility of detecting deviations in behavior.

The last section of this chapter analyzed if the grouping of the same kind of processes would happen. By using heatmaps the evidence was provided that processes with the same executable path token, so the same type of application, have the tendency to group together.

To summarize this chapter we can conclude that the proposed variables, events per second, location in the process tree and clustering, can be used to describe process characteristics.

In the next chapter three algorithms will be proposed to compare the malware datasets against the clean datasets. The proposed algorithms will be evaluated in chapter 7.

6

Building the detection method

In the previous chapter it was proven that the use of process tree information, events per second triggered by a process and cluster information are viable variables for comparing malware datasets against the clean datasets. In this chapter, three different algorithms will be described that can be used for the detection of malicious processes. These three algorithms will compare the datasets containing the malicious processes against the clean datasets.

In the first section we will describe the basic setup of the algorithms after which each algorithm will be discussed separately in sections 6.1.1, 6.1.2 and 6.1.3. The section thereafter will discuss a method for ranking the malicious marked processes. Section 6.3 will show an overview of the running times of each comparing method, after which this chapter will end with the conclusion.

6.1 Comparing methods

Before the three different algorithms will be explained in detail, this section will describe the steps that are general for all three algorithms. Such as calculating a distance matrix and the selection of processes to be marked as malicious.

All three algorithms make use of a distance matrix. As stated earlier in this master thesis we assume that malicious processes show deviations in the number of process activities performed per second. Therefore calculating the distance between the processes, based on the variables below, will give a higher distance when the process activities of the processes deviate from each other. The distance matrices in all three methods are calculated with the Euclidean distance, see equation 6.1, on the following variables:

- filesystem
- registry
- process create
- thread create
- module load
- ob
- depth

- fit cluster

$$\sqrt{\sum (x_i - y_i)^2} \quad (6.1)$$

The Euclidean distance is calculated between two vectors x and y with the dimension i [40, pp.509]. In this case the dimensions are the eight variables mentioned above. A distance matrix contains the distance of every combination of processes between both datasets.

As the usage of a programs can differ every day, comparing a dataset in which program A is used to a dataset where program A is not used might result in the marking of program A as malicious. However when comparing to a dataset in which program A the processes will be matched together.

To overcome this problem we will run the comparing methods, described in the following subsections, on each malware set against every clean dataset. As mentioned before the outcome of each comparison gives each process a distance between the process of the malware set and a matched process of the clean dataset. By using a threshold value, explained in the next chapter, processes above this threshold value will be marked as malicious. For every malware set compared to the four clean sets by a comparing algorithm, the processes above the threshold value will be gathered. When a process is marked in all four comparisons as malicious, it will be marked as malicious. If a process is set to malicious in only three or less comparisons it will not be marked as malicious.

In the next three subsections the three different algorithms will be described.

6.1.1 Compare method 1

This method is the loosest method and will probably be the fastest. It generates one distance matrix and will each time take the lowest distance. The method can be described as follow:

1. Create a distance matrix between between all the nodes from dataframe A (one of the four clean datasets) and dataframe B
2. select the lowest distance present in the distance matrix and match node B with the corresponding node A from dataframe A
3. set the distance to NA
4. repeat step 2 and 3 until all nodes from dataframe B are matched to a node in A

6.1.2 Compare method 2

Compare method 2 is the most strict comparing method. In this method we try firstly to match all nodes on each depth if they have the same path token and their parent has the same path token. There after we will loop again through all depths except this time there is no check on the path tokens. By doing this the we have to loop two times through the whole tree structure, making this method probably the slowest method.

The steps of compare method 2 are:

1. Loop through all depths from dataframe B
 - (a) select all nodes in datafram A and B on depth i
 - (b) create a distance matrix with these nodes
 - (c) select lowest distance
 - (d) check whether node B and node A both have the same executable token, as well as the same parent executable token if this is the case match node B with corresponding node A and set distance to NA. If this is not the case set distance to NA without matching
 - (e) repeat step 1a to 1d until all depths have been reached

- (f) select all nodes in dataframe B not matched
- (g) loop through all depths again and select lowest distance, however do not match on executable path token
- (h) select still unmatched nodes in dataframe B
- (i) create a distance matrix with all nodes from dataframe A
- (j) select lowest distance and set as matched

6.1.3 Compare method 3

The last method also tries to match per depth first, however without the strict rule of matching path tokens. Below are the steps from compare method 3.

1. Loop through all depths from dataframe B
 - (a) select all nodes in datafram A and B on depth i
 - (b) create a distance matrix with these nodes
 - (c) select lowest distance and match A to node B
 - (d) set distance NA
 - (e) perform step 1a to 1d again
2. select all unmatched nodes from dataframe B
3. create distance matrix
4. select lowest distance and match A to node B
5. repeat 4 until all nodes are matched

6.2 Ranking of malicious marked processes

After a malware set is compared to every clean set by using one of the three proposed algorithms and the processes above the threshold are marked as malicious the ranking of the process can be calculated. The ranking is based upon distances of the malicious marked process to the matched process in all the four clean sets.

6.3 Running times

As stated in the design requirements the speed of the algorithm is of importance. Therefor we recorded the running time of 10 runs for each algorithm. This is done by comparing two large datasets and a malwareset against a large dataset. The outcome of the large datasets are presented in 6.2 and the outcome of the malware against the large set are presented in 6.1.

As we can see in these tables algorithm 3 is almost 10 times as fast as algorithm 1 and 100 times as fast as algorithm 2. Depending on the outcome of the evaluation, algorithm 3 would be preferred.

Table 6.1: Running times (in seconds) of comparison methods using large dataset and banking malware

expr	min	lq	mean	median	uq	max	neval
Algorithm 1	13.554	13.625	14.135	13.723	14.856	15.328	10
Algorithm 2	197.314	197.544	197.787	197.825	198.046	198.243	10
Algorithm 3	1.481	1.495	1.500	1.505	1.507	1.509	10

Table 6.2: Running times (in seconds) of comparison methods using large datasets

expr	min	lq	mean	median	uq	max	neval
Algorithm 1	220.065	221.157	225.188	223.458	226.023	239.979	10
Algorithm 2	784.511	788.251	789.519	789.177	791.402	795.422	10
Algorithm 3	21.830	22.027	22.235	22.173	22.493	22.631	10

6.4 Conclusion

The above described algorithms will be run by using the clean malware datasets as dataframe A and the malware dataframes will be used as dataframe B. This implies that a malware set will be compared with 4 clean sets. Giving twenty constructed data frames per comparing method, so a total of 60 constructed dataframes.

7

Evaluation

In this chapter we will evaluate the proposed algorithms described in the previous chapter.

7.1 Evaluation Set-up

The evaluation will test if the algorithm can be used in detecting malicious processes. As explained in the previous chapter for every matched process the distance between the two processes will be calculated. In the beginning of this research thesis we already stated the assumption that we expect malicious processes to deviate their event activities from the normal behavior of benign processes. This will as well influence the distance between the matched nodes. Based on this assumption we will test if using the distance measure and using a threshold measure, the distance mean and 75, 80, 85, 90 and 95% quantile.

As explained in the previous chapter, the processes above the threshold value will be checked for presence in all four comparisons. If this is not the case the process will not be marked malicious. We will end up with a list of processes above the threshold in all comparisons with the clean datasets.

As we will be evaluating six threshold, this means that for every malware dataset we will run 6 tests on all four matched dataframes. For every matched dataframe we will analyze if the processes above the set threshold measure include the malicious processes. We will calculate the True Positive Rate, see equation 7.1, the number of malicious processes marked as malicious, the False Positive Rate, equation 7.2, the number of non malicious processes marked as malicious and the Accuracy 7.3, the total number of process marked correct [69, p.7].

$$TPR = TP / (TP + FN) \quad (7.1)$$

$$FPR = FP / (FP + TN) \quad (7.2)$$

$$ACC = (TP + TN) / (TP + TN + FP + FN) \quad (7.3)$$

The aim is to have a high TPR and a low FPR. When there are too much processes falsely marked as malicious the detection system would invoke too much time from a security manager to check whether the processes are really false or not.

Furthermore we will analyze if using a distance ranking, as described in the previous chapter, would help a security officer. In other words will the malicious processes be ranked high enough to say that

checking the first few process would suffice. During the evaluation of the algorithms we will analyze the top five non-malicious processes from the ranking, to evaluate why these processes are highly ranked.

During the analysis of the process trees we found that all malware was in the same area, although we are not sure if this is because of the way we executed it, so we could look at the process trees to see which process are marked malicious are in this area. Besides we will try to see if we can find any pattern for the malicious marked processes in the process tree.

7.2 Evaluation

The evaluation will be done by the following steps:

1. Analyze the TPR, FPR and ACC values
2. Analyze the top five ranked benign process from the distance ranking
3. Analyze the process tree

7.2.1 Algorithm 1

In the next sections the above described steps will be performed for algorithm 1.

7.2.1.1 Algorithm 1:Banking malware

As stated in section 4.6.1 the following twelve process ids of the 161 processes are malicious: 111, 112, 116, 117, 118, 120, 121, 122, 123, 124, 125, 126, from which the path tokens from process 117 and 121 appear in all datasets. This might imply that the malware is using non-malicious applications to perform malicious actions.

In table 7.1 an overview is presented of the number of processes marked as malicious, the number of processes above the threshold which are malicious and the FPR, TPR and ACC for the six different threshold types. An overview including the process ids marked as malicious and the real malicious process ids marked as malicious are shown in appendix F.1.

The data of table 7.1 is plot in figure 7.1 to provide an graphical overview of the TPR, FPR, ACC, number of processes marked malicious and number of processes which are marked malicious and are malicious.

Table 7.1: Outcome using different threshold values for bank malware using method1

threshold type	total nr above threshold	number malicious above threshold	FPR	TPR	ACC
mean	19	8	0.074	0.667	0.907
q0.75	35	11	0.161	0.917	0.845
q0.8	27	8	0.128	0.667	0.857
q0.85	21	8	0.087	0.667	0.894
q0.9	13	3	0.067	0.250	0.882
q0.95	3	1	0.013	0.083	0.919

Looking at the FPR, TPR and ACC rates in the table, it can be concluded that different threshold values give different number of processes marked as malicious and number of malicious processes marked as malicious. This is reflected in the values for FPR, TPR and ACC. The highest TPR, 0.917, is reached by using the 75% quantile. However it also has the highest FPR (0.161) and lowest ACC, as 24 benign processes are marked as malicious. A FPR of 0.161 means that 16.1% of the non-malicious processes are wrongly marked as malicious.

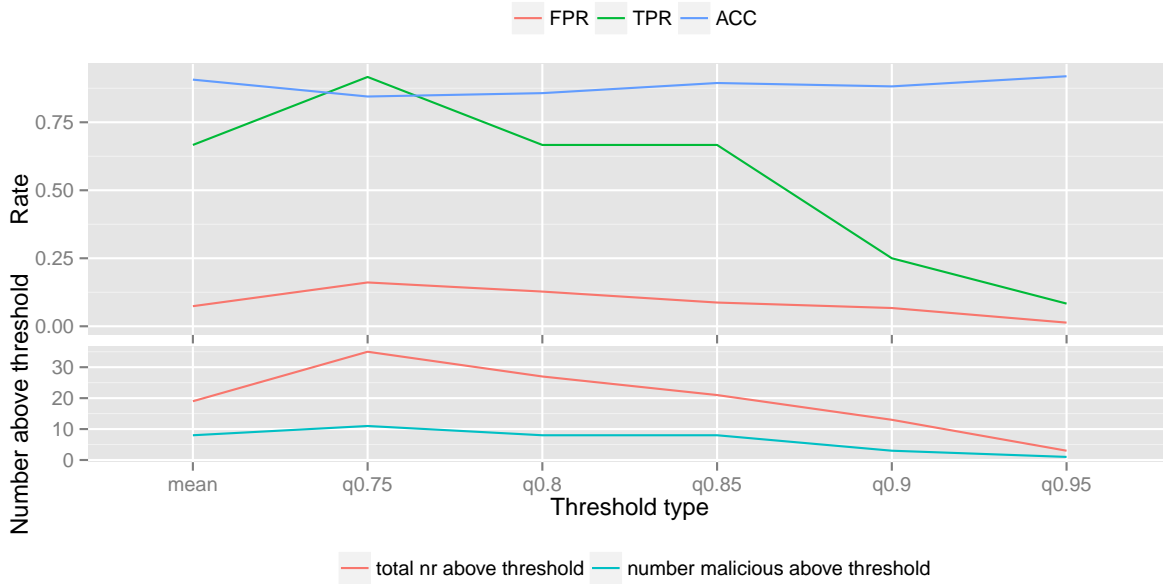


Figure 7.1: Plot of the ACC, FPR and TPR of banking malware

This high value of wrongly marked benign processes might be the consequence of the short running times for generating the malware datasets. The effect of this short data generating period needs to be tested in future research.

Using higher thresholds values gives a higher ACC, a lower FPR but the TPR will become lower as well. Using the highest threshold value only three processes will be marked malicious, of which one process is really malicious and giving FPR of 0.013, TPR of 0.083 and an ACC of 0.919.

From the data shown, it can be concluded that using algorithm 1 is capable to detect the banking malware. Even with the highest threshold value, at least one of the malicious processes from the banking malware is marked as malicious and only two benign processes are marked as malicious.

In chapter 6, ranking of the malicious marked processes was explained. Table I.1 in appendix I.1 shows these processes. The lines in the table represent the border for a threshold type. So all process above the first dark colored line are marked as malicious using the 95% quantile threshold value. In addition the processes which are really malicious are indicated.

Table 7.2: The five highest ranked benign processes for banking malware using algorithm 1.

unique ids	distance sum	malicious
4	23.489	No
135	21.236	No
27	5.383	No
119	4.587	No
143	4.584	No

Analyzing table I.1 we can see the first two processes marked as malicious, process id 4 and 135, are benign processes. These two processes are already discussed in 5.1.5 and 5.2.2. As stated the process with process id 4 has a shorter running time in the malware datasets, however performing the same amount of actions, therefore the process activities per second are higher. Although it is a

benign process marked as malicious, it proofs that using this comparison method the deviation from the known behavior is detected. However it also shows that certain processes perform a certain number of activities and different running times have an impact on the process activities per second.

The executable of process 135 can only be found in the malware datasets and is a software inventory service from Microsoft.

A short analysis was done on the top five of the benign processes marked as malicious and is presented in table I.2. As we can see all the processes, except process 135, were present in all the clean datasets.

Table 7.3 shows the processes from the same executable as process 27 in the banking malware. As can be seen, the process activities differ significantly especially on the registry activities. Here again if we calculate the normalized activities per second back to normalized activities we see that the range between the values decreases. This proofs again that some of the processes perform a certain number of activities and the running time influences the process activities per second. This is something that should be researched in more depth in future work, which will be discussed in section 8.2.

A short analysis on the processes belonging to the executable of the process 119 in the banking malware dataset, we can conclude again that the running time is of influence on the process activities. Moreover there is a difference noticeable between the processes belonging to the malware dataset and the processes of the clean datasets. Calculating the normalized values instead of the normalized activities per second the malware datasets activities are close to each other, whilst the clean datasets have a little higher normalized value, see table 7.5. As mentioned earlier these difference might be forthcoming from the fact that the clean data was collected on a different system than the malicious datasets. In future research the collection of both datasets on the same machine will take this shortcoming away.

Table 7.3: Showing the processes from other datasets from the same executable as process 27 from the banking malware

filesys- tem	reg- istry	process create	thread create	module load	ob	df	depth	fit clus- ter	running time
0.467	1.017	0	0	0.418	0.119	malware bank	4	6	0.062
0.267	0.582	0	0	0.239	0.136	malware rat session2	4	3	0.109
0.004	0.007	0	0	0.003	0.001	win8 1604 avond	4	3	9.203
0.088	0.152	0	0	0.067	0.017	win8 1604	4	3	0.437
0.622	1.397	0	0	0.557	0.158	win8 1704	4	6	0.047
0.128	0.142	0	0	0.063	0.016	win8 1804	4	3	0.469

As stated in the beginning of the chapter the malware we analyzed did only occur in a specific part of the process tree. In figure 7.2 the process tree of the banking malware is shown, with all the process marked malicious using the 75% quantile threshold having a red label. Analyzing the tree shows us that the malicious marked processes are present throughout the whole tree. However in chapter 5.1.5 it is shown that all malicious processes take place in a specific part of the tree at the left.

By assuming that all malware will be present in the left part of the tree we could eliminate a number of processes and would only leave us with the following processes 4, 157, 27, 13, 159, 146, 134, 111, 120, 118, 116, 112, 117, 121, 122, 126, 125 and 124. This brings down the number of processes that need to be checked by the security officer to 18 processes instead of 35.

If the security officer checked the marked processes and found the marked malware processes the tree can also be used to find the remaining not marked malicious malware processes.

However we need to emphasize that the assumption used here is based on three different malware

Table 7.4: Showing a selection the processes from other datasets from the same executable as process 119 from the banking malware

filesys- tem	reg- istry	process create	thread create	module load	ob	df	depth	fit clus- ter	running time
1.295	1.122	0	0.454	0.456	0.475	malware bank	6	6	0.172
1.737	1.543	0	0.624	0.627	0.654	malware bank	6	7	0.125
0.088	0.078	0	0.032	0.032	0.033	malware bank	6	3	2.472
1.736	1.542	0	0.624	0.627	0.654	malware bank	6	7	0.125
1.455	1.301	0	0.479	0.491	0.547	malware rat session1	6	7	0.163
1.584	1.377	0	0.569	0.572	0.651	malware rat session1	6	7	0.137
0.419	0.356	0	0.147	0.147	0.154	malware rat session2	6	6	0.531
0.128	0.079	0	0.133	0.043	0.127	malware rat session2	6	3	2.053
0.141	0.119	0	0.049	0.050	0.052	malware zeus session1	6	3	1.579
0.502	0.452	0	0.166	0.171	0.174	malware zeus session1	6	6	0.469
0.162	0.160	0	0.152	0.051	0.145	malware zeus session1	6	3	1.797
1.736	1.511	0	0.624	0.627	0.654	malware zeus session1	6	7	0.125
0.066	0.057	0	0.024	0.024	0.025	malware zeus session2	6	3	3.312
0.065	0.056	0	0.023	0.023	0.024	malware zeus session2	6	3	3.374
2.145	3.119	0	0.454	0.484	0.476	win8 1604 avond	6	4	0.172
0.338	0.437	0	0.306	0.077	0.274	win8 1604 avond	6	6	1.656
0.194	0.308	0	0.232	0.056	0.211	win8 1604 avond	6	3	2.186
0.983	1.427	0	0.208	0.222	0.238	win8 1604	6	6	0.375
0.238	0.372	0	0.280	0.068	0.251	win8 1604	6	6	1.809
0.243	0.387	0	0.292	0.070	0.261	win8 1604	6	6	1.738
0.241	0.382	0	0.288	0.070	0.258	win8 1604	6	6	1.759
0.236	0.376	0	0.283	0.068	0.253	win8 1604	6	6	1.790
0.658	0.952	0	0.139	0.148	0.145	win8 1704	6	6	0.562
0.608	0.879	0	0.128	0.137	0.134	win8 1804	6	6	0.609

Table 7.5: Calculate normalized values for the process activities for the same executable as process 119 in the banking malware set

filesystem	registry	process create	thread create	module load	ob	df	running time
0.223	0.193	0	0.078	0.078	0.082	malware bank	0.172
0.217	0.193	0	0.078	0.078	0.082	malware bank	0.125
0.217	0.193	0	0.078	0.078	0.082	malware bank	2.472
0.217	0.193	0	0.078	0.078	0.082	malware bank	0.125
0.237	0.212	0	0.078	0.080	0.089	malware rat session1	0.163
0.217	0.189	0	0.078	0.078	0.089	malware rat session1	0.137
0.223	0.189	0	0.078	0.078	0.082	malware rat session2	0.531
0.263	0.162	0	0.273	0.088	0.260	malware rat session2	2.053
0.223	0.189	0	0.078	0.078	0.082	malware zeus session1	1.579
0.235	0.212	0	0.078	0.080	0.082	malware zeus session1	0.469
0.292	0.287	0	0.273	0.091	0.260	malware zeus session1	1.797
0.217	0.189	0	0.078	0.078	0.082	malware zeus session1	0.125
0.219	0.189	0	0.078	0.078	0.082	malware zeus session2	3.312
0.221	0.189	0	0.078	0.078	0.082	malware zeus session2	3.374
0.368	0.536	0	0.078	0.083	0.082	win8 1604 avond	0.172
0.560	0.723	0	0.507	0.127	0.453	win8 1604 avond	1.656
0.423	0.674	0	0.507	0.122	0.461	win8 1604 avond	2.186
0.368	0.535	0	0.078	0.083	0.089	win8 1604	0.375
0.430	0.673	0	0.507	0.122	0.453	win8 1604	1.809
0.423	0.673	0	0.507	0.122	0.453	win8 1604	1.752
0.423	0.673	0	0.507	0.122	0.453	win8 1604	1.545
0.423	0.673	0	0.507	0.122	0.453	win8 1604	2.106
0.423	0.673	0	0.507	0.122	0.453	win8 1604	1.637
0.423	0.673	0	0.507	0.122	0.453	win8 1604	1.709
0.427	0.673	0	0.507	0.122	0.453	win8 1604	1.931
0.423	0.673	0	0.507	0.122	0.453	win8 1604	2.002
0.423	0.673	0	0.507	0.122	0.453	win8 1604	2.058
0.423	0.673	0	0.507	0.122	0.453	win8 1604	1.786
0.423	0.673	0	0.507	0.122	0.453	win8 1604	1.853
0.423	0.673	0	0.507	0.122	0.453	win8 1604	1.695
0.423	0.673	0	0.507	0.122	0.453	win8 1604	1.887
0.423	0.673	0	0.507	0.122	0.453	win8 1604	1.856
0.423	0.673	0	0.507	0.122	0.453	win8 1604	2.118
0.423	0.673	0	0.507	0.122	0.453	win8 1604	2.050
0.423	0.673	0	0.507	0.122	0.453	win8 1604	1.738
0.423	0.673	0	0.507	0.122	0.453	win8 1604	1.759
0.423	0.673	0	0.507	0.122	0.453	win8 1604	1.790
0.370	0.536	0	0.078	0.083	0.082	win8 1704	0.562
0.370	0.536	0	0.078	0.083	0.082	win8 1804	0.609

samples tested for a short period of time, as already mentioned in section 4.2.2 this assumption must be checked in future research. Furthermore using this assumption creates a dangerous trade-off as the security officer will now use known malicious behavior, always present in a certain part of the tree, to investigate malicious marked processes.

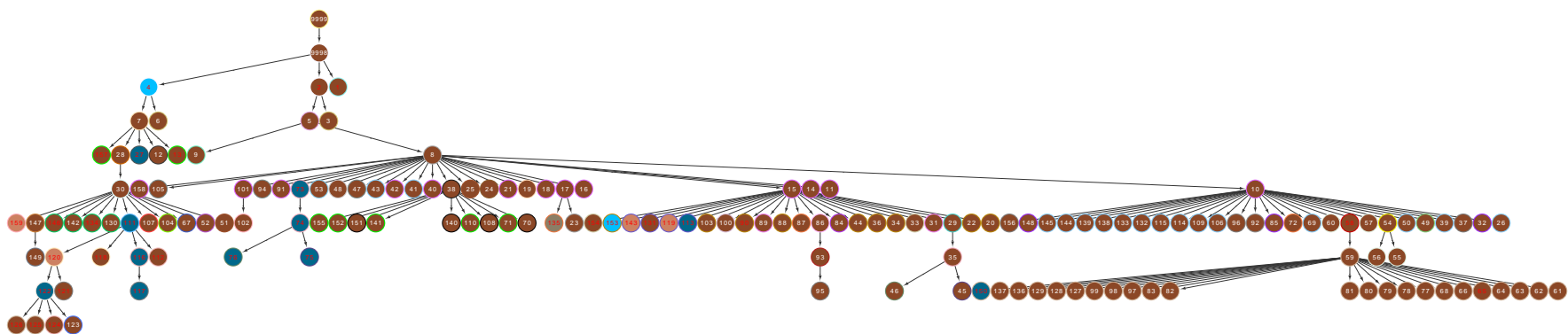


Figure 7.2: The process tree of the banking malware set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.

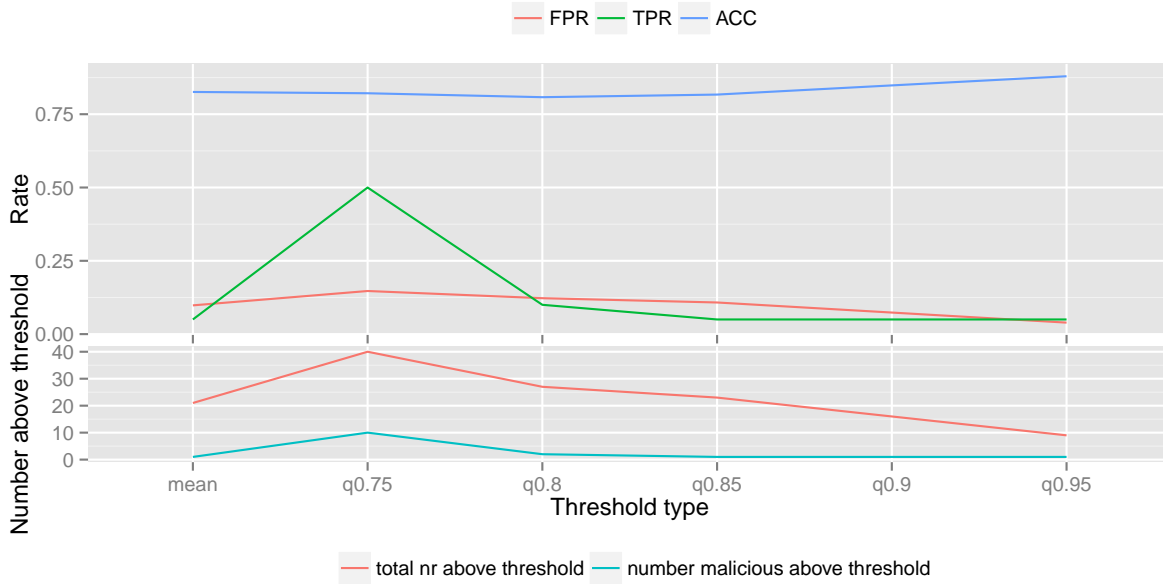


Figure 7.3: Plot of the ACC, FPR and TPR of rat session 1

7.2.1.2 Algorithm 1: Rat session 1

In the rat session 1 dataset 20 processes from the 224 processes are malicious processes, which were identified in section 4.6.2. The outcome of using algorithm 1 on the rat session 1 dataset and using the different threshold values, is shown in table 7.6. Here again malicious processes are correctly marked as malicious. However the highest TPR, 0.5, is significantly lower than seen in the previous section. And again in this case the number of benign processes marked as malicious, is quite high, namely 14.7%. However as concluded before this could be brought down by using longer time periods to collect malware datasets.

Table 7.6: Outcome using different threshold values for rat1 malware using method1

threshold type	total nr above threshold	number malicious above threshold	FPR	TPR	ACC
mean	21	1	0.098	0.050	0.826
q0.75	40	10	0.147	0.500	0.821
q0.8	27	2	0.123	0.100	0.808
q0.85	23	1	0.108	0.050	0.817
q0.9	16	1	0.074	0.050	0.848
q0.95	9	1	0.039	0.050	0.879

Analyzing table I.3, we can see that the non malicious process with process id 4 is again ranked highest. Normally an equal process id between sets, is not automatically the same executable being executed. However, as mentioned in previous chapters, process id is in all datasets, clean and malware, the same process. The reason this process is again ranked highest is already discussed in the previous section and section 5.1.5.

The second highest process has id 109, and its executable is present in all clean datasets and in

Table 7.7: The five highest ranked benign processes for rat session 1 using algorithm 1.

unique ids	distance sum	malicious
4	18.626	No
109	10.674	No
111	8.652	No
95	7.347	No
204	6.634	No

Zeus session 1 and Zeus session 2 I.4. Taking a closer look at the process activities of this executable, it can be concluded that a comparable situation is present as with previously analyzed processes 7.8, and strengthens the previously mention recommendations for future work. Running the collecting of clean and malware datasets on the same host and investigate the effect of how the normalized activity values per second are calculated.

Analyzing process 111 we can conclude the same as for process 109. However process 95 and 204, which have are processes from the same executable, look different if we analyze their normalized values, table 7.10 and 7.9, and is only present in the malware datasets (table I.4). The fact that the executable is only present in the malware datasets again shows that collecting both data types on the same system could prevent these type of problems. Thereby the tables show that this process does behave differently every time it is started, running times differ greatly and the recalculated normalized values show a bigger range.

From the information presented above we can conclude that in the top of the ranked malicious processes their can be processes from the same executable. This information can be incorporated in the calculation of the ranking or the list of malicious processes. However due to time constraints this is not explored during this master thesis and will be given as a recommendation for future work.

Analyzing the process tree for the rat session 1 data the following processes are in the left part of the tree: 4, 222, 191, 187, 167, 186, 185, 183, 181, 179, 177, 175, 173, 171. This brings down the number of process from 40 to 14.

Table 7.8: The calculated normalized values of processes with the same executable as process 109 in the rat session 1 data

filesys- tem	reg- istry	pro- cess.create	thread.create	mod- ule.load	ob	df	running time
0.069	0.018	0.312	0.078	0.038	0.082	malware rat session1	0.281
0.151	0.042	0	0.078	0.057	0.104	malware rat session1	1.266
0.071	0.018	0.312	0.078	0.038	0.082	malware rat session1	0.060
0.153	0.042	0	0.078	0.057	0.097	malware rat session1	0.040
0.069	0.012	0.156	0.039	0.038	0.052	malware rat session1	3.150
0.071	0.018	0.312	0.078	0.038	0.082	malware zeus session1	0.352
0.153	0.042	0	0.078	0.057	0.104	malware zeus session1	21.375
0.069	0.018	0.312	0.078	0.038	0.082	malware zeus session1	0.047
0.151	0.042	0	0.078	0.057	0.097	malware zeus session1	0.031
0.071	0.018	0.312	0.078	0.038	0.082	malware zeus session2	0.266
0.155	0.042	0	0.078	0.057	0.111	malware zeus session2	52.875
0.071	0.018	0.312	0.078	0.038	0.082	malware zeus session2	0.047
0.151	0.042	0	0.078	0.057	0.097	malware zeus session2	0.031
0.181	0.454	0.156	0.117	0.055	0.104	win8 1604 avond	0.776
0.181	0.454	0.156	0.117	0.055	0.104	win8 1604 avond	0.151
0.181	0.454	0.156	0.117	0.055	0.104	win8 1604	0.172
0.181	0.454	0.156	0.117	0.055	0.104	win8 1604	0.141
0.182	0.454	0.156	0.117	0.055	0.104	win8 1704	0.203
0.181	0.454	0.156	0.117	0.055	0.104	win8 1704	0.156
0.181	0.454	0.156	0.117	0.055	0.104	win8 1804	0.891
0.181	0.454	0.156	0.117	0.055	0.104	win8 1804	0.141

Table 7.9: Summary of the process activities of the processes with the same executable as process 95 and 204 in the rat session 1 dataset

Statistic	N	Mean	St. Dev.	Min	Max
filesystem	47	0.253	0.226	0.064	0.998
registry	47	0.042	0.030	0.017	0.197
process.create	47	0.007	0.046	0.000	0.312
thread.create	47	0.264	0.174	0.039	0.780
module.load	47	0.053	0.014	0.029	0.085
ob	47	0.284	0.175	0.067	0.780
running time	47	68.592	101.511	0.078	370.360

Table 7.10: Showing the processes from the same executable as process 95 and 204 in the rat session 1 dataset

filesystem	registry	process create	thread create	module load	ob	df	running time
0.064	0.017	0	0.039	0.029	0.067	malware bank	4.984
0.102	0.019	0	0.078	0.044	0.097	malware bank	3.219
0.228	0.068	0	0.351	0.080	0.357	malware bank	32.701
0.131	0.040	0	0.156	0.051	0.178	malware bank	27.938
0.106	0.020	0	0.078	0.046	0.111	malware bank	16.438
0.653	0.049	0	0.429	0.062	0.505	malware bank	243.356
0.111	0.028	0	0.273	0.047	0.275	malware bank	0.818
0.562	0.048	0	0.429	0.062	0.475	malware bank	146.644
0.111	0.028	0	0.273	0.047	0.275	malware bank	0.094
0.069	0.018	0	0.078	0.029	0.097	malware rat session1	0.857
0.102	0.019	0	0.078	0.044	0.097	malware rat session1	0.479
0.106	0.020	0	0.078	0.046	0.097	malware rat session1	18.494
0.133	0.040	0	0.156	0.051	0.178	malware rat session1	19.747
0.193	0.052	0	0.273	0.072	0.290	malware rat session1	15.719
0.111	0.028	0	0.273	0.047	0.275	malware rat session1	0.084
0.998	0.060	0	0.429	0.062	0.475	malware rat session1	129.554
0.560	0.048	0	0.429	0.062	0.461	malware rat session1	123.551
0.589	0.049	0	0.429	0.062	0.468	malware rat session1	125.213
0.111	0.028	0	0.273	0.047	0.275	malware rat session1	0.078
0.069	0.018	0	0.078	0.029	0.097	malware rat session2	0.297
0.102	0.019	0	0.078	0.044	0.097	malware rat session2	0.094
0.131	0.040	0	0.156	0.051	0.178	malware rat session2	14.025
0.106	0.020	0	0.078	0.046	0.097	malware rat session2	11.141
0.193	0.052	0	0.273	0.072	0.282	malware rat session2	10.547
0.111	0.028	0	0.273	0.047	0.275	malware rat session2	0.996
0.584	0.048	0	0.429	0.062	0.468	malware rat session2	330.363
0.219	0.073	0	0.546	0.085	0.550	malware rat session2	61.313
0.266	0.078	0.312	0.780	0.054	0.780	malware rat session2	370.360
0.241	0.089	0	0.351	0.059	0.364	malware rat session2	43.297
0.686	0.197	0	0.624	0.070	0.624	malware rat session2	366.345
0.440	0.048	0	0.429	0.062	0.431	malware rat session2	246.555
0.069	0.018	0	0.078	0.029	0.097	malware zeus session1	1.984
0.102	0.019	0	0.078	0.044	0.097	malware zeus session1	0.344
0.232	0.068	0	0.351	0.080	0.386	malware zeus session1	36.109
0.133	0.040	0	0.156	0.051	0.178	malware zeus session1	31.723
0.106	0.020	0	0.078	0.046	0.097	malware zeus session1	6.687
0.569	0.048	0	0.429	0.062	0.475	malware zeus session1	137.233
0.111	0.028	0	0.273	0.047	0.275	malware zeus session1	0.137
0.443	0.048	0	0.390	0.062	0.394	malware zeus session1	87.280
0.069	0.018	0	0.078	0.029	0.097	malware zeus session2	0.695
0.102	0.019	0	0.078	0.044	0.097	malware zeus session2	0.351
0.224	0.068	0	0.351	0.080	0.379	malware zeus session2	78.625
0.135	0.040	0	0.156	0.051	0.186	malware zeus session2	80.500
0.108	0.020	0	0.078	0.046	0.097	malware zeus session2	28.922
0.648	0.049	0	0.429	0.062	0.468	malware Zeus session2	240.576
0.111	0.028	0	0.273	0.047	0.275	malware Zeus session2	0.899
0.560	0.048	0	0.429	0.062	0.468	malware Zeus session2	126.438

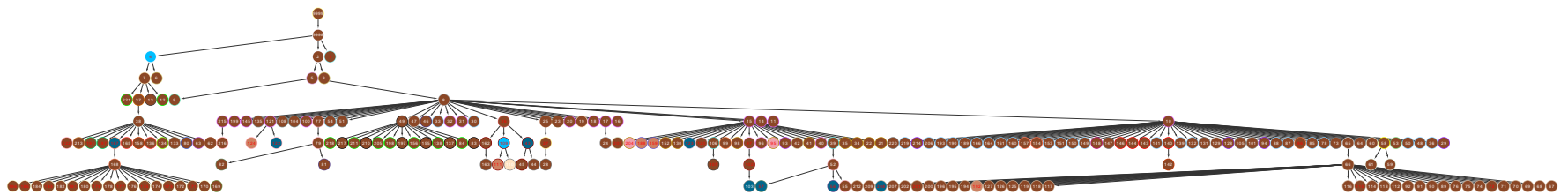


Figure 7.4: The process tree of the rat session 1 set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.

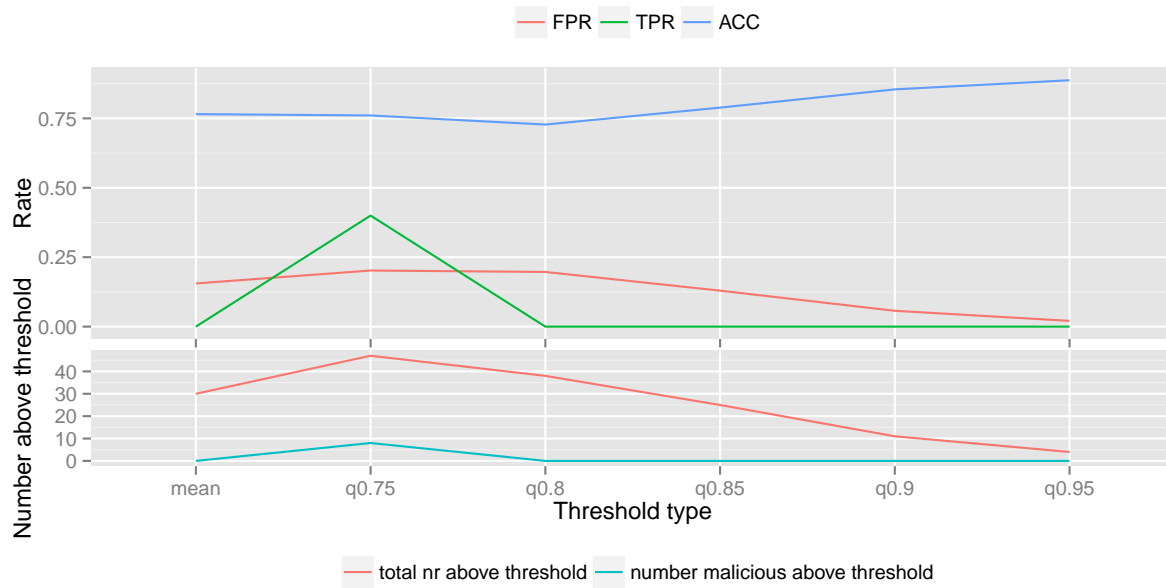


Figure 7.5: Plot of the ACC, FPR and TPR of rat session 2

7.2.1.3 Algorithm 1: Rat session 2

Table 7.11: Outcome using different threshold values for rat2 malware using method1

threshold type	total nr above threshold	number malicious above threshold	FPR	TPR	ACC
mean	30	0	0.155	0	0.765
q0.75	47	8	0.202	0.400	0.761
q0.8	38	0	0.197	0	0.728
q0.85	25	0	0.130	0	0.789
q0.9	11	0	0.057	0	0.854
q0.95	4	0	0.021	0	0.887

Table 7.12: The five highest ranked benign processes for rat session 2 using algorithm 1.

unique ids	distance sum	malicious
4	20.393	No
111	8.395	No
161	4.538	No
206	4.142	No
194	3.791	No

The rat session 2 dataset consists of twenty malicious processes out of the 213 process in total. In table 7.6 and figure 7.5 an overview is given of the outcome of algorithm 1. Although we could spot

the malicious processes in the process tree, discussed in chapter/section 5.1.5.1, the algorithm was only possible to mark eight malicious processes using the 75% quantile.

Apparently the behavior of the malicious processes does not deviate enough to be detected. In table 7.13 a summary is given of the process activities of the malicious processes in the rat session 2 dataset. If we compare the activities summary to that of the rat session 1, see table 7.14, it can be concluded that the activities of malicious processes in rat session 1 dataset is higher.

There could be several reasons for this, in the rat session 1 set the malware was installed, this means filesystem and registry actions are executed to install itself and make sure it is booted after reboot. After reboot the malware only needs to be started and waiting for orders, as discussed in chapter 4. As we did not monitor the internet connection we do not know if it received any commands from the command and control server and thus it might have been idling, waiting for commands. The fact it might not have received any commands could be because of the short running time of the malware dataset collection. In future research a better setup should be used to collect all the actions performed by the malware.

Table 7.13: Summary of the process activities of the malicious processes of the rat session 2 dataset

Statistic	N	Mean	St. Dev.	Min	Max
filesystem	20	0.006	0.026	0.000	0.115
registry	20	0.001	0.002	0.000	0.010
process.create	20	0.004	0.016	0.000	0.071
thread.create	20	0.001	0.004	0.000	0.018
module.load	20	0.023	0.018	0.00004	0.045
ob	20	0.033	0.028	0.00002	0.068

Table 7.14: Summary of the process activities of the malicious processes of the rat session 1 dataset

Statistic	N	Mean	St. Dev.	Min	Max
filesystem	20	0.091	0.408	0.000	1.824
registry	20	0.008	0.037	0.000	0.165
process.create	20	0.062	0.277	0.000	1.238
thread.create	20	0.016	0.069	0.000	0.310
module.load	20	0.047	0.098	0.00004	0.453
ob	20	0.063	0.114	0.00001	0.531

In analyzing the top ranked benign processes, process 4 is again ranked as first. Hereby the same applies as in the previously evaluated malware sets. Process with id 111 is again only present in the malware datasets.

Processes 161 and 206 are only present in this dataset. Both executables are started two times during the data collection, but show differences in activity, see tables 7.15 and 7.16. It is not strange that these processes are marked as malicious, as they do not appear in any other dataset. This provides proof that unknown processes can be detected. However due to the shortcomings of recording the malicious processes it could be the case that these processes are malicious, coming from the rat malware, but could not be found using the log sensor information we used.

The last process from the top five ranking is 194. This process is available in the rat session 2 dataset and in the clean 1604 and 1704 dataset. Here again the process has a little higher activity in comparison to the clean datasets, which probably is caused by using different systems for collecting the malware and clean datasets.

Table 7.15: Process information from the executable belonging to process 161 of the rat session 2 malware dataset

filesystem	registry	process.create	thread.create	module.load	ob	unique ids	depth	fit.cluster
1.049	1.465	0	0	0.678	0.712	161	7	6
0.004	0.005	0	0.004	0.003	0.008	162	7	3

Table 7.16: Process information from the executable belonging to process 206 of the rat session 2 malware dataset

filesystem	registry	process.create	thread.create	module.load	ob	unique ids	depth	fit.cluster
1.910	0.201	0	0	0.465	0.043	206	8	6
0.443	0.126	0	0.063	0.078	0.084	210	6	3

Using the process tree, figure 7.6, to eliminate processes for further analysis by the security officer, the following processes will be left: 4, 34, 202, 69, 60, 85, 84, 82, 80, 76, 74 and 72. This means only 12 of the 47 malicious marked processes need to be investigated thoroughly.

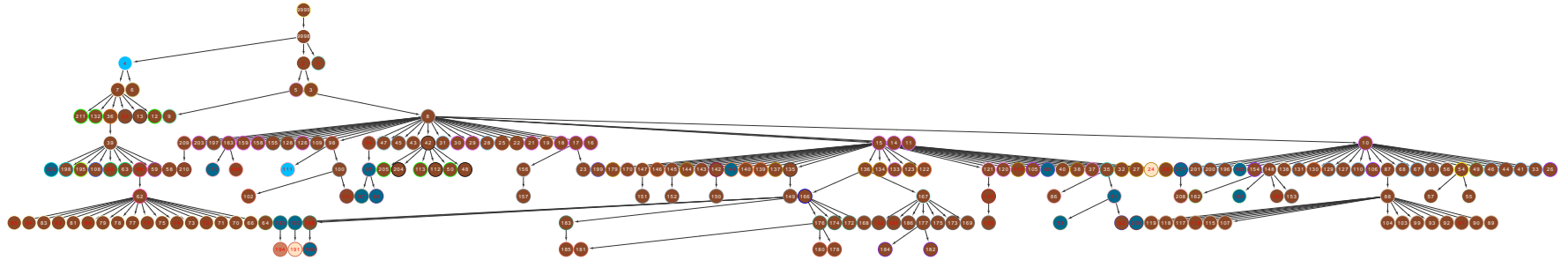


Figure 7.6: The process tree of the rat session 2 dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.

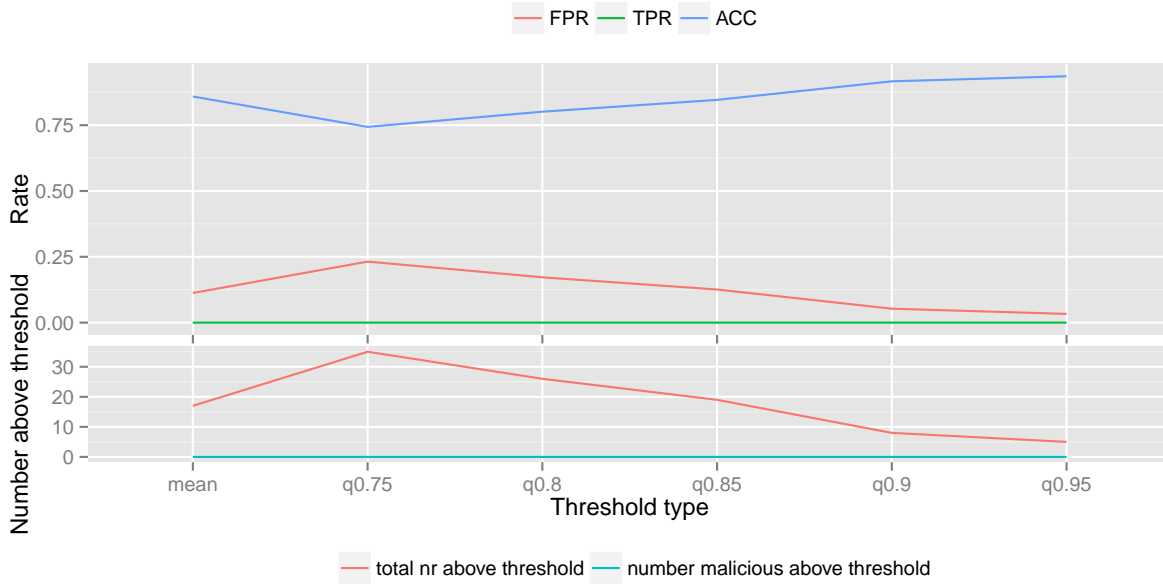


Figure 7.7: Plot of the ACC, FPR and TPR of Zeus session 1

7.2.1.4 Algorithm 1: Zeus session 1

The Zeus session 1 data set consists of 156 processes of which five are malicious. In table 7.17 the summary of the outcome of algorithm 1 is given. However none of the malicious processes is marked as malicious, giving a TPR of 0 for all threshold values used.

Table 7.17: Outcome using different threshold values for zeus1 malware using method1

threshold type	total nr above threshold	number malicious above threshold	FPR	TPR	ACC
mean	17	0	0.113	0	0.859
q0.75	35	0	0.232	0	0.744
q0.8	26	0	0.172	0	0.801
q0.85	19	0	0.126	0	0.846
q0.9	8	0	0.053	0	0.917
q0.95	5	0	0.033	0	0.936

In table 7.18 and 7.19 the process activities of the malicious processes in the Zeus session 1 dataset and the clean dataset 1604 are summarized. If we compare these values, we see that the activity of the malicious processes in the Zeus dataset is quite low and should probably find a match with a process in the clean dataset with a low distance. Apparently installing the Zeus malware shows a low amount of activities per second. Again due to shortcomings in our test setup we do not know if the Zeus malware did receive any instructions from the malware's owner.

Again the process with id 4 is the highest ranked process. The next two process in the top five are processes 111 and 114, which are the same executable as processes 109 and 111 in the rat session 1 malware dataset. The probable causes for being marked as malicious are data is collected on a different host and therefor the running times differ to perform the same number of activities.

Table 7.18: A summary of the process activities of the malicious processes of the Zeus session 1 malware dataset

Statistic	N	Mean	St. Dev.	Min	Pctl(25)	Median	Pctl(75)	Max
filesystem	5	0.006	0.004	0.002	0.003	0.005	0.007	0.012
registry	5	0.002	0.001	0.001	0.001	0.002	0.003	0.004
process.create	5	0.003	0.003	0.000	0.000	0.004	0.005	0.006
thread.create	5	0.006	0.004	0.002	0.003	0.006	0.006	0.012
module.load	5	0.001	0.001	0.0004	0.001	0.002	0.002	0.002
ob	5	0.007	0.005	0.002	0.003	0.006	0.006	0.015

Table 7.19: Summary of the process activity of the clean 1604 dataset for comparison.

Statistic	N	Mean	St. Dev.	Min	Pctl(25)	Median	Pctl(75)	Max
filesystem	925	0.328	0.607	0.000	0.002	0.024	0.304	4.677
registry	925	0.143	0.400	0.000	0.001	0.007	0.138	6.981
process.create	925	0.083	0.362	0.000	0.000	0.000	0.000	5.000
thread.create	925	0.225	0.497	0.000	0.001	0.024	0.219	5.000
module.load	925	0.122	0.324	0.000	0.001	0.007	0.112	2.828
ob	925	0.293	0.657	0.000	0.003	0.031	0.264	5.485

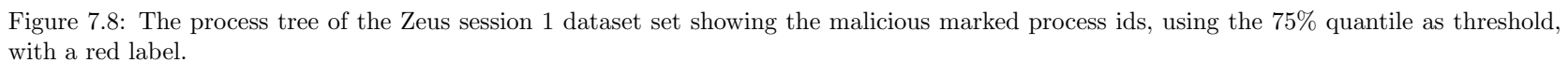
Table 7.20: The five highest ranked benign processes for zeus session 1 using algorithm 1.

unique ids	distance sum	malicious
4	23.482	No
111	14.668	No
113	12.345	No
147	4.503	No
107	4.418	No

Process 147 is the same executable as process 119 in the banking malware and here again is the running time and the number of activities performed of influence. In future research it is important to find a method which takes into account the fact that some processes only perform a certain number of actions and the influence of the running time. Although this can probably be eliminated by collecting all data on the same machine these type of process differences are important to keep in mind.

The last process, 107, is the same executable as the analyzed process 95 in the rat session 1 data, and is only present in malware datasets.

Although none of the malicious processes were marked as malicious the process tree, figure 7.8 can still be used to shrink the number of processes the security officer needs to investigate. In using the process tree only six processes (4, 152, 12, 150, 134, 121, 117) instead of 35 needs to be checked.



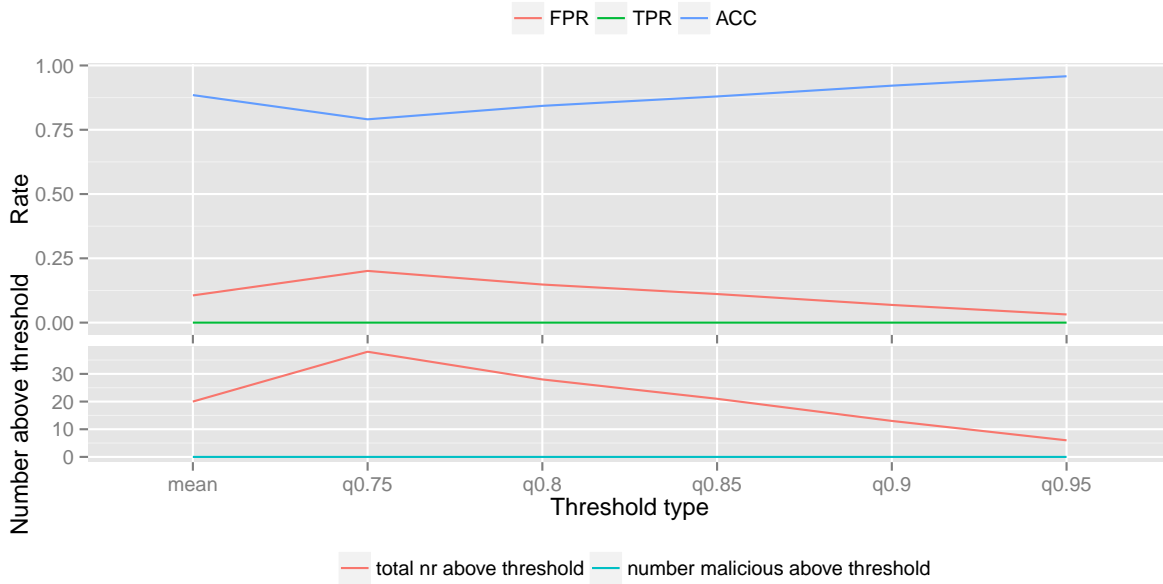


Figure 7.9: Plot of the ACC, FPR and TPR of Zeus session 2

7.2.1.5 Algorithm 1: Zeus session 2

The dataset Zeus session 2 contains 191 processes from which two are marked as malicious. Here again the TPR for all comparisons is zero. It might be that the Zeus malware was only started to listen to receive commands. However as we did not track the network traffic, we do not know if the malware did receive any commands.

Table 7.21: Outcome using different threshold values for zeus2 malware using method1

threshold type	total nr above threshold	number malicious above threshold	FPR	TPR	ACC
mean	20	0	0.106	0	0.885
q0.75	38	0	0.201	0	0.791
q0.8	28	0	0.148	0	0.843
q0.85	21	0	0.111	0	0.880
q0.9	13	0	0.069	0	0.921
q0.95	6	0	0.032	0	0.958

For the two malicious processes of Zeus session 2 the same beholds as for the malicious processes of the Zeus malware in session 1 as we can see from table 7.22 and table 7.19 in the previous chapter.

The top three of the ranked non-malicious processes were already discussed in previous sections. This leaves only processes 116 and 130 and are present in all datasets I.10. If the summary of the process activity of the two processes, shown in table 7.24, are compared to the summary of all the same executables, table 7.25, it can be concluded that the process activity per second of the processes from the Zeus malware are high. For this reason the distance between the matched processes is high as well.

Analyzing the process tree in figure 7.10, only four processes, namely, 4; 187; 59; 139, will be left

Table 7.22: Summary of the process activities of the zeus session 2 dataset

Statistic	N	Mean	St. Dev.	Min	Pctl(25)	Median	Pctl(75)	Max
filesystem	2	0.004	0.005	0.0004	0.002	0.004	0.006	0.007
registry	2	0.002	0.002	0.0003	0.001	0.002	0.003	0.003
process.create	2	0.004	0.006	0.000	0.002	0.004	0.006	0.008
thread.create	2	0.005	0.005	0.001	0.003	0.005	0.007	0.008
module.load	2	0.002	0.003	0.0001	0.001	0.002	0.003	0.004
ob	2	0.010	0.001	0.010	0.010	0.010	0.011	0.011

Table 7.23: The five highest ranked benign processes for zeus session 2 using algorithm 1.

unique ids	distance sum	malicious
4	23.482	No
112	14.718	No
114	12.306	No
116	5.340	No
130	5.331	No

Table 7.24: Summary of process activity of process id 116 and 130 of the Zeus session 2 malware set

Statistic	N	Mean	St. Dev.	Min	Pctl(25)	Median	Pctl(75)	Max
filesystem	2	1.754	0.081	1.696	1.725	1.754	1.783	1.812
registry	2	0.274	0.013	0.265	0.270	0.274	0.279	0.283
process.create	2	0.000	0.000	0	0	0	0	0
thread.create	2	0.540	0.025	0.522	0.531	0.540	0.548	0.557
module.load	2	0.161	0.007	0.156	0.158	0.161	0.164	0.166
ob	2	0.514	0.024	0.497	0.505	0.514	0.522	0.531

Table 7.25: Summary of the process activities of the executable belonging to process 116 and 130 in the zeus session 2 data from all datasets

Statistic	N	Mean	St. Dev.	Min	Pctl(25)	Median	Pctl(75)	Max
filesystem	492	0.227	0.488	0.000	0.003	0.045	0.181	2.797
registry	492	0.045	0.099	0.000	0.001	0.007	0.033	0.626
process.create	492	0.0001	0.001	0.000	0.000	0.000	0.000	0.018
thread.create	492	0.075	0.182	0.000	0.001	0.013	0.056	1.446
module.load	492	0.042	0.133	0.00000	0.0003	0.004	0.017	1.110
ob	492	0.176	0.591	0.00002	0.002	0.019	0.092	6.428

for further analysis. This brings down the number significantly from 38 to 4. However none of the malicious marked processes is really malicious.

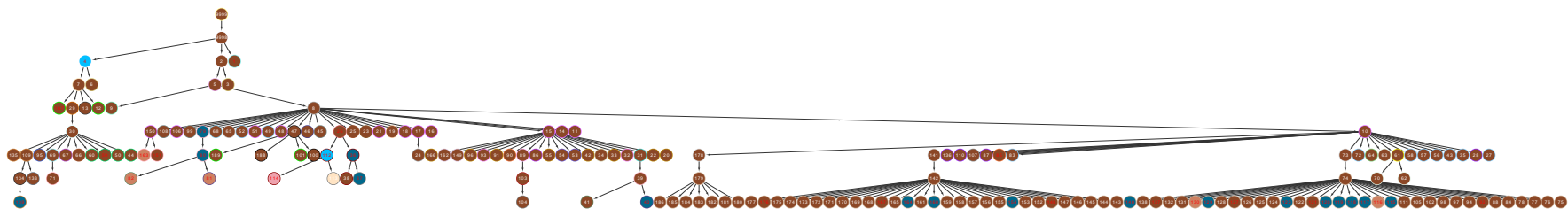


Figure 7.10: The process tree of the Zeus session 2 dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.

7.2.1.6 Conclusion Algorithm 1

In this section we will conclude the evaluation of the algorithm 1, done in the previous five sections. Figure 7.11 gives a graphical presentation of the FPR, TPR and ACC of using algorithm 1 on the five malware sets.

Looking at the graphical overview we can see that using the 75% quantile threshold gives the highest TPR for the banking malware, rat session 1 and rat session 2, where the bank malware the highest TPR. However the downside is that the FPR is going up and the ACC goes down. Disappointingly using algorithm 1, we could not detect any of the malicious processes in the Zeus datasets.

By analyzing the the top five non-malicious ranked processes several shortcomings and recommendations were for future research were found, these will be elaborated in chapter 8. One of the main shortcomings is the fact that a non-malicious process, process id 4, was first ranked in all five rankings. The reason for this is the usage of different systems to collect the data. Nevertheless algorithm 1 showed a positively the ability to detect malware. Although the outcome at the moment is not usable for security officers in enterprise environments it still looks promising.

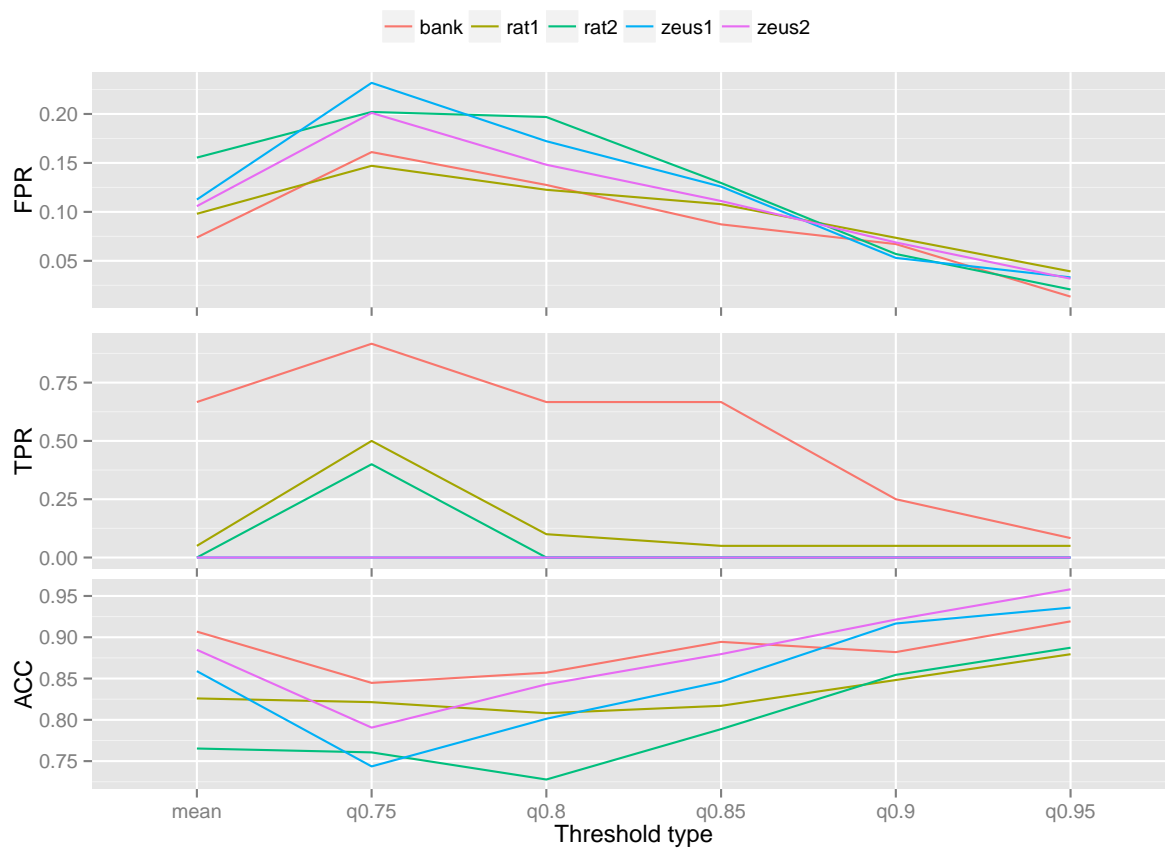


Figure 7.11: Plot of the ACC, FPR and TPR of all malware sets using method 1

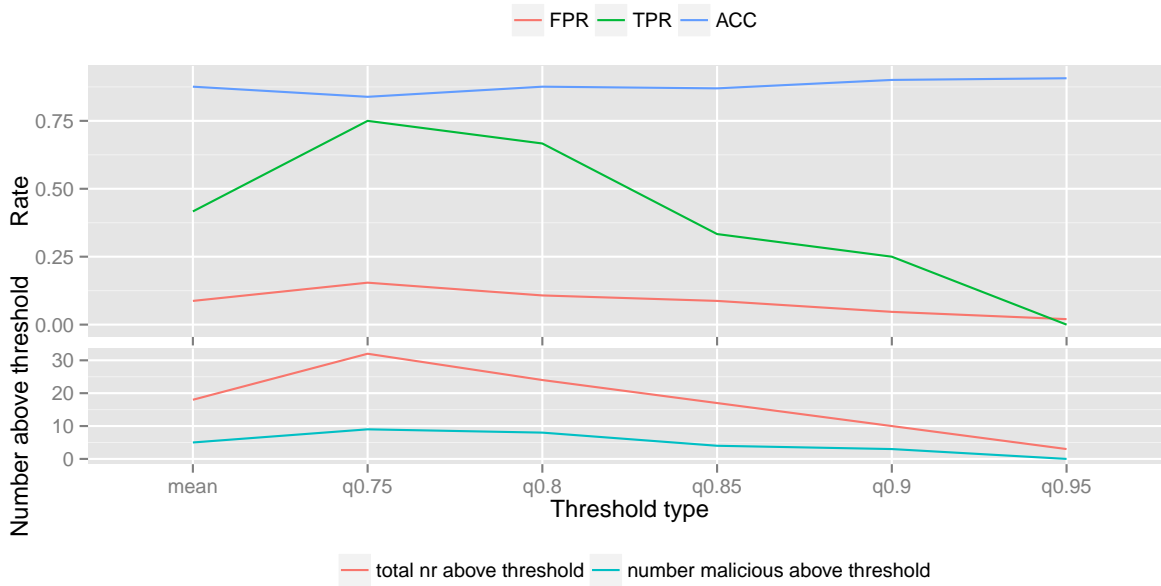


Figure 7.12: Plot of the ACC, FPR and TPR of bank using method 2

7.2.2 Algorithm 2

Algorithm 2 is the strict algorithm. Here again we will follow the steps as described in the evaluation set-up.

7.2.2.1 Algorithm 2: Banking malware

Table 7.26 and 7.12 show the outcome of using algorithm 2 on the banking malware dataset. As can be seen algorithm 2 is able to correctly mark malicious processes, however the FPR is still high. In contrast with algorithm 1, the highest threshold value, 95% quantile, does not mark any of the malicious processes as malicious.

Table 7.26: Outcome using different threshold values for bank malware using method2

threshold type	total nr above threshold	number malicious above threshold	FPR	TPR	ACC
mean	18	5	0.087	0.417	0.876
q0.75	32	9	0.154	0.750	0.839
q0.8	24	8	0.107	0.667	0.876
q0.85	17	4	0.087	0.333	0.870
q0.9	10	3	0.047	0.250	0.901
q0.95	3	0	0.020	0	0.907

In the ranking of the benign processes marked incorrectly, table I.11, two different process showed up in comparison with the top five of algorithm 1. These processes are 159 and 153. As can be seen in table I.12 both processes are only available in the malware datasets. Which probably means these executables are not existing on the system used collecting clean data. As stated during the analysis

Table 7.27: The five highest ranked benign processes for banking malware using algorithm 2.

unique ids	distance sum	malicious
4	32.514	No
135	24.814	No
159	10.431	No
27	9.995	No
153	9.905	No

of algorithm 1 this is caused by the shortcoming that two different systems where used during data collection.

Applying the same technique as used in analyzing algorithm 1 on the process tree, figure 7.13 the following processes remain: 4, 157, 27, 13, 159, 146, 134, 111, 120, 118, 116, 112, 122, 121, 117, 126, 125 and 124. Thus reducing the number of processes from 32 to 18.

Figure 7.13: The process tree of the banking malware dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.

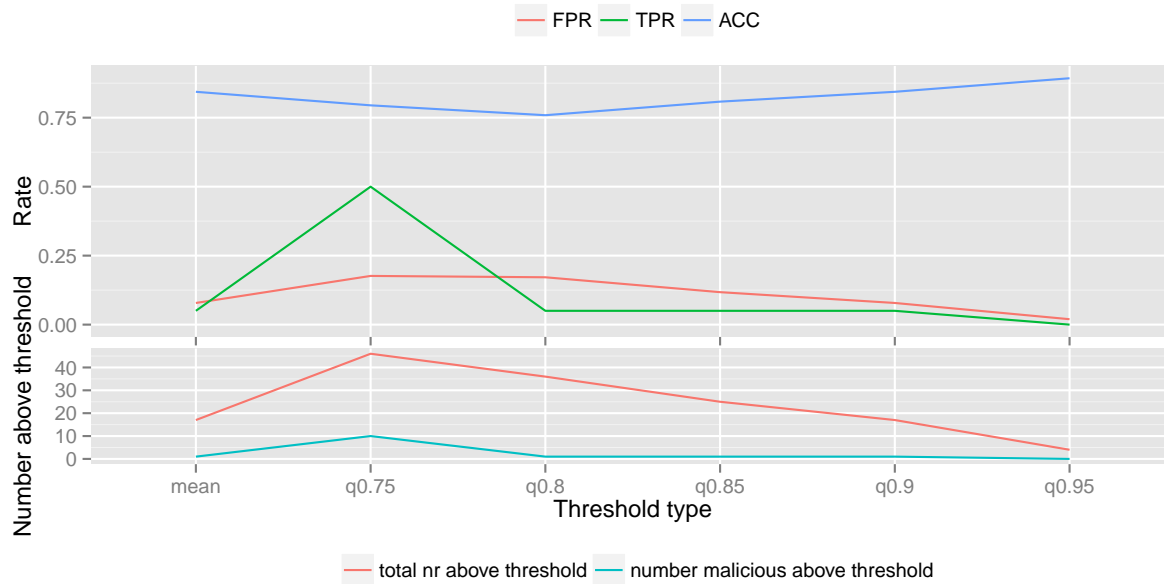


Figure 7.14: Plot of the ACC, FPR and TPR of rat session 1 using method 2

7.2.2.2 Algorithm 2: Rat 1

Looking at the summary of the outcome of using algorithm 2 on the rat session 1 data, table 7.28 and figure 7.14, we see that again at the 75% quantile the most malicious processes are marked correctly. However the other threshold limits marked less malicious processes as being malicious.

Table 7.28: Outcome using different threshold values for rat1 malware using method2

threshold type	total nr above threshold	number malicious above threshold	FPR	TPR	ACC
mean	17	1	0.078	0.050	0.844
q0.75	46	10	0.176	0.500	0.795
q0.8	36	1	0.172	0.050	0.759
q0.85	25	1	0.118	0.050	0.808
q0.9	17	1	0.078	0.050	0.844
q0.95	4	0	0.020	0	0.893

In the top five presented in table I.13 only process is shown which is not already analyzed, namely process 110. The executable of this process can be found in all datasets, as can be seen in table I.14. However here again the process performs a certain set of activities, but due to different running times the normalized activities performed per second is higher than in the clean datasets. If we calculate the normalized values, so not per second we can see they are in the same range as other datasets, see tables 7.30 and 7.31.

The following 14 processes are placed at the left side of the process tree, figure 7.15: 4, 222, 191, 187, 167, 185, 183, 181, 179, 177, 175, 173 and 171. Leaving only 14 of the 64 processes to be analyzed by the security officer.

Table 7.29: The five highest ranked benign processes for rat session 1 using algorithm 2.

unique ids	distance sum	malicious
109	25.014	No
4	24.599	No
111	20.723	No
110	15.170	No
204	11.975	No

Table 7.30: Summary of normalized process activities for process id 110 from the rat session 1

Statistic	N	Mean	St. Dev.	Min	Pctl(25)	Median	Pctl(75)	Max
filesystem	5	0.026	0.010	0.020	0.020	0.022	0.024	0.044
registry	5	0.010	0.0002	0.010	0.010	0.010	0.010	0.010
process.create	5	0.000	0.000	0	0	0	0	0
thread.create	5	0.070	0.017	0.039	0.078	0.078	0.078	0.078
module.load	5	0.020	0.000	0.020	0.020	0.020	0.020	0.020
ob	5	0.068	0.014	0.045	0.067	0.074	0.074	0.082

Table 7.31: Summary of the process activities of all processes from the same executable as process 110 in rat session 1 dataset.

Statistic	N	Mean	St. Dev.	Min	Pctl(25)	Median	Pctl(75)	Max
filesystem	286	0.028	0.014	0.016	0.024	0.024	0.024	0.091
registry	286	0.013	0.003	0.010	0.011	0.011	0.017	0.024
process.create	286	0.001	0.009	0.000	0.000	0.000	0.000	0.156
thread.create	286	0.075	0.011	0.039	0.078	0.078	0.078	0.117
module.load	286	0.021	0.003	0.020	0.020	0.020	0.020	0.034
ob	286	0.076	0.023	0.045	0.074	0.074	0.074	0.327

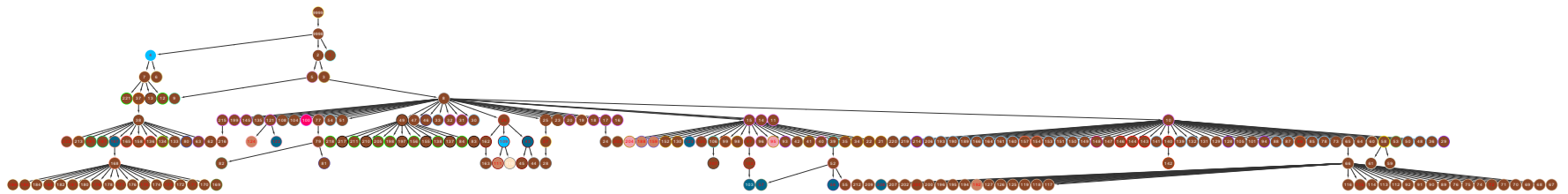


Figure 7.15: The process tree of the rat session 1 dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.

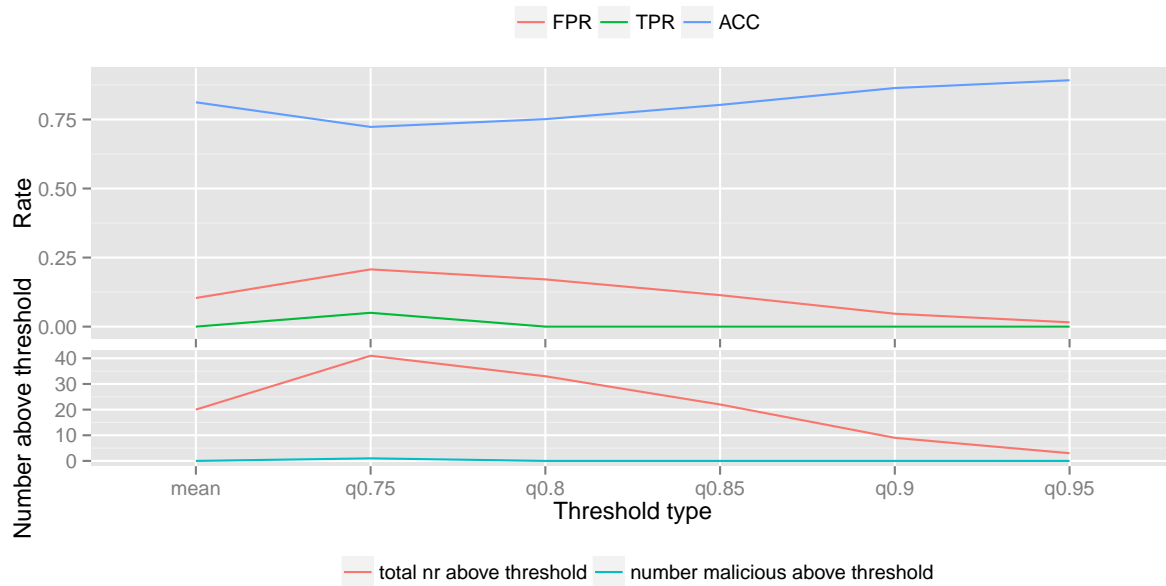


Figure 7.16: Plot of the ACC, FPR and TPR of rat session 2 using method 2

7.2.2.3 Algorithm 2: Rat 2

Only one of the twenty malicious processes is marked correctly in the rat session 2 dataset. This is reached by using the 75% quantile, as can be seen in table 7.32 and figure 7.16.

Table 7.32: Outcome using different threshold values for rat2 malware using method2

threshold type	total nr above threshold	number malicious above threshold	FPR	TPR	ACC
mean	20	0	0.104	0	0.812
q0.75	41	1	0.207	0.050	0.723
q0.8	33	0	0.171	0	0.751
q0.85	22	0	0.114	0	0.803
q0.9	9	0	0.047	0	0.864
q0.95	3	0	0.016	0	0.892

Table 7.33: The five highest ranked benign processes for rat session 2 using algorithm 2.

unique ids	distance sum	malicious
4	27.284	No
111	8.395	No
199	6.871	No
125	6.690	No
124	6.588	No

If we look at table I.13 we see other processes, than analyzed during the analysis of algorithm

1. These processes are 199, 125 and 124. Process 199 is the same executable as 119 in the banking malware. The following process 125, is as can be seen in table I.16, is present in all datasets. Here again early discussed shortcomings due to using different machines is probably the reason for differences in process activities. The same is the case for process 124, as can be seen in the calculated normalized number of process activities in table 7.34.

Table 7.34: Overview of the normalized number of activities for the processes.

filesystem	registry	process.create	thread.create	module.load	ob	df	running time
0.069	0.019	0.156	0.039	0.034	0.045	malware bank	9.748
0.069	0.019	0.156	0.039	0.034	0.045	malware rat session1	0.293
0.069	0.019	0.156	0.039	0.034	0.045	malware rat session2	0.593
0.069	0.019	0.156	0.039	0.034	0.045	malware zeus session1	0.277
0.069	0.019	0.156	0.039	0.034	0.045	malware zeus session2	18.850
0.080	0.024	0.156	0.039	0.039	0.045	win8 1604	0.203
0.080	0.024	0.156	0.039	0.039	0.045	win8 1704	0.313
0.080	0.024	0.156	0.039	0.039	0.045	win8 1804	0.266

Using the process tree shown in figure 7.17 the number of processes can be reduced to 12 of the 41 malicious marked processes. The remaining processes are: 4, 34, 202, 69, 60, 85, 84, 82, 80, 76, 74 and 72.

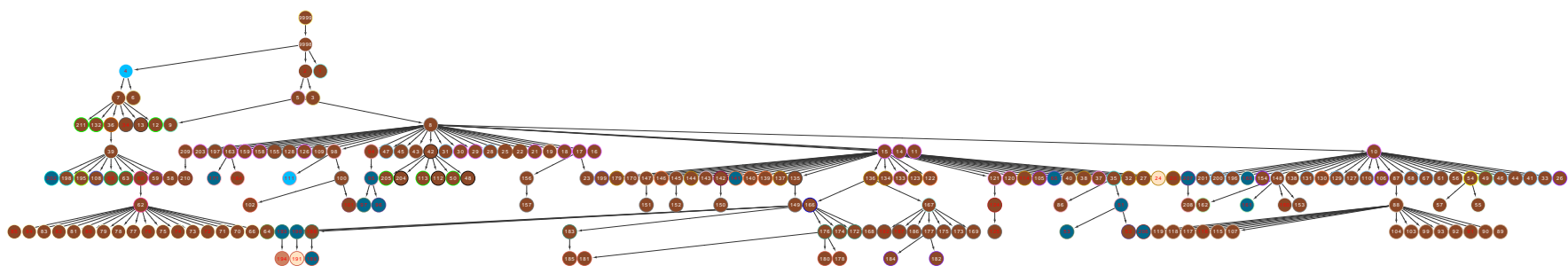


Figure 7.17: The process tree of the rat session 2 malware dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.

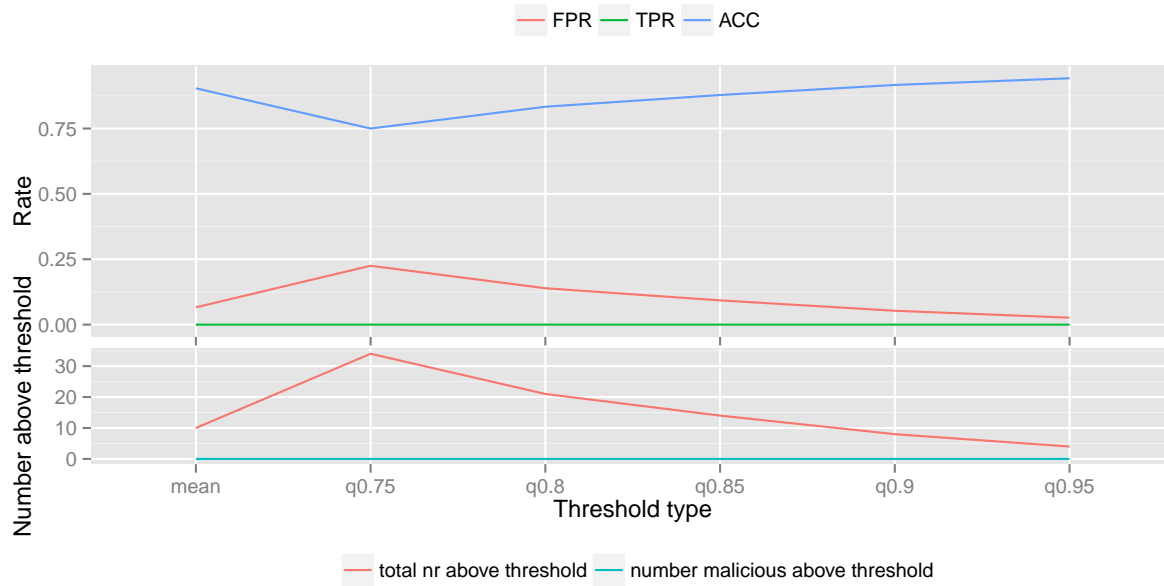


Figure 7.18: Plot of the ACC, FPR and TPR of Zeus session 1 using method 2

7.2.2.4 Algorithm 2: Zeus 1

Algorithm 2 did as well not succeed in marking any of the malicious processes of the Zeus malware correctly, hereby generating a TPR of zero for every threshold type as can be seen in the table and figure below.

Table 7.35: Outcome using different threshold values for zeus1 malware using method2

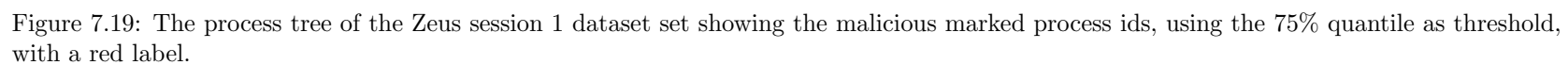
threshold type	total nr above threshold	number malicious above threshold	FPR	TPR	ACC
mean	10	0	0.066	0	0.904
q0.75	34	0	0.225	0	0.750
q0.8	21	0	0.139	0	0.833
q0.85	14	0	0.093	0	0.878
q0.9	8	0	0.053	0	0.917
q0.95	4	0	0.026	0	0.942

Table 7.36: The five highest ranked benign processes for zeus session 1 using algorithm 2.

unique ids	distance sum	malicious
4	32.503	No
111	29.827	No
113	21.907	No
112	15.990	No
40	10.839	No

There are two processes of the zeus session 1 dataset in top five, which were not analyzed previously. These processes are 112 and 40. Process 40 is from the same executable as the process 109 of the rat session 1 malware dataset. For process 112 the same shortcoming of the calculated process activities becomes evident.

Only the following seven processes are left using the process tree elimination technique: 4, 152, 12, 150, 134, 121 and 117. This reduces the number significantly from 34. However still no malicious process is marked as malicious.



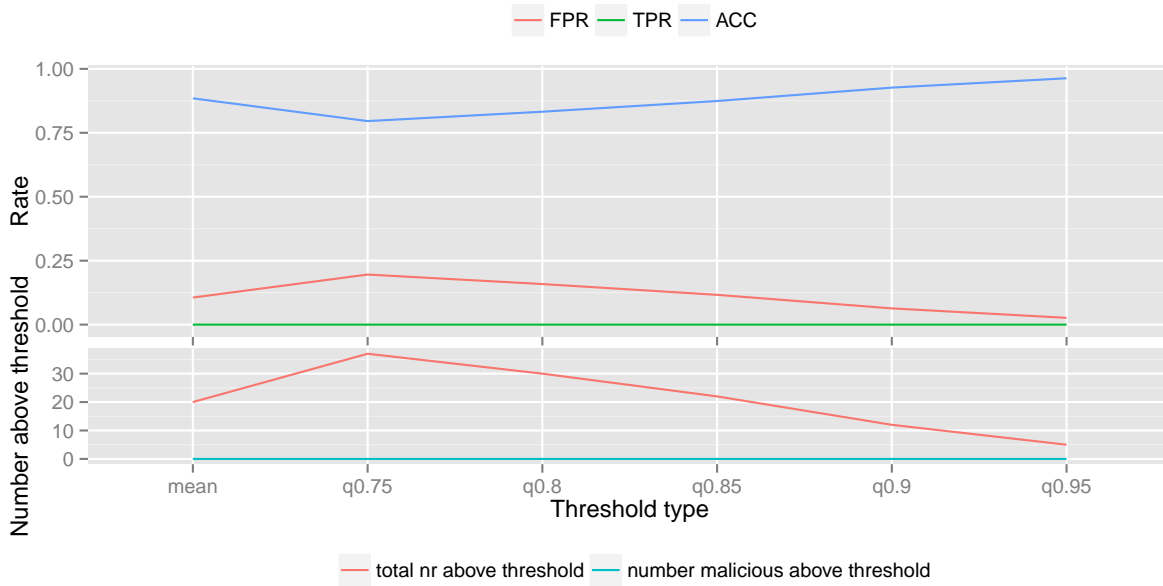


Figure 7.20: Plot of the ACC, FPR and TPR of Zeus session 2 using method 2

7.2.2.5 Algorithm 2: Zeus 2

In this dataset none of the two malicious processes is marked as malicious, hereby again generating a TPR of zero. If we look at the processes in the distribution ranking two not earlier analyzed process can be seen. These processes are 113 and 38. Analyzing these processes shows that process 113 is the same executable as 112 in zeus session 1 data set and process 38 is the same as process 38 in the zeus session 1 dataset and process 109 in the rat session 1 dataset.

Table 7.37: Outcome using different threshold values for zeus2 malware using method2

threshold type	total nr above threshold	number malicious above threshold	FPR	TPR	ACC
mean	20	0	0.106	0	0.885
q0.75	37	0	0.196	0	0.796
q0.8	30	0	0.159	0	0.832
q0.85	22	0	0.116	0	0.874
q0.9	12	0	0.063	0	0.927
q0.95	5	0	0.026	0	0.963

Only 4, 4; 187; 59; 139, of the 37 process marked as malicious are left by using the process tree shown in figure 7.21

Table 7.38: The five highest ranked benign processes for zeus session 2 using algorithm 2.

unique ids	distance sum	malicious
4	32.503	No
112	29.887	No
114	21.837	No
113	14.955	No
38	10.848	No

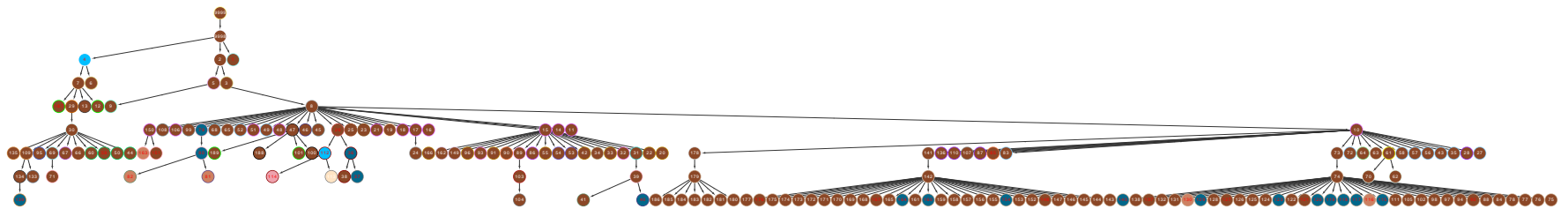


Figure 7.21: The process tree of the Zeus session 2 dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.

7.2.2.6 Conclusion Algorithm 2

Figure 7.22 shows the FPR, TPR and ACC of all malware sets using algorithm 2. Compared to the first algorithm this one does less of a job in marking malicious processes correctly as malicious. This shows that using a strict logarithm for matching does not provide a better outcome.

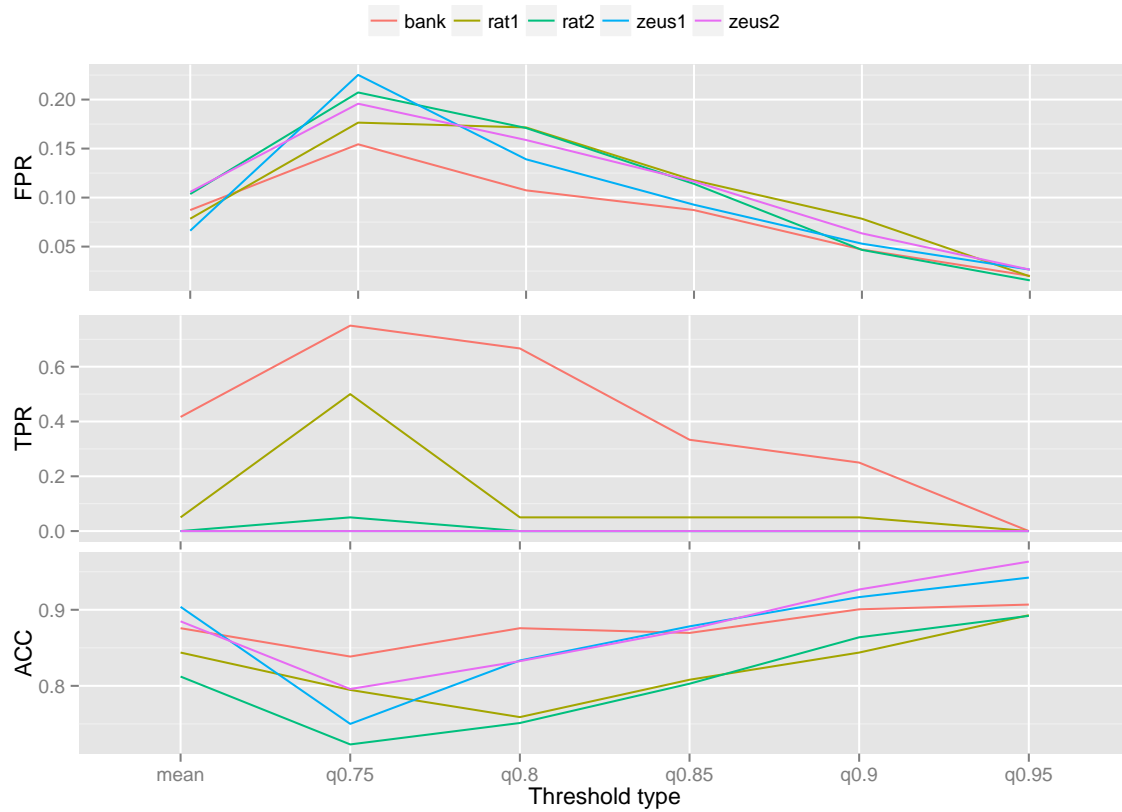


Figure 7.22: Plot of the ACC, FPR and TPR of all malware sets using algorithm 2

7.2.3 Algorithm 3

In the previous two conducted analysis of the algorithms, the usage of the process tree has shown it could drastically reduce the number of processes to analyse for the security officer. However analyzing ten process trees provided no new insights. Therefor we will not include the analysis of these process trees in the coming analysis. However the trees are included in appendix J.

7.2.3.1 Algorithm 3:banking malware

Concluding from table 7.39 and figure 7.23 algorithm 3 is also capable of marking some of the malicious processes from the banking malware correctly as malicious. The outcome is does not differ much from using algorithm 1, expect that it does only marks seven malicious processes compared to the eight processes marked by using algorithm 1.

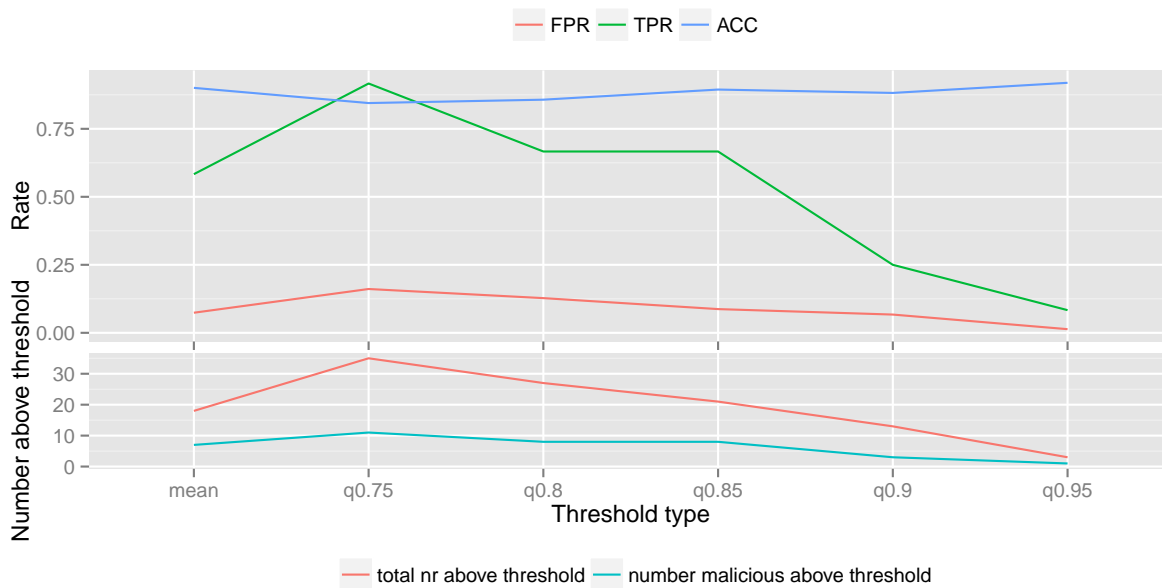


Figure 7.23: Plot of the ACC, FPR and TPR of bank using method 3

Table 7.39: Outcome using different threshold values for bank malware using method3

threshold type	total nr above threshold	number malicious above threshold	FPR	TPR	ACC
mean	18	7	0.074	0.583	0.901
q0.75	35	11	0.161	0.917	0.845
q0.8	27	8	0.128	0.667	0.857
q0.85	21	8	0.087	0.667	0.894
q0.9	13	3	0.067	0.250	0.882
q0.95	3	1	0.013	0.083	0.919

From the ranking top in table I.21 only one process shows up which is not already analysed. As can be seen in table I.12 this executable is only present in the malware datasets.

Table 7.40: The five highest ranked benign processes for banking malware using algorithm 3.

unique ids	distance sum	malicious
4	32.514	No
135	24.814	No
27	7.616	No
153	6.880	No
73	6.591	No

7.2.3.2 Algorithm 3: rat 1

Using algorithm 3 on the rat session 1 dataset provides a comparable outcome as using algorithm 1. As can be seen in the table and figure below. The only difference is a lower FPR using the mean threshold value.

Table 7.41: Outcome using different threshold values for rat1 malware using method3

threshold type	total nr above threshold	number malicious above threshold	FPR	TPR	ACC
mean	18	1	0.083	0.050	0.839
q0.75	40	10	0.147	0.500	0.821
q0.8	27	2	0.123	0.100	0.808
q0.85	23	1	0.108	0.050	0.817
q0.9	16	1	0.074	0.050	0.848
q0.95	9	1	0.039	0.050	0.879

In the ranking no new processes were in the top five, there analyzing these processes would not provide any new insights.

Table 7.42: The five highest ranked benign processes for rat session 1 using algorithm 3.

unique ids	distance sum	malicious
4	24.599	No
109	12.328	No
95	10.132	No
204	10.043	No
111	8.817	No

7.2.3.3 Algorithm 3: rat 2

Here again only using the 75% quantile threshold value marked some of the malicious processes correctly and generates a lower FPR by using the mean as threshold value.

Only one new process was present in the top five of the wrongly marked non-malicious processes, namely process 191. This process is the same executable as process 194 in the rat session 2 data set as well, using algorithm 1.

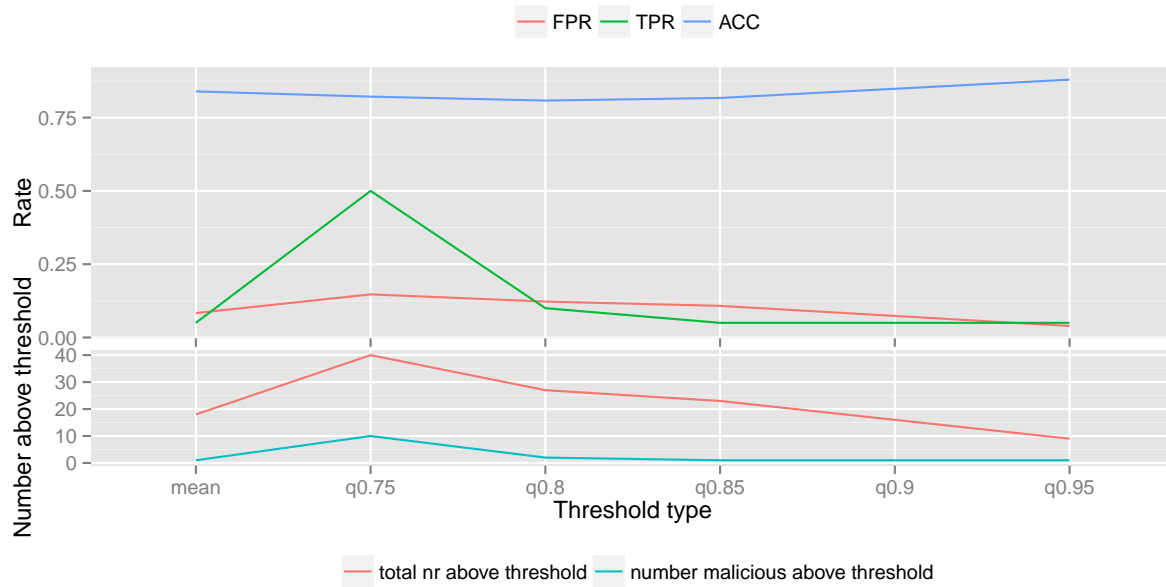


Figure 7.24: Plot of the ACC, FPR and TPR of rat session 1 using method 3

Table 7.43: Outcome using different threshold values for rat2 malware using method3

threshold type	total nr above threshold	number malicious above threshold	FPR	TPR	ACC
mean	28	0	0.145	0	0.775
q0.75	47	8	0.202	0.400	0.761
q0.8	38	0	0.197	0	0.728
q0.85	25	0	0.130	0	0.789
q0.9	11	0	0.057	0	0.854
q0.95	4	0	0.021	0	0.887

Table 7.44: The five highest ranked benign processes for rat session 2 using algorithm 3.

unique ids	distance sum	malicious
4	27.284	No
111	11.374	No
191	4.836	No
161	4.538	No
206	4.142	No

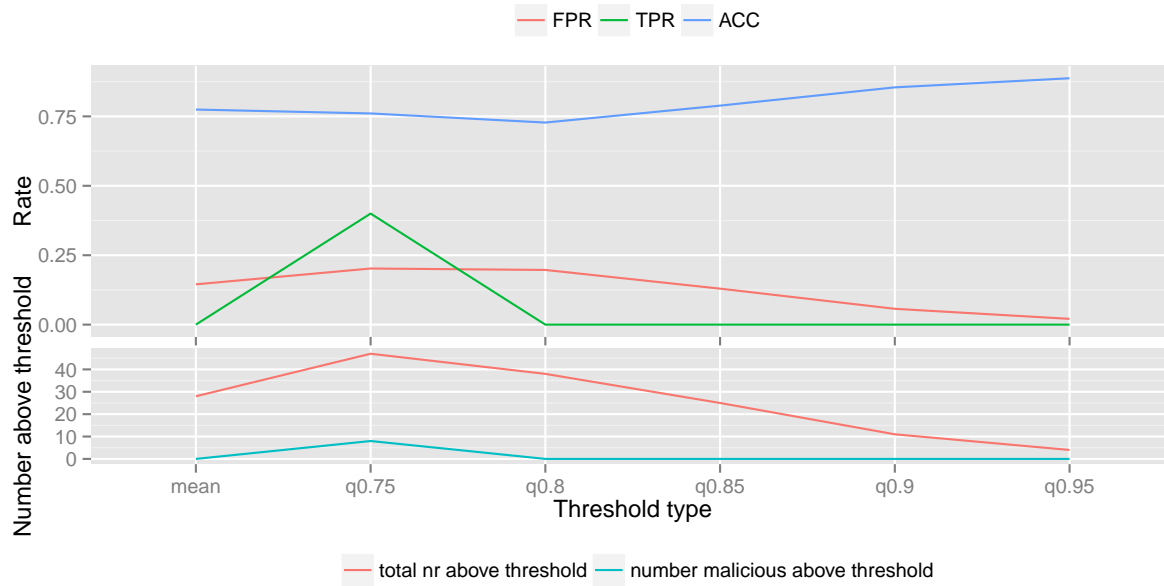


Figure 7.25: Plot of the ACC, FPR and TPR of rat session 2 using method 3

7.2.3.4 Algorithm 3: Zeus 1

Also algorithm 3 is not able to mark any of the malicious processes from the Zeus malware correctly. Here again a TPR of zero is the maximum reached.

appendix H.4

Table 7.45: Outcome using different threshold values for zeus1 malware using method3

threshold type	total nr above threshold	number malicious above threshold	FPR	TPR	ACC
mean	12	0	0.079	0	0.891
q0.75	35	0	0.232	0	0.744
q0.8	26	0	0.172	0	0.801
q0.85	19	0	0.126	0	0.846
q0.9	8	0	0.053	0	0.917
q0.95	5	0	0.033	0	0.936

The not already analyzed process 80 from the top five ranking, is the same executable as process 73 from the banking malware discussed in the first section of this analysis.

7.2.3.5 Algorithm 3: Zeus 2

As can be seen in figure 7.27 and table 7.47 here again none of the malicious processes is marked as malicious.

In table I.29 one new process is present, namely 79. This process is only ran in the malware datasets as can be seen from table I.30. The executable belonging to process 79 is the same as the earlier discussed process 80 in zeus session 1 and process 73 in the banking malware set.

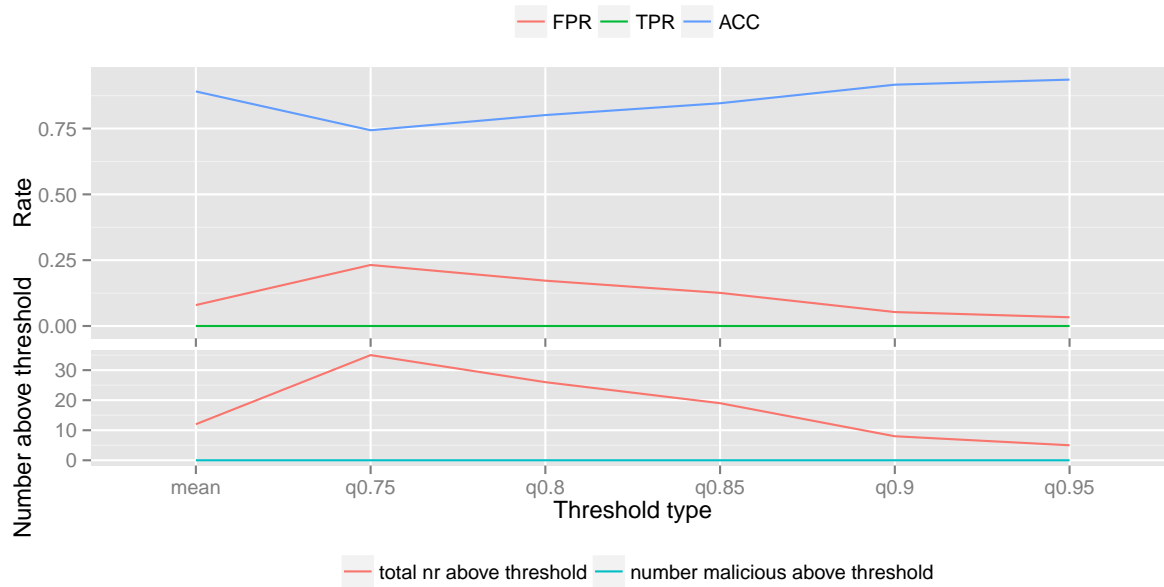


Figure 7.26: Plot of the ACC, FPR and TPR of Zeus session 1 using method 3

Table 7.46: The five highest ranked benign processes for zeus session 1 using algorithm 3.

unique ids	distance sum	malicious
4	32.503	No
111	15.913	No
113	15.806	No
80	6.663	No
107	6.191	No

Table 7.47: Outcome using different threshold values for zeus2 malware using method3

threshold type	total nr above threshold	number malicious above threshold	FPR	TPR	ACC
mean	19	0	0.101	0	0.890
q0.75	38	0	0.201	0	0.791
q0.8	28	0	0.148	0	0.843
q0.85	21	0	0.111	0	0.880
q0.9	13	0	0.069	0	0.921
q0.95	6	0	0.032	0	0.958

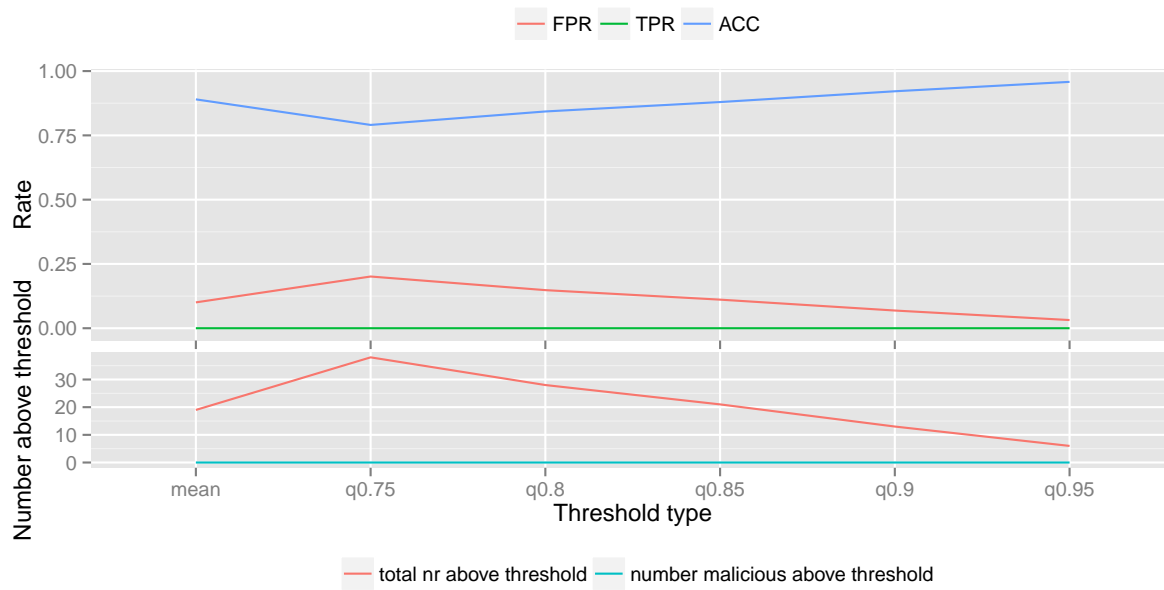


Figure 7.27: Plot of the ACC, FPR and TPR of Zeus session 2 using method 3

Table 7.48: The five highest ranked benign processes for zeus session 2 using algorithm 3.

unique ids	distance sum	malicious
4	32.503	No
112	15.942	No
114	15.749	No
79	6.511	No
130	5.498	No

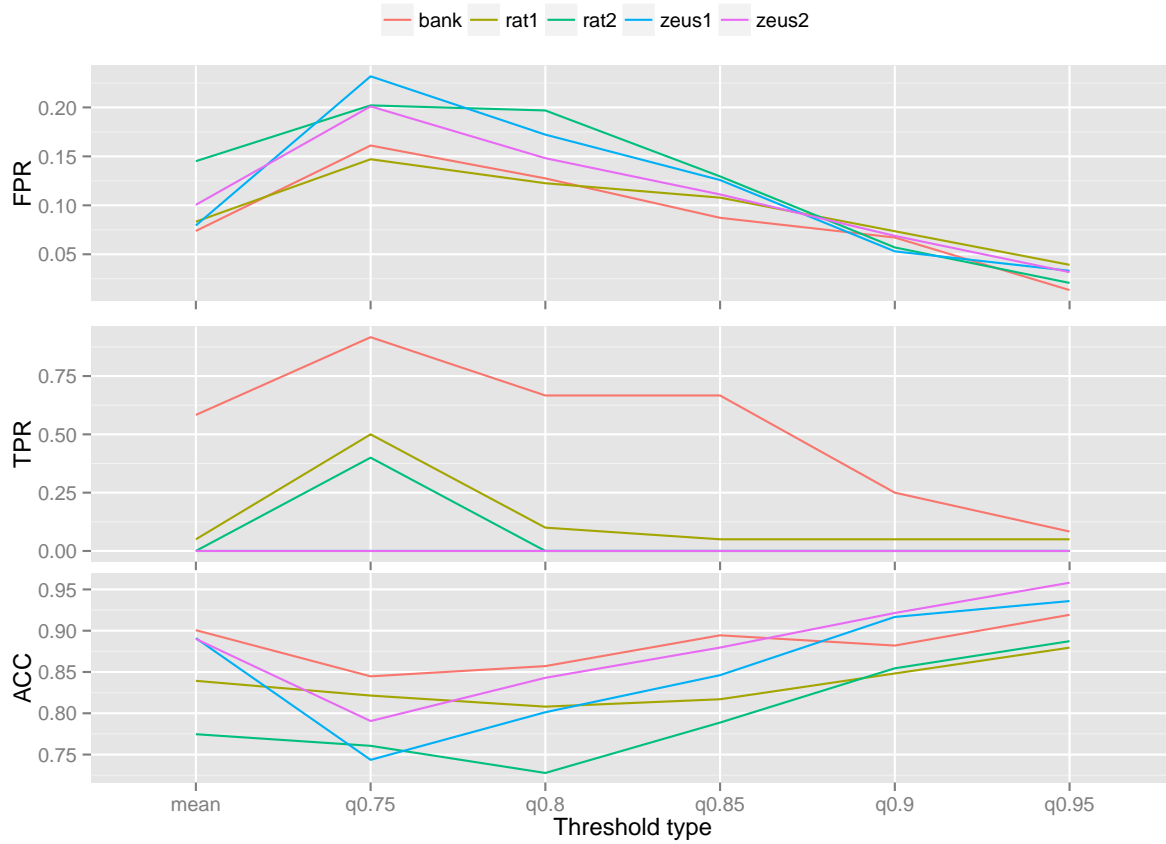


Figure 7.28: Plot of the ACC, FPR and TPR of all malware sets using method 3

7.2.3.6 Conclusion Algorithm method 3

The evaluation of algorithm 3 showed that it performed quite comparable to algorithm 1. Using algorithm 3 on the banking malware dataset it had one less malicious process correctly marked. On the other hand algorithm 3 provided a lower FPR for some of the datasets. Figure 7.28 provides a graphical overview of the FPR, TPR and ACC of the analyzed malware datasets.

7.3 Conclusion

In the previous sections we have evaluated the three proposed algorithms, which showed positive results. All the three proposed algorithms were able to mark malicious processes correctly. However not every malware types was detectable using these algorithms. None of the algorithms could detect any of the malicious processes in both Zeus malware datasets. To evaluate which algorithm performed the best, the TPR, FPR and ACC was calculated for every algorithm on every dataset. In figure 7.29 for every dataset and algorithm the TPR, FPR and ACC are presented. Concluding from the graphical presentation we see that the TPR for the banking malware set is the highest on all algorithms.

In table 7.49 the summary is given for the count of highest TPR and ACC and lowest FPR. As we can see algorithm 1 and 3 score on all three evaluation measure almost the same. The TPR score of algorithm 2 is lower than the scores of algorithm 1 and 3. However algorithm 2 has a better score on the FPR and ACC. The main reason for this is the fact that algorithm 2 marks less processes as malicious. Taking into account the time algorithm 2 needs to run, as discussed in the previous chapter,

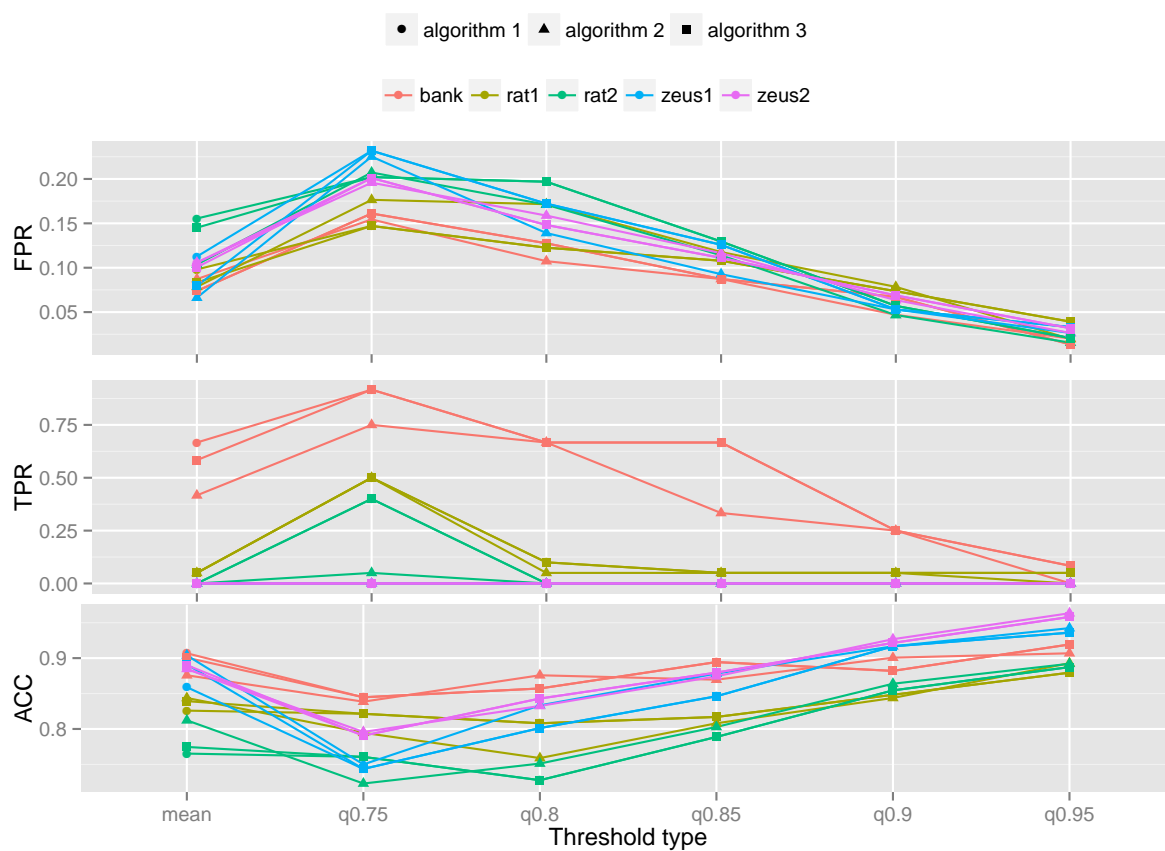


Figure 7.29: Plots of FPR, TPR and ACC of all malware and all methods

and that our focus is on gaining a high TPR, algorithm 2 should be discarded for future work.

In related work discussed in chapter 2 the TPR ranges mostly between 0.7 and up to 0.99 and the FPR ranges from 0.0 up to 0.32 [55, 69]. The values we have evaluated range from a TPR of 0.0 up to 0.917 and FPR of 0.013 up to 0.232. So we can conclude that these rates are not to strange for a novelty research.

Furthermore during the evaluation several shortcomings and recommendations for future work were discussed. These recommendations will be explained in more detail in the next chapter which will conclude this thesis.

Table 7.49: Overview of the number highest TPR and lowest FPRs

Algorithm	nr of times highest TPR	nr of times lowest FPR	nr of times highest ACC
Algorithm 1	30	11	12
Algorithm 2	23	20	18
Algorithm 3	29	12	12

8

Conclusion

In this chapter we provide the conclusion of this master thesis research. We will first evaluate the answers to sub questions, after which we will answer the main research question. This chapter will conclude with shortcomings and recommendations for future research.

8.1 Reflection on research questions

This section will provide the answers found during the research conducted for this master thesis. *What are the current state-of-the-art anomaly detection methods for malicious behavior?*

In chapter 2 this question was answered. The conclusion of the literature research conducted is described below.

In [66] a tree is created showing the relations between processes and process, process and programs and processes and system calls. To train the detection model a supervised SVM was used. The use of trees to show process activity is done in [63] as well. The use of filesystem activity, changes to registry, infection of running processes, network activity and starting and stopping of services for detection malware is presented in [53].

We have extended upon this work by creating process trees which presents information on the process activities as well adding process tree information to the data to be used in the detection model.

Which design requirements should be taken into account when developing an anomaly detection method?

The answer to this question is given in chapter 3. Concluding from this chapter we have found that when we are dealing with data that can be traced back to a person in the Netherlands the law *Wet bescherming persoonsgegevens*. Although the data collected in this thesis could contain information that traces back to a person. However the processed data used for creating the process trees and heatmaps and on which the algorithms were deployed does not contain any personal data. So depending on which data will be collected in the final implementation, privacy laws and regulations should be kept in mind.

Another design requirement is the limitation of processing power and storage capacity. As shown in chapter 4 the amount of data collected during one working day contained about 9 to 10 million

events with about 50 variables. By processing the data this was reduced to only 900 events with about 18 variables.

In chapter 6 the running times of the three algorithms are shown. Concluding from this, algorithm 3 has the shortest running time, of about 22 seconds to compare two datasets with about 800 to 900 events.

Which data can be used for modeling benign process behavior?

As explained in section 4.5 the following events are used: filesystem, registry, process create, thread create, object callback and module load. For these events we calculated the number of times such an event is trigger per second by a process. Furthermore we use the unique id of the parent process and use the timestamp of the process create and process exit to calculate the running time of a process. Based on the information from the process tree the depth of each process in the tree is added to the data, as well as the cluster to which the process belongs.

How well does the constructed method detect malicious behavior?

Malicious processes from two out of the three malware samples were detected using the algorithms and threshold value to mark processes malicious. In figure 7.29 in section 7.3 a graphical overview is given of the TPR, FPR and ACC of all three algorithms on all malware sets. The TPR ranges between 0 and 0.917, the FPR is between 0.013 up to 0.232. For a novelty research these values are not strange.

What graphical presentations of the malicious marked processes can aid a security officer?

In this thesis we showed heatmaps and process trees as graphical tools to aid security officers in analyzing malicious processes. As shown in chapter 5 the process trees aided in detecting process with deviations. However the this deviation was in the upper part of the process tree. Going deeper down the process tree the number of nodes becomes huge and small deviations can easily be missed.

The heatmap showed the possibility to pin point processes with higher process activities compared to the other processes in the dataset. Whenever a cell is colored more yellow to white the number of events trigger per second for that process activity is higher than other processes.

We can conclude that all sub questions are answered. Below we will provide an answer to our main question.

How can anomaly based detection be used for detecting unknown malicious processes based on the detailed process information gathered on a single endpoint?

The focus of this master thesis research was to do an explorative research on the possibility of using process activities as input for an anomaly detection method to detect malicious processes. These process activities enclose filesystem, registry, process create, process exit, thread create, thread exit, module load and ob activities. We assume that malicious process activities would deviate from benign process activities.

During the research project we proved that processes perform different number of activities per time unit and herein differ from each other. Some processes do not create new process whilst other process, such as for example Google Chrome do create a lot of new processes.

We calculated the amount of events, such as filesystem activities, triggered per second by a process. Using this measure we showed by using heatmaps that processes started by the same executable tend to group together. However not every process from the same executable would group with its fellow processes. As the heatmaps proved processes could be grouped a k-means clustering with, eight clusters, was applied to the collected data set.

As the data logged provided the first process started by Windows, which would spawn new processes we were able to construct a process tree. These process trees showed that the first few nodes where

always the same in all datasets. Based on this information we concluded that certain process mostly tend to be started at a certain depth in the tree.

This information combined with the clustering of the processes based on the process activities, was used as input for developing three comparing algorithms, to compare the datasets containing malicious processes with clean datasets. In these comparing methods the Euclidean distance measure was used to match the nodes together based on the six event activities and their depth and cluster.

As we assume that malicious processes would have deviating behavior from benign processes, we expect that malware processes would have a higher distance with a matching node. Therefor in evaluating the compare methods a border values was set. Every process with a distance higher than the border measure was marked as malicious. We have tested several limit values: using the distance mean and the 75, 80, 85, 90, 95% quantile.

The malicious processes were identified in a earlier stage of the research, so we could calculate the False Positive Rate and True Positive Rate. In analyzing the outcome of the three algorithms we found that mostly the malicious processes of the banking malware were detected. Even getting up to a TPR of 0.917. However the downside was the fact that the FPR was 0.161.

For the other malware sets the algorithms seemed to perform less, for the zeus malware none of the malicious processes was detected. The rat malware gained higher TPR, namely 0.5.

8.2 Shortcomings and recommendations for future research

While conducting this research several shortcomings and recommendations were mentioned. These are summarized below:

The first and most influential shortcoming is the set-up for collecting the data. Due to security constraints the collection of malware data was done on a different machine than the clean data. Although the effort was made to construct an identical system as possible, not all software was identical. This became clear in executables which were only present on the system used for collecting the malware data or vice versa. This would generate false positives as the normal behavior of the applications on the malware system were not available in the clean data. Therefor we would recommend that for future research one system is used to collect the data.

A second important deficiency is the amount of data collect, which is especially true for the collection of the malware data. The short time periods used for collecting the malware data was imposed by both security and time restrictions. As the malware was installed on a system on the company network, the amount of time the malware was allowed to run was limited. Furthermore as the malware is installed it can gather information from the system, to prevent leakage of any personal or company information we could not do any real work, and had to imitate it during the infected period.

Another shortcoming encountered during this master thesis research was the problem of different running times creating different normalized values for the number of events per second, mentioned and analyzed in chapter 5.1.5. Apparently some processes perform a certain set of actions, which would give different per second values if the running time is different. Therefor it is important that a more robust calculation method should be constructed to create a value which can be compared.

In this thesis the data was normalized between 0 and 10. However other normalization algorithms can be used. For example the Z-score might be a viable option.

Furthermore during this research only k-means clustering was used with eight clusters. However it could be that using less or more clusters would provide a better basis for finding malicious processes. Other clustering methods could be researched as well. Some of the papers analyzed during the literature research mentioned the use of a one-class SVM for clustering.

Analyzing the process trees, showed us that all malware happens in a particular place in the process tree, namely the left side. As the malware samples were extracted from a zip and executed, this might have an implication on the place in the process tree where the malware is executed. Tests with infection via malicious advertisement and other methods of getting infected should be tested as well. It could well be that malware is always running at a specific place in the process tree.

In evaluation none of the algorithms detected the malicious processes of the Zeus malware. As we did not monitor the outgoing traffic we do not know if the Zeus malware received any commands from the command and control centers. In future research combining the logging of process information with the logging of network data might provide viable information.

The ranking of the malicious processes is done on the distance. However other ranking methods are possible as well. We assume that a process started by a malicious process is malicious as well. If the detection method works it should mark both processes as malicious. If we add the summed distance of the child process to the distance of the parent process it will move up in the ranking and will be investigated earlier by the security officer.

During this research we used only datasets from complete boot procedures, which we would analyse afterwards. When implementing a malicious behavior detection system you would like to know as fast as possible if there is a possible infection. To overcome this problem the method should be rewritten in such a way that it could in real time add new processes to the process tree, assign the appropriate cluster and analyze if it has such a deviation it would be marked as benign or malicious.

9

Bibliography

- [1] Fox-it. <https://www.fox-it.com/en/>. [Online; accessed 21-04-2015].
- [2] Number of new malware per year. <http://www.av-test.org/en/statistics/malware/>. [Online; accessed 15-january-2015].
- [3] Cisco systems netflow services export version 9. <https://tools.ietf.org/html/rfc3954>, October 2004. [Online; accessed 27-Januari-2015].
- [4] Classless inter-domain routing (cidr): The internet address assignment and aggregation plan. <http://tools.ietf.org/html/rfc4632>, August 2006. [Online; accessed 27-Januari-2015].
- [5] Five predictions for information security and cybercrime in 2014. <http://www.theguardian.com/media-network/media-network-blog/2013/dec/10/predictions-information-security-cybercrime-2014>, December 2013. [Online; accessed 30-June-2014].
- [6] Hp reveals cost of cybercrime escalates 78 percent, time to resolve attacks more than doubles. <http://www8.hp.com/us/en/hp-news/press-release.html?id=1501128#.U6wwULGnDIX>, October 2013. [Online; accessed 26-June-2014].
- [7] Security threat report 2014: Smarter, shadier, stealthier malware. Report, 2013.
- [8] Life with nest thermostat. <https://nest.com/thermostat/life-with-nest-thermostat/>, June 2014. [Online; accessed 23-July-2014].
- [9] Malicious advertisements served via yahoo. <http://blog.fox-it.com/2014/01/03/malicious-advertisements-served-via-yahoo/>, januari 2014. [Online; accessed 28-August-2015].
- [10] Microsoft security intelligence report, volume 17. Technical report, Microsoft, 2014.
- [11] Net losses: Estimating the global cost of cybercrime, Jun 2014.
- [12] Samsung smart home becomes reality, set to transform everyday life. <http://global.samsungtomorrow.com/?p=35496>, 2014. [Online; accessed 23-July-2014].
- [13] 143 miljoen nieuwe malware-exemplaren in 2014. <https://www.security.nl/posting/414387/143+miljoen+nieuwe+malware-exemplaren+in+2014>, January 2015. [Online; accessed 08-January-2015].

-
- [14] Amerikaans politiekorps betaalt 300 dollar aan ransomware. <https://www.security.nl/posting/424722/Amerikaans+politiekorps+betaalt+300+dollar+aan+ransomware>, 2015. [Online; accessed 21-04-2015].
 - [15] Grootschalige aanval via macro's verspreidt ransomware. <https://www.security.nl/posting/424616/Grootschalige+aanval+via+macro%27s+verspreidt+ransomware>, 2015. [Online; accessed 21-04-2015].
 - [16] Malafide hugo boss-advertentie verspreidt ransomware. <https://www.security.nl/posting/424883/Malafide+Hugo+Boss-advertentie+verspreidt+ransomware>, 2015. [Online; accessed 21-04-2015].
 - [17] R: What is r? <http://www.r-project.org/about.html>, 2015. [Online; accessed 28-August-2015].
 - [18] Ransomware besmet tientallen computers zorgverlener. <https://www.security.nl/posting/424457/Ransomware+besmet+tientallen+computers+zorgverlener>, 2015. [Online; accessed 21-04-2015].
 - [19] Ransomware infecteert ziekenhuizen via privmail. <https://www.security.nl/posting/423555/Ransomware+infecteert+ziekenhuizen+via+priv%C3%A9mail>, 2015. [Online; accessed 21-04-2015].
 - [20] Riskiq reports 260 percent spike in malicious advertisements in 2015. <http://www.businesswire.com/news/home/20150804006069/en/RiskIQ-Reports-260-Percent-Spike-Malicious-Advertisements#.VehKd5eje00>, august 2015. [Online; accessed 28-August-2015].
 - [21] Schooldistrict verschuift examens wegens ransomware. <https://www.security.nl/posting/423064/Schooldistrict+verschuift+examens+wegens+ransomware>, 2015. [Online; accessed 21-04-2015].
 - [22] The summer of malicious online advertising: Msn users targeted. <http://www.lavasoft.com/mylavasoft/company/blog/the-summer-of-malicious-online-advertising-msn-users-targeted>, august 2015. [Online; accessed 28-August-2015].
 - [23] Wet bescherming persoonsgegevens. wetten.overheid.nl/BWBR0011468/geldigheidsdatum_09-09-2015, august 2015. [Online; accessed 28-August-2015].
 - [24] Pieter Burghouwt, Marcel Spruit, and Henk Sips. Towards detection of botnet communication through social media by monitoring user activity. In Sushil Jajodia and Chandan Mazumdar, editors, *Information Systems Security*, volume 7093 of *Lecture Notes in Computer Science*, pages 131–143. Springer Berlin Heidelberg, 2011.
 - [25] Pieter Burghouwt, Marcel Spruit, and Henk Sips. Detection of covert botnet command and control channels by causal analysis of traffic flows. In Guojun Wang, Indrakshi Ray, Dengguo Feng, and Muttukrishnan Rajarajan, editors, *Cyberspace Safety and Security*, volume 8300 of *Lecture Notes in Computer Science*, pages 117–131. Springer International Publishing, 2013.
 - [26] Pedro Casas, Johan Mazel, and Philippe Owezarski. Unsupervised network intrusion detection systems: Detecting the unknown without knowledge. *Computer Communications*, 35(7):772 – 783, 2012.
 - [27] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

-
- [28] Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect malicious patterns. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03, pages 12–12, Berkeley, CA, USA, 2003. USENIX Association.
 - [29] Weidong Cui, R.H. Katz, and Wai tian Tan. Design and implementation of an extrusion-based break-in detector for personal computers. In *Computer Security Applications Conference, 21st Annual*, pages 10 pp.–370, Dec 2005.
<http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1565263&url=http>
 - [30] Jaime Devesa, Igor Santos, Xabier Cantero, Yoseba K Penya, and Pablo Garcia Bringas. Automatic behaviour-based analysis and classification system for malware detection. In *ICEIS (2)*, pages 395–399, 2010.
 - [31] L. Dolberg, J. Francois, and T. Engel. Multi-dimensional aggregation for dns monitoring. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pages 390–398, Oct 2013.
 - [32] Nicolas Falliere. Stuxnet introduces the first known rootkit for industrial control systems. <http://www.symantec.com/connect/blogs/stuxnet-introduces-first-known-rootkit-scada-devices>, 2010. [Online; accessed 24-07-2014].
 - [33] H.H. Feng, O.M. Kolesnikov, P. Fogla, Wenke Lee, and W. Gong. Anomaly detection using call stack information. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, pages 62–75, May 2003.
 - [34] J. Francois, R. State, and T. Engel. Aggregated representations and metrics for scalable flow analysis. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 478–482, Oct 2013.
 - [35] J. Francois, C. Wagner, R. State, and T. Engel. Safem: Scalable analysis of flows with entropic measures and svm. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 510–513, April 2012.
 - [36] David French and William Casey. 2 fuzzy hashing techniques in applied malware analysis. *Results of SEI LineFunded Exploratory New Starts Projects*, 2012.
 - [37] Google. Protocol buffers. <https://developers.google.com/protocol-buffers/>, 2015. [Online; accessed 02-06-2015].
 - [38] Jaspreet Minhas Harjinder Kaur, Gurpreet Singh. A review of machine learning based anomaly detection techniques. *International Journal of Computer Applications Technology and Research*, 2(2):185 – 187, 2013.
 - [39] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):pp. 100–108, 1979.
 - [40] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2009.
 - [41] Dae-Ki Kang, D. Fuller, and V. Honavar. Learning classifiers for misuse and anomaly detection using a bag of system calls representation. In *Information Assurance Workshop, 2005. IAW '05. Proceedings from the Sixth Annual IEEE SMC*, pages 118–125, June 2005.
 - [42] H.D. Kuna, R. Garca-Martinez, and F.R. Villatoro. Outlier detection in audit logs for application systems. *Information Systems*, 44(0):22 – 33, 2014.
 - [43] Derek Manky. Cybercrime as a service: a very modern business. *Computer Fraud & Security*, 2013(6):9 – 13, 2013.

-
- [44] Morgan Marquis-Boire, Claudio Guarnieri, and Ryan Gallagher. Secret malware in european union attack linked to u.s. and british intelligence. <https://firstlook.org/theintercept/2014/11/24/secret-regin-malware-belgacom-nsa-gchq/>, November 2014. [Online; accessed 02-December-2014].
 - [45] VasileiosA. Memos and KostasE. Psannis. A new methodology based on cloud computing for efficient virus detection. In Khaled Elleithy and Tarek Sobh, editors, *New Trends in Networking, Computing, E-learning, Systems Sciences, and Engineering*, volume 312 of *Lecture Notes in Electrical Engineering*, pages 37–47. Springer International Publishing, 2015.
 - [46] Huub Modderkolk. Kwaadaardige malware regin gebruikt bij hack belgacom. <http://www.nrc.nl/nieuws/2014/11/24/kwaadaardige-malware-regin-gebruikt-bij-hack-belgacom/>, November 2014. [Online; accessed 02-December-2014].
 - [47] Sorin Mustaca. Are your IT professionals prepared for the challenges to come? *Computer Fraud & Security*, 2014(3):18 – 20, 2014.
 - [48] Emeric Nasi. Bypass antivirus dynamic analysis: Limitations of the av model and how to exploit them. internet, August 2014.
 - [49] Pierluigi Paganini. 2013 norton report, the impact of cybercrime according symantec. <http://securityaffairs.co/wordpress/18475/cyber-crime/2013-norton-report.html>, October 2013. [Online; accessed 26-June-2014].
 - [50] Abigail Prichel. Cybercriminals unleash bitcoin-mining malware. <http://about-threats.trendmicro.com/us/webattack/93/Cybercriminals+Unleash+BitcoinMining+Malware>. [Online; accessed 02-July-2014].
 - [51] Brian Prince. Hacker builds massive dogecoin mining operation with synology nas boxes. <http://www.securityweek.com/hacker-builds-massive-dogecoin-mining-operation-synology-nas-boxes>, 2014. [Online; accessed 02-July-2014].
 - [52] Symantec Security Response. W32.duqu: The precursor to the next stuxnet. http://www.symantec.com/connect/w32_duqu_precursor_next_stuxnet, October 2011. [Online; accessed 24-07-2014].
 - [53] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Dssel, and Pavel Laskov. Learning and classification of malware behavior. In Diego Zamboni, editor, *Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 5137 of *Lecture Notes in Computer Science*, pages 108–125. Springer Berlin Heidelberg, 2008.
 - [54] Mithun Sanghavi. Dridex and how to overcome it. <http://www.symantec.com/connect/blogs/dridex-and-how-overcome-it>, October 2015. [Online; accessed 02-06-2015].
 - [55] Zhiyong Shan and Xin Wang. Growing grapes in your computer to defend against malware. *Information Forensics and Security, IEEE Transactions on*, 9(2):196–207, Feb 2014.
 - [56] Alisa Shevchenko. Malicious code detection technologies. *Moscow: Kaspersky Lab*, 2008.
 - [57] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 305–316, May 2010.
 - [58] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, and Koji Nakao. Toward a more practical unsupervised anomaly detection system. *Information Sciences*, 231(0):4 – 14, 2013. Data Mining for Information Security.

-
- [59] Symantec Security Response. Dragonfly: Western energy companies under sabotage threat. <http://www.symantec.com/connect/blogs/dragonfly-western-energy-companies-under-sabotage-threat>, June 2014. [Online; accessed 01-July-2014].
- [60] Peter Szor. *The art of computer virus research and defense*. Pearson Education, 2005.
- [61] P Vinod, R Jaipur, V Laxmi, and M Gaur. Survey on malware detection methods. In *Proceedings of the 3rd Hackers Workshop on Computer and Internet Security (IITKHACK09)*, pages 74–79, 2009.
- [62] P. Vinod, V. Laxmi, M.S. Gaur, and G. Chauhan. Momentum: Metamorphic malware exploration techniques using msa signatures. In *Innovations in Information Technology (IIT), 2012 International Conference on*, pages 232–237, March 2012.
- [63] Gérard Wagener, Alexandre Dulaunoy, Thomas Engel, et al. Self adaptive high interaction honeypots driven by game theory. In *Stabilization, Safety, and Security of Distributed Systems*, pages 741–755. Springer, 2009.
- [64] C. Wagner, J. Francois, R. State, and T. Engel. Danak: Finding the odd! In *Network and System Security (NSS), 2011 5th International Conference on*, pages 161–168, Sept 2011.
- [65] Cynthia Wagner, Jrme Franois, Radu State, and Thomas Engel. Machine learning approach for ip-flow record anomaly detection. In Jordi Domingo-Pascual, Pietro Manzoni, Sergio Palazzo, Ana Pont, and Caterina Scoglio, editors, *NETWORKING 2011*, volume 6640 of *Lecture Notes in Computer Science*, pages 28–39. Springer Berlin Heidelberg, 2011.
- [66] Cynthia Wagner, Gérard Wagener, R State, and Thomas Engel. Malware analysis with graph kernels and support vector machines. In *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*, pages 63–68. IEEE, 2009.
- [67] Y. Wang, J. Wong, and A. Miner. Anomaly intrusion detection using one class svm. In *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, pages 358–364, June 2004.
- [68] Rui Yang, Panos Kalnis, and Anthony K. H. Tung. Similarity evaluation on tree-structured data. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’05, pages 754–765, New York, NY, USA, 2005. ACM.
- [69] Ding Yuxin, Yuan Xuebing, Zhou Di, Dong Li, and An Zhanchao. Feature representation and selection in malicious code detection methods based on static system calls. *Computers & Security*, 30(67):514 – 524, 2011.
- [70] Hao Zhang, W. Banick, Danfeng Yao, and N. Ramakrishnan. User intention-based traffic dependence analysis for anomaly detection. In *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*, pages 104–112, May 2012.
- [71] Li Zhang and Li Zhang. On matching nodes between trees. Technical report, 2003.
- [72] Qinghua Zhang. *Polymorphic and metamorphic malware detection*. ProQuest, 2008.
- [73] Yang Zhong, Christopher A. Meacham, and Sakti Pramanik. A general method for tree-comparison based on subtree similarity and its use in a taxonomic database. *Biosystems*, 42(1):1 – 8, 1997.
- [74] Linhong Zhu, Wee Keong Ng, and James Cheng. Structure and attribute index for approximate graph matching in large graphs. *Information Systems*, 36(6):958 – 972, 2011.

Appendices

A

Collected data

A.1 File system event

File system information

Data field	Required/ Optional	Information
filesystem_event.requestor_mode	Required	Operation initiated from kernel or user land
filesystem_event.minor_function	Required	Subfunction for major function, used by some processes
filesystem_event.major_function	Required	What kind of filesystem event was triggered
filesystem_event.operation_id	Required	Unique ID per filesystem event to track post and pre events
filesystem_event.is_network_operation	Required	Operation over the network yes or no
filesystem_event.parameters. create_parameters.create_options	Required	Create options
filesystem_event.parameters. create_parameters.desired_access_mask	Required	Desired access mask
filesystem_event.parameters. create_parameters.full_create_options	Required	Create options
filesystem_event.parameters. create_parameters.share_access	Required	How the file is shared (read/write/locked)
filesystem_event.parameters. create_parameters.file_attributes	Required	Defines the file attributes values
filesystem_event.parameters. create_parameters.allocation_size	Optional	Size of created file

Table A.1 – continues on next page

Table A.1 – continued from previous page

Data field	Required/ Optional	Information
filesystem_event.parameters. read_write_parameters.byte_offset	Required	Offset where to start reading/writing
filesystem_event.parameters. read_write_parameters.length	Required	Length of read/write
filesystem_event.parameters. file_information_parameters.file_information_class	Required	Information requested from the filesystem
filesystem_event.parameters. section_sync_parameters.page_protection	Required	Requested memory access for a file mapped in memory
filesystem_event.parameters. section_sync_parameters.sync_type	Required	New memory map was created if enum=1
filesystem_event.source_process_id.id	Required	The process ID of the process initiating the filesystem event
filesystem_event.source_process_id.unique_id	Required	The unique process ID of the process initiating the filesystem event
filesystem_event.source_thread_id.id	Required	The thread ID of the process initiating the filesystem event
filesystem_event.source_thread_id.unique_id	Required	The unique thread ID of the process initiating the filesystem event
filesystem_event.file_path.token	Required	Tokenized path to the file
filesystem_event.file_path.data	Required	Path to the file
filesystem_event.operation_result.information	Required	Information depending on the performed major operation for a file create
filesystem_event.operation_result.status	Required	The operation result status
filesystem_event.rename_file_path.token	Optional	The tokenized renamed file path
filesystem_event.rename_file_path.data	Optional	The renamed file path String
filesystem_event.file_hash	Optional	The hash of a file

Table A.1: File system data fields

A.2 Registry event

A detail overview of the information in a registry event.

Data field	Required/ Optional	Information
registry_event.desired_access_mask	Required	Requested access mask (read, write)
registry_event.granted_access_mask	Required	Granted access mask (read, write)
registry_event.registry_operation	Required	What kind of registry operation was performed

Table A.2 – continues on next page

Table A.2 – continued from previous page

Data field	Required/ Optional	Information
registry_event.result_status	Required	Result of the operation
registry_event.registry_value_type	Optional	Type of registry information saved in the key
registry_event.registry_path.token	Required	Tokenized registry path
registry_event.registry_path.data	Optional	Registry key path
registry_event.registry_value_data	Optional	The registry key value data
registry_event.registry_value_name.data	Optional	Registry value name when an operation affects a registry key value
registry_event.registry_value_name.token	Optional	Tokenized Registry value name
registry_event.registry_value_index	Optional	Index into the registry key value names
registry_event.new_registry_path.data	Optional	New Registry key path when renaming a key
registry_event.new_registry_path.token	Optional	Tokenized new Registry key path when renaming a key
registry_event.registry_filepath.data	Optional	File path to the registry file when saving/loading the registry from/to file
registry_event.registry_filepath.token	Optional	Tokenized File path to the registry file when saving/loading the registry from/to file
registry_event.source_process_id.id	Required	Source process id
registry_event.source_process_id.unique_id	Required	Source process unique id
registry_event.source_thread_id.id	Required	Thread process id
registry_event.source_thread_id.unique_id	Required	Thread process unique id

Table A.2: Registry data fields

Registry information Information on the registry numbers in sensor events Information. Registry events will provide information containing what kind of registry access is asked for, which type is granted.

A.3 Process create event

Data field	Required/ Optional	Information
process_create_event.session_id	Required	Windows session id
process_create_event.command_line.data	Optional	The command line arguments
process_create_event.command_line.token	Required	The tokenized command line arguments
process_create_event.process_exe_path.token	Required	The tokenized path of the process executable
process_create_event.process_exe_path.data	Optional	The path of the started process executable
process_create_event.process_id.id	Required	The process id
process_create_event.process_id.unique_id	Required	The unique process id

Table A.3 – continues on next page

Table A.3 – continued from previous page

Data field	Required/ Optional	Information
process_create_event.source_process_id.id	Required	Parent process id
process_create_event.source_process_id.unique_id	Required	Unique parent process id
process_create_event.source_thread_id.id	Required	Parent thread id
process_create_event.source_thread_id.unique_id	Required	Unique parent thread id

Table A.3: Process create data fields

A.4 Process exit event

Data field	Required/ Optional	Information
process_exit_event.process_id.id	Required	Process id
process_exit_event.process_id.unique_id	Required	Unique process id

Table A.4: Process exit data fields

A.5 Thread create event

Data field	Required/ Optional	Information
thread_create_event.start_address	Required	The thread memory start address
thread_create_event.process_id.id	Required	The process id in which the new thread is created
thread_create_event.process_id.unique_id	Required	The unique process id in which the new thread is created
thread_create_event.source_process_id.id	Required	The source process id
thread_create_event.source_process_id.unique_id	Required	The unique source process id
thread_create_event.source_thread_id.id	Required	The source thread id
thread_create_event.source_thread_id.unique_id	Required	The unique source thread id
thread_create_event.thread_id.id	Required	the created thread id
thread_create_event.thread_id.unique_id	Required	The unique created thread id

Table A.5: Thread create data fields

A.6 Thread exit event

Data field	Required/ Optional	Information
thread_exit_event.thread_id.id	Required	The thread id
thread_exit_event.thread_id.unique_id	Required	The unique thread id

Table A.6: Thread exit data fields

A.7 Module load event

Data field	Required/ Optional	Information
module_load_event.entry_point	Required	The entrypoint of the module
module_load_event.image_base	Required	The base address in memory
module_load_event.image_size	Required	The size of the module in memory
module_load_event.module_hash	Required	The hash of the loaded module file
module_load_event.module_path.token	Required	The tokenized path of the loaded module
module_load_event.module_path.data	Optional	The path of the loaded module
module_load_event.source_process_id.id	Required	The source process id
module_load_event.source_process_id.unique_id	Required	The unique source process id
module_load_event.source_thread_id.id	Required	The source thread id
module_load_event.source_thread_id.unique_id	Required	The unique source thread id

Table A.7: Module load data fields

A.8 Object callback events

Data field	Required/ Optional	Information
ob_event.desired_access_mask	Required	Desired access mask
ob_event.granted_access_mask	Required	Granted access mask
ob_event.object_type	Required	The type of object the operation is performed on (none, thread or process)
ob_event.operation	Required	Kind of object operation logged
ob_event.operation_result	Required	The result of the operation
ob_event.object_id.id	Required	The id of the object defined in ob_event.object_type
ob_event.object_id.unique_id	Required	The unique id of the object defined in ob_event.object_type

Table A.8 – continues on next page

Table A.8 – continued from previous page

Data field	Required/ Optional	Information
ob_event.source_process_id.id	Required	The source process id
ob_event.source_process_id.unique_id	Required	The unique source process id
ob_event.source_thread_id.id	Required	The source thread id
ob_event.source_thread_id.unique_id	Required	The unique source thread id
ob_event.target_process_id.id	Required	The target process id
ob_event.target_process_id.unique_id	Required	The unique target process id

Table A.8: Object callbacks data fields

B

Process trees

B.1 Cut Process trees from clean datasets

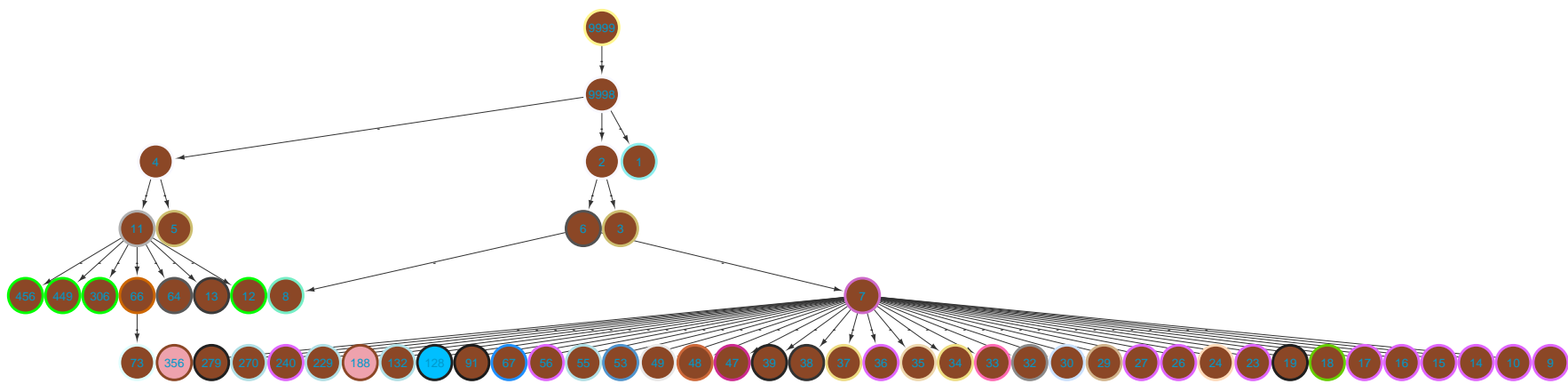


Figure B.1: First 5 levels of win8 1604 avond

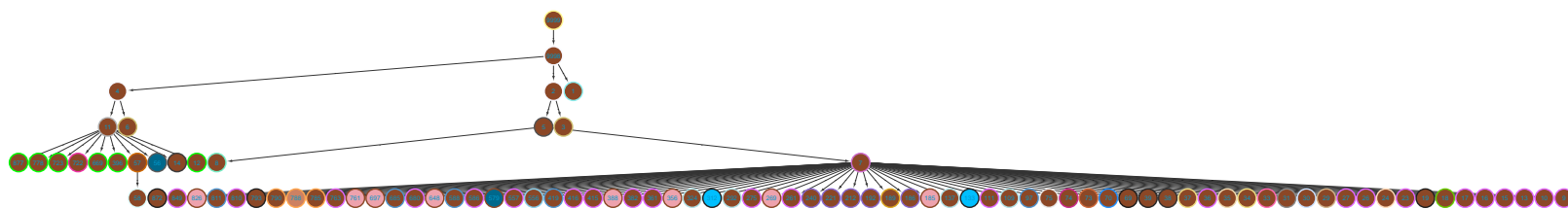


Figure B.2: First 5 levels of win8 1704

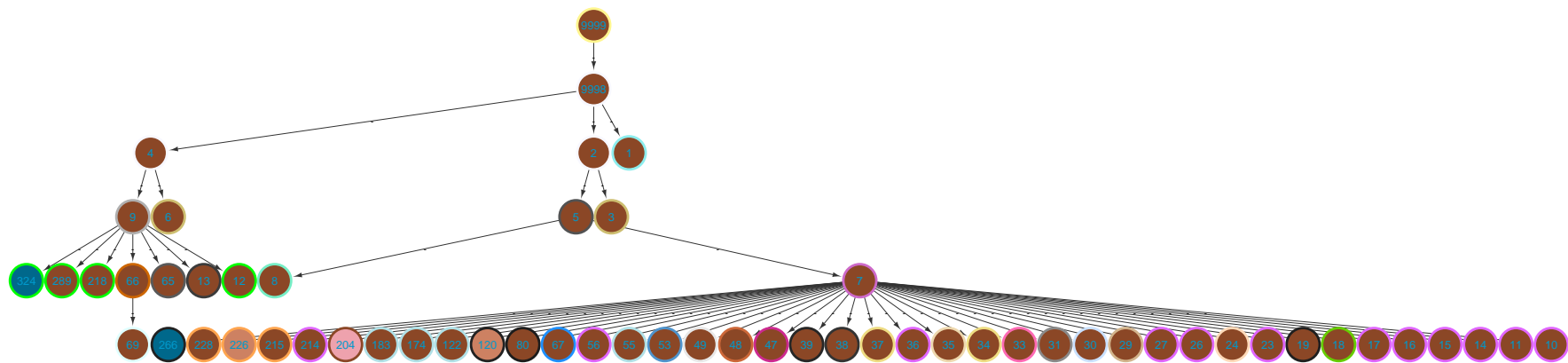


Figure B.3: First 5 levels of win8 1804 avond

B.2 Cut process trees from malware datasets

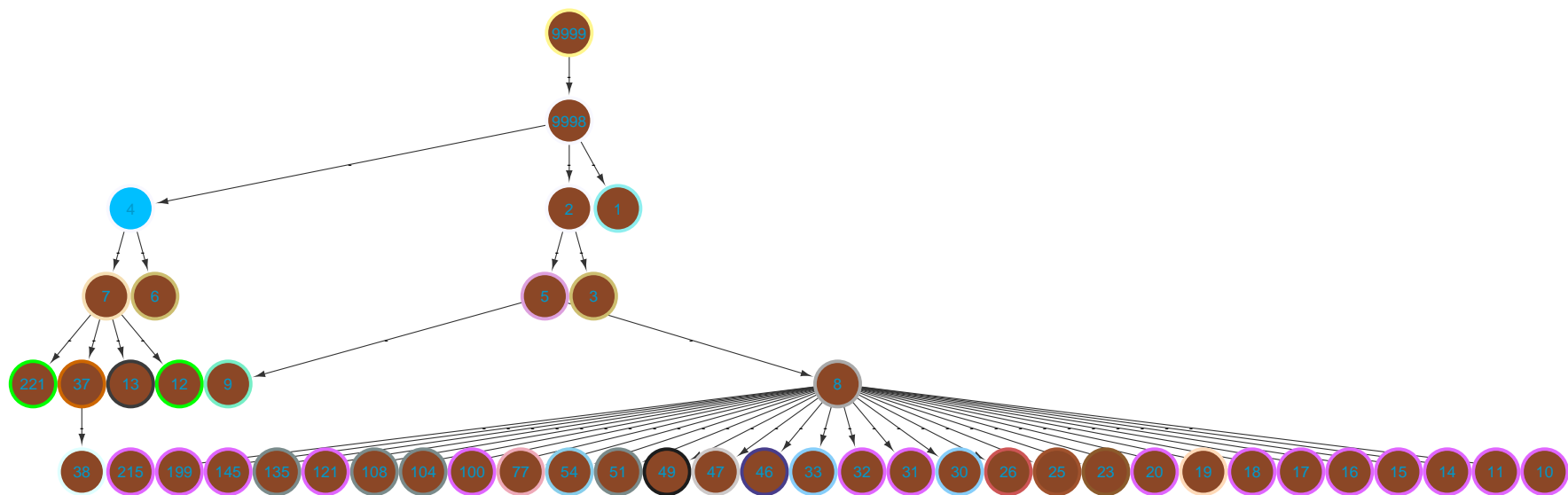


Figure B.4: First 5 levels of rat malware session 1

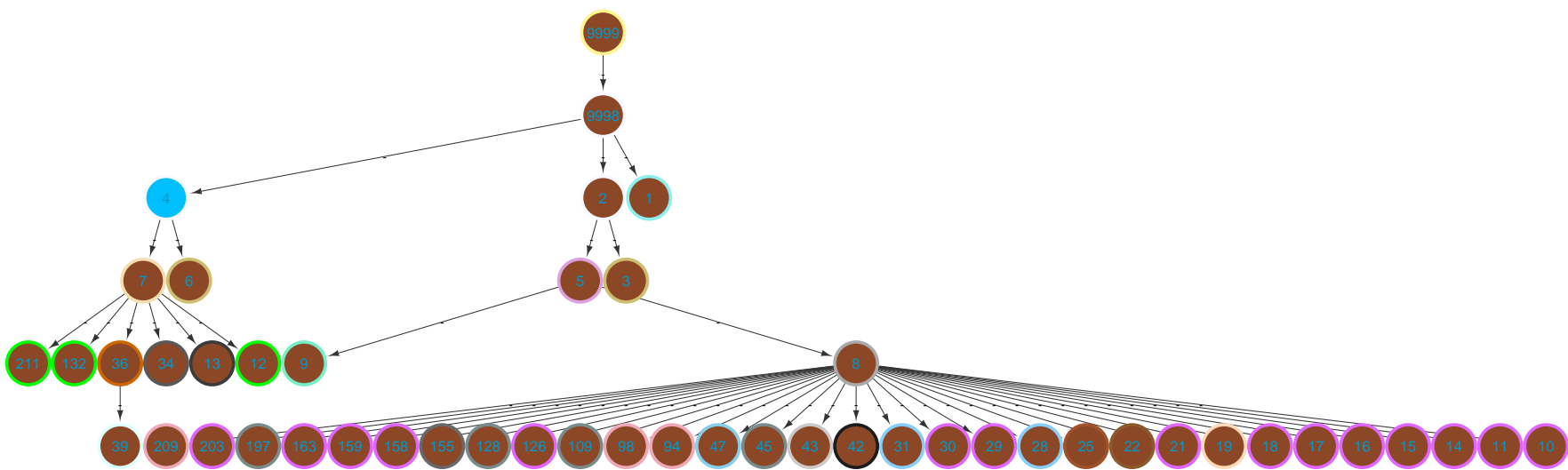


Figure B.5: First 5 levels of rat malware session 2

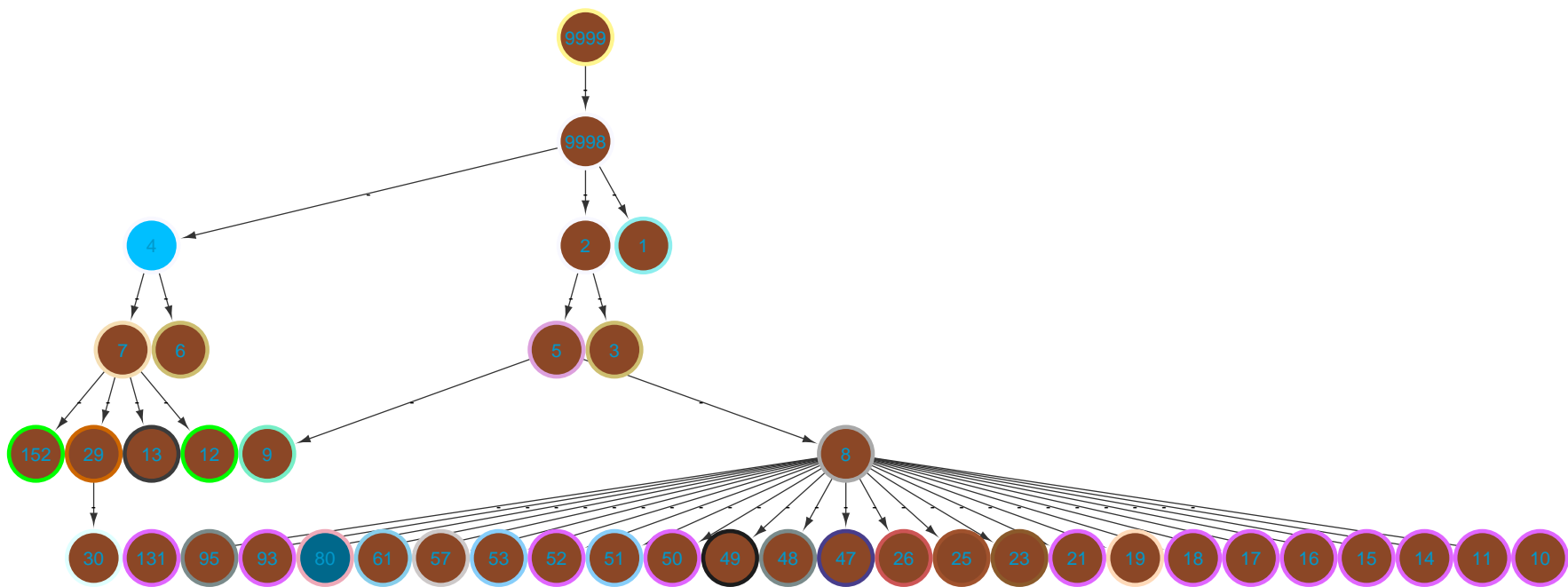


Figure B.6: First 5 levels of zeus malware session 1

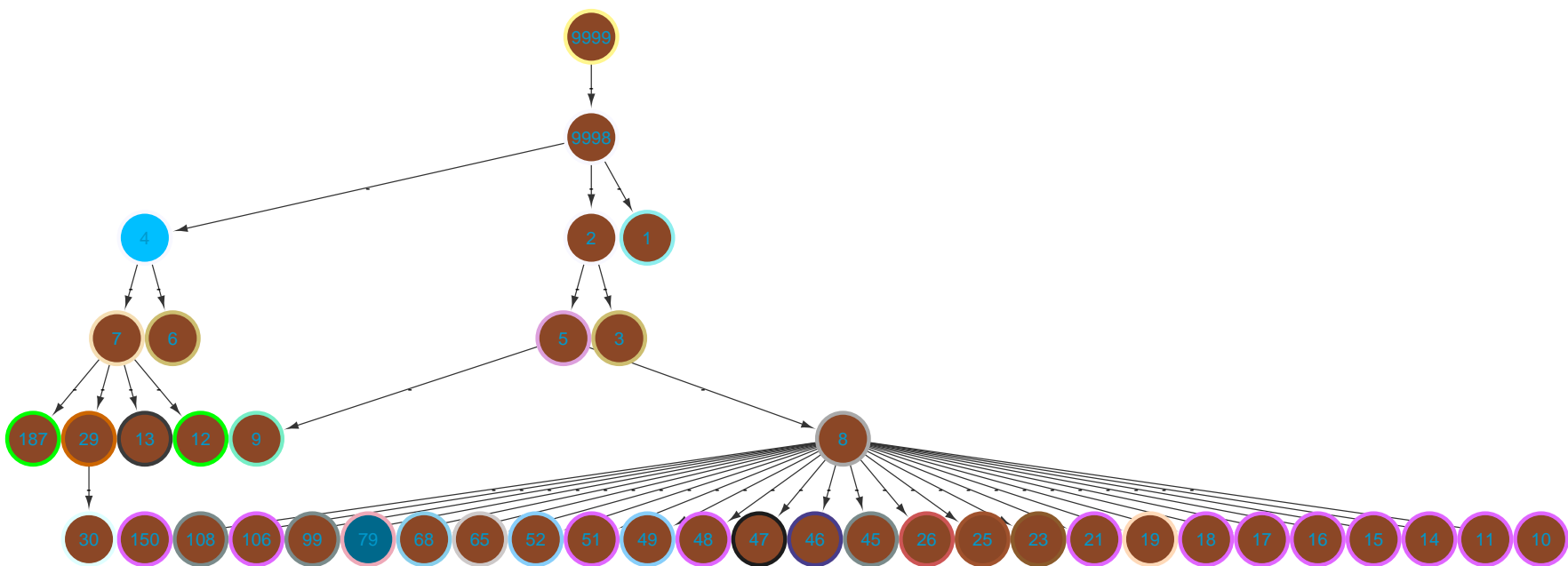


Figure B.7: First 5 levels of zeus malware session 2

B.3 Merged process trees

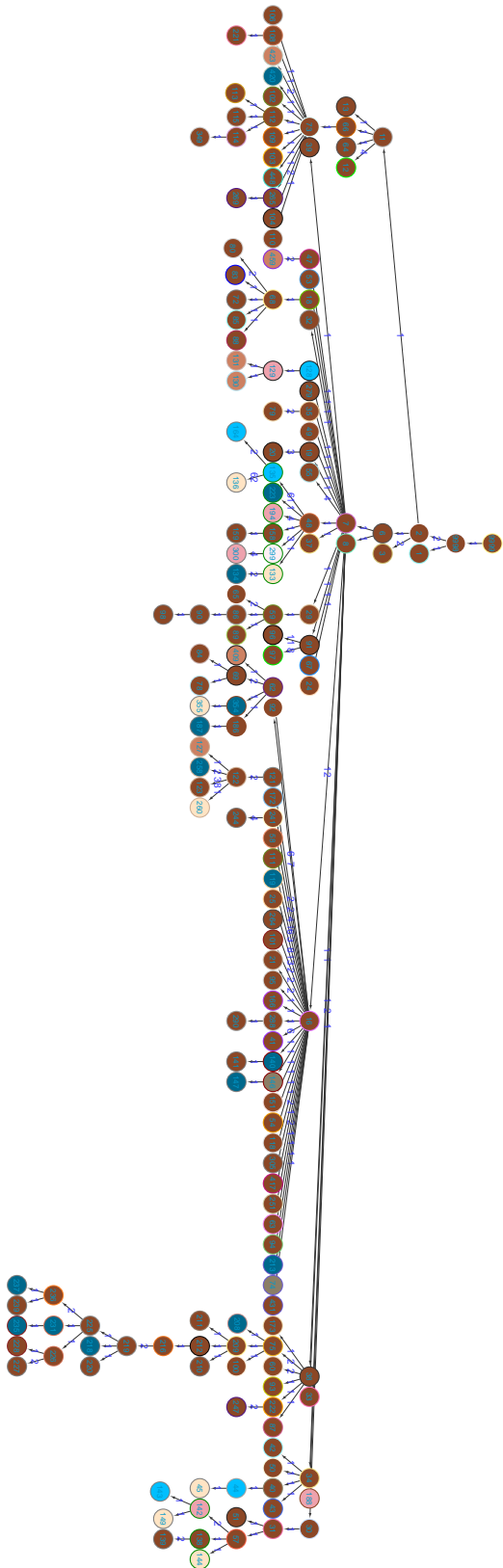


Figure B.8: 1604 avond merged tree

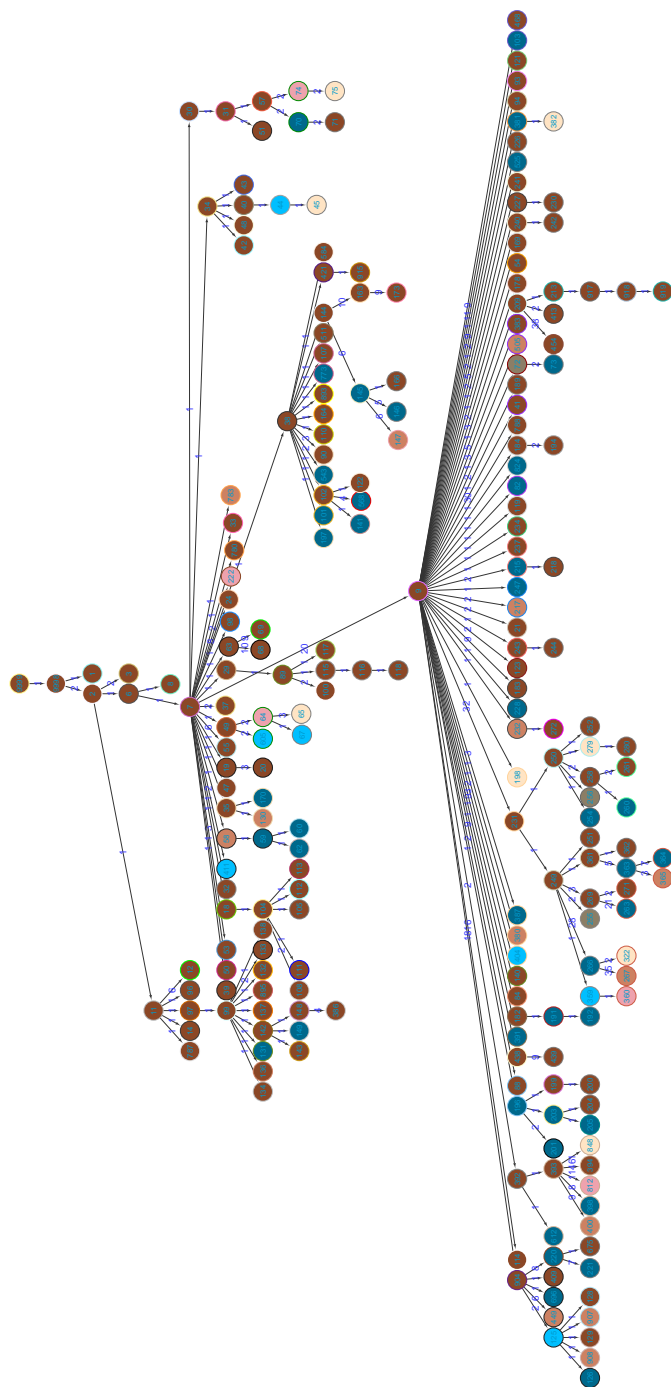


Figure B.9: 1604 merged tree

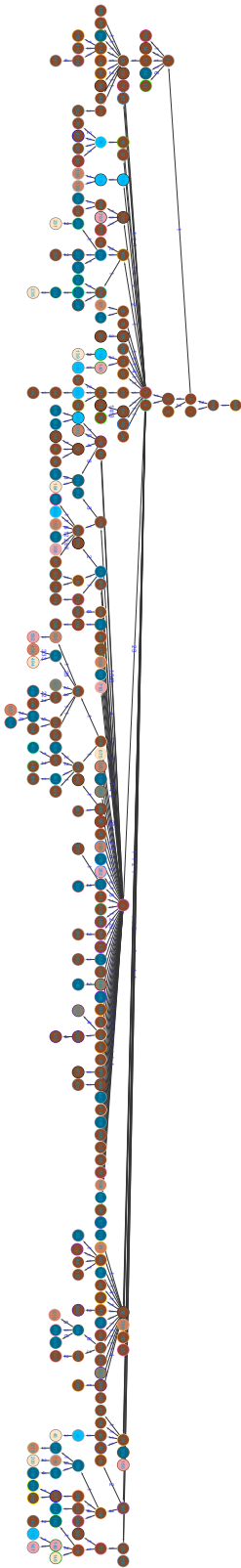


Figure B.10: 1704 merged tree

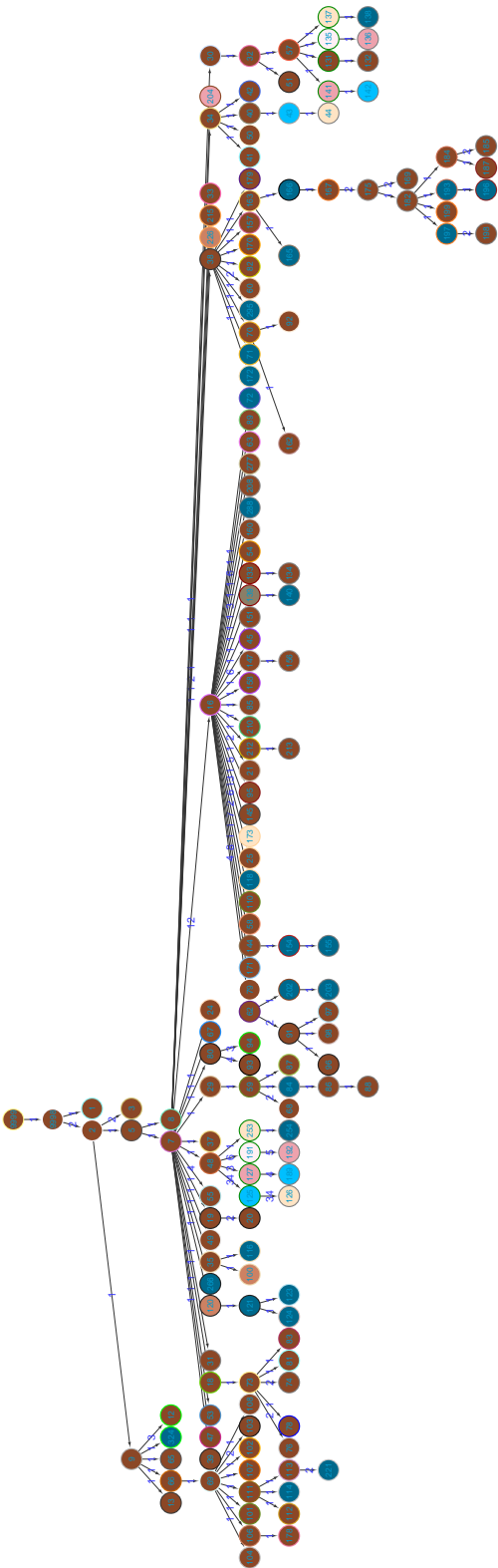


Figure B.11: 1804 merged tree

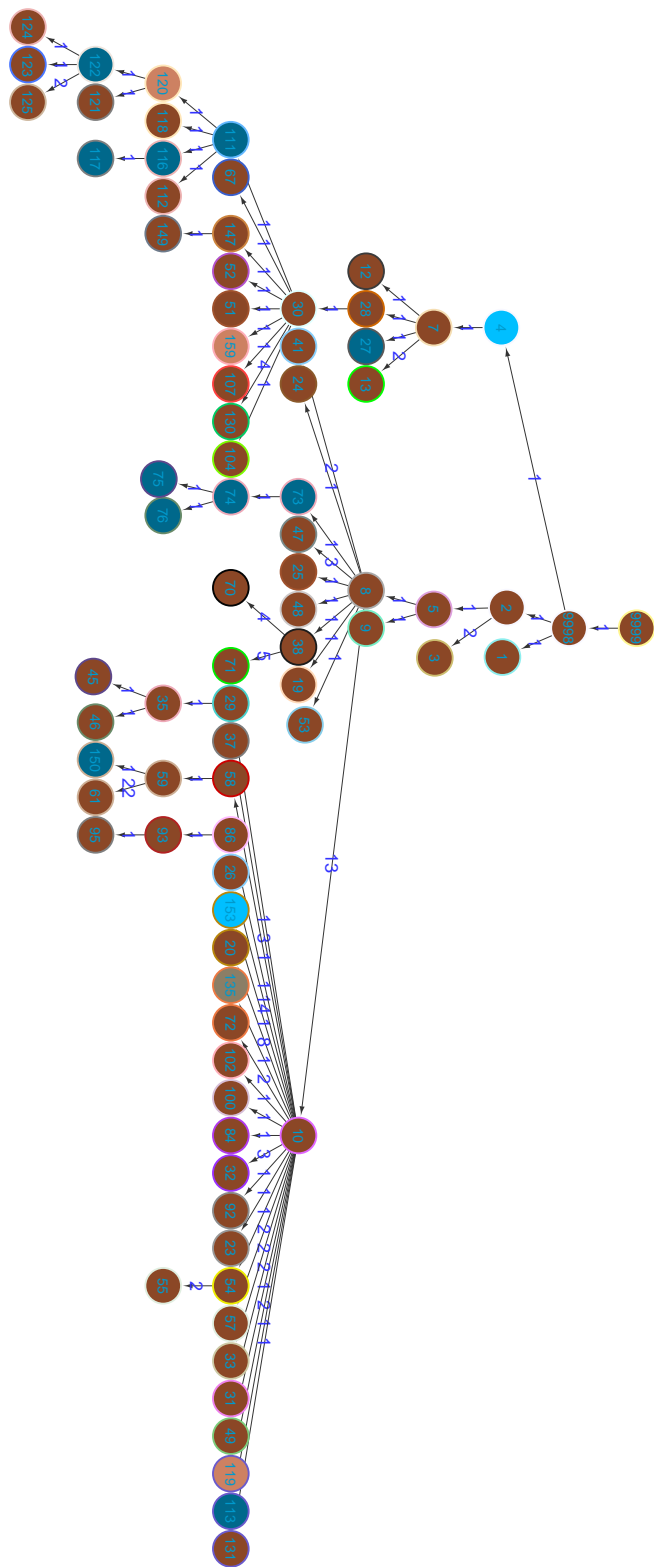


Figure B.12: bank malware merged tree

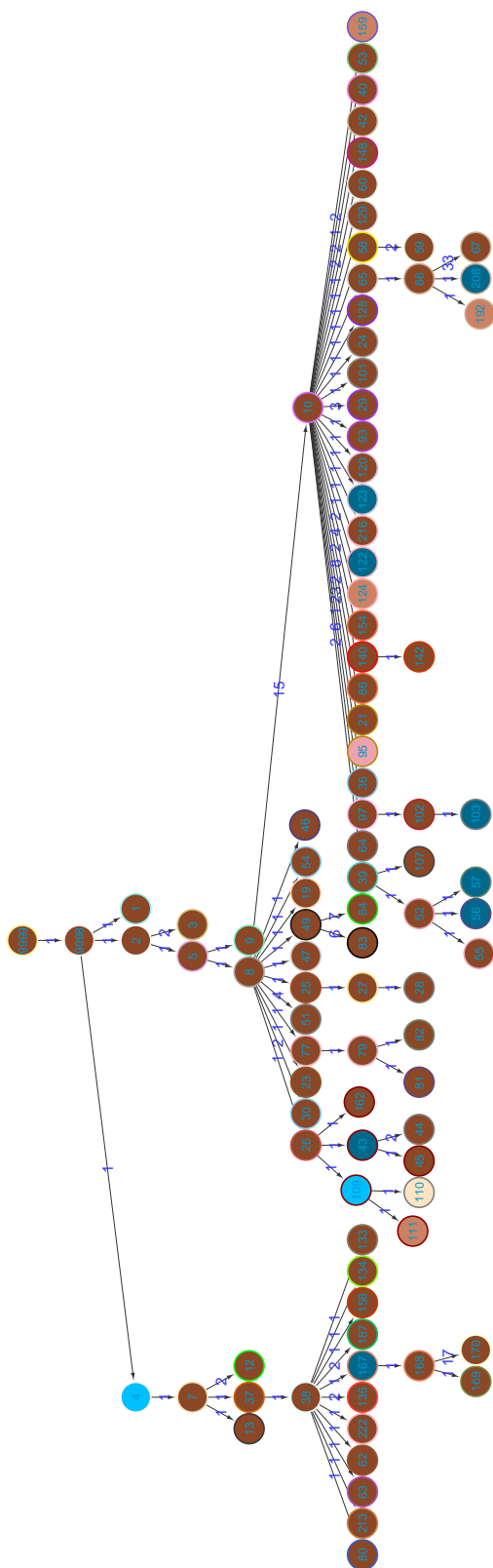


Figure B.13: rat malware session 1 merged tree

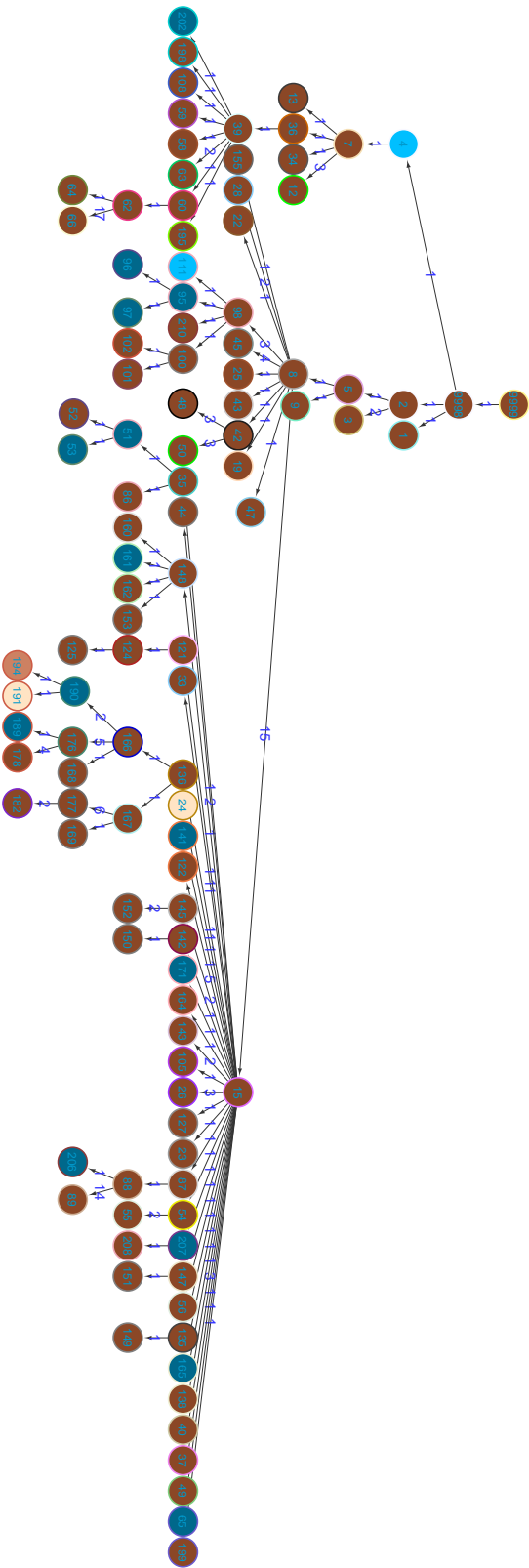


Figure B.14: rat malware session 2 merged tree

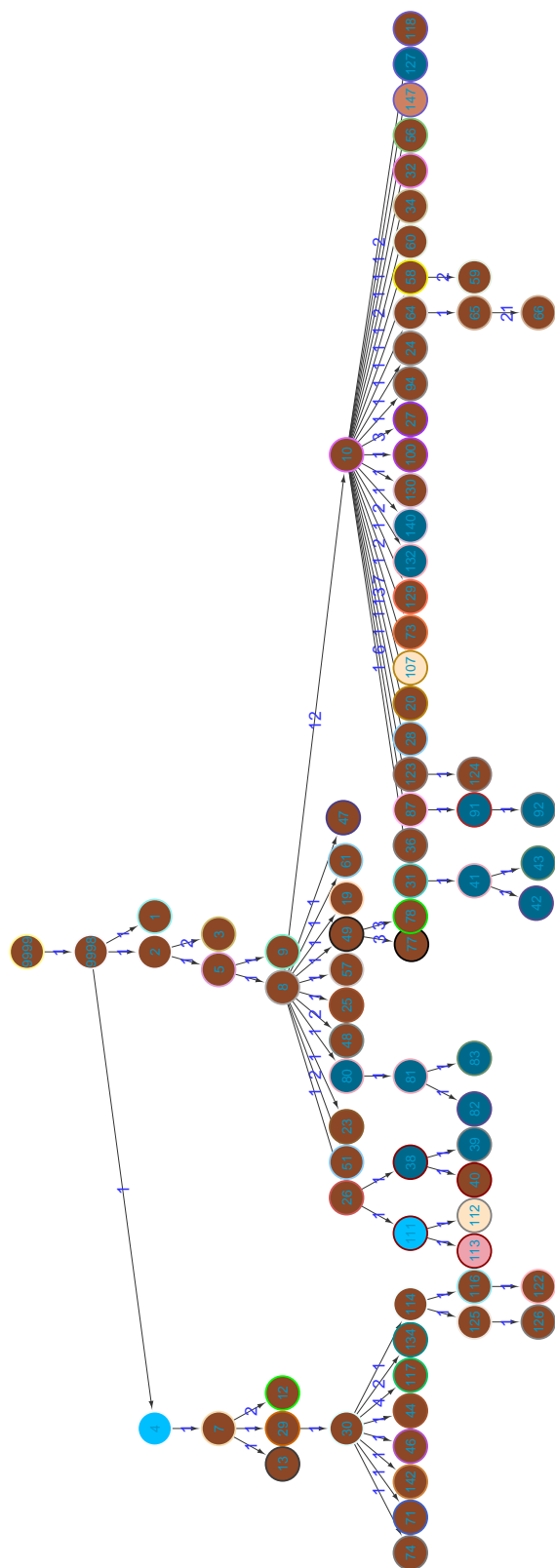


Figure B.15: Zeus malware session 1 merged tree



C

Process Activities

C.1 Process activities clean dataset

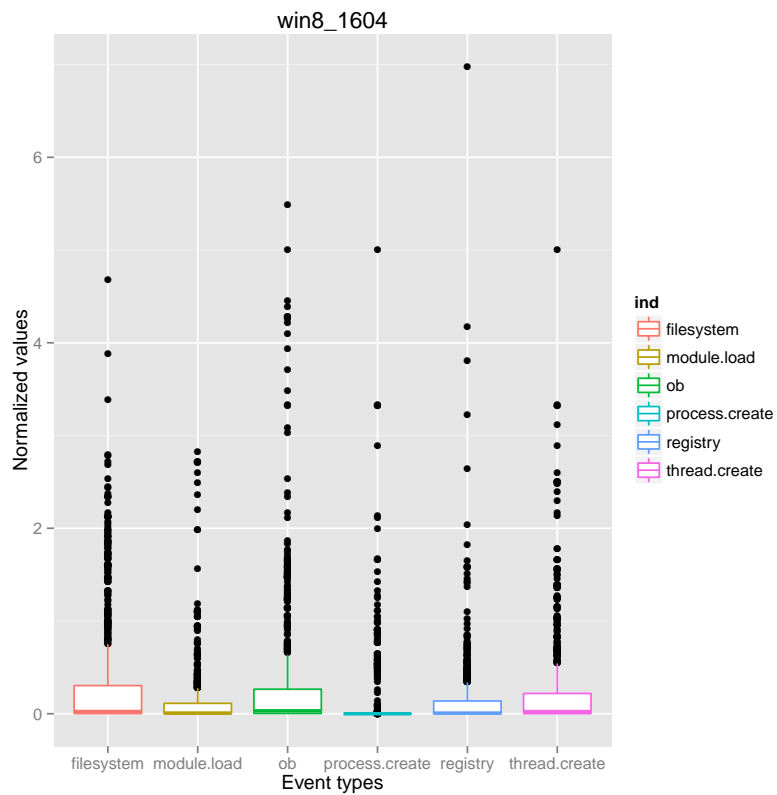


Figure C.1: Boxplot win 8 1604

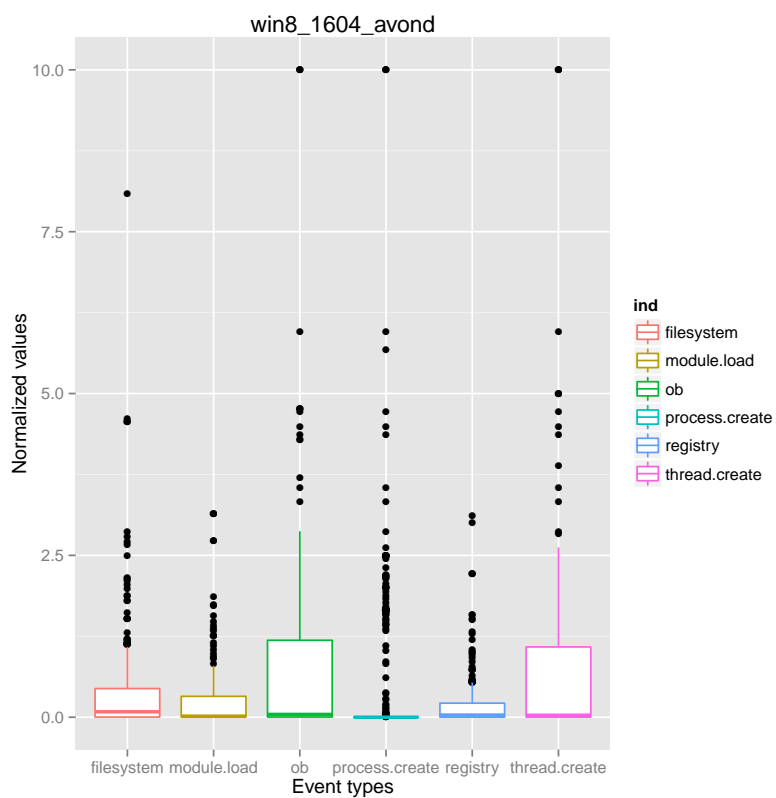


Figure C.2: Boxplot win 8 1604 avond

C.2 Process activities malware dataset

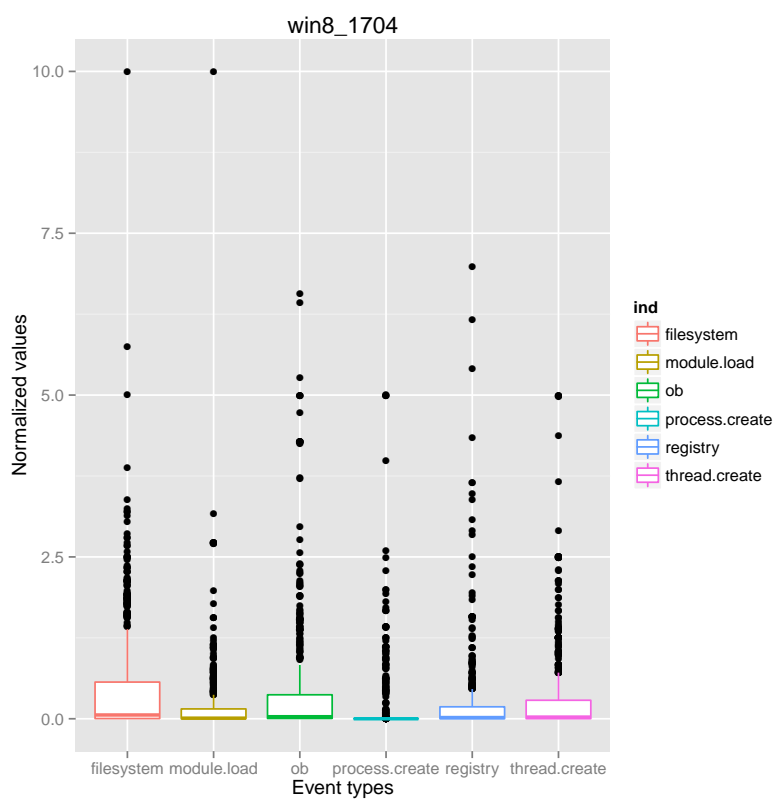


Figure C.3: Boxplot win 8 1704

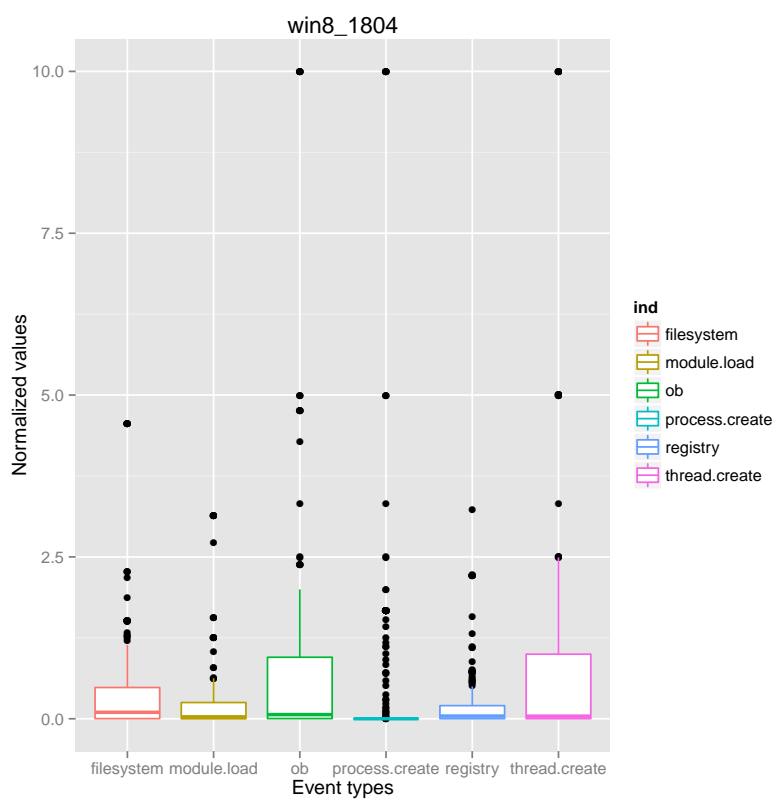


Figure C.4: Boxplot win 8 1804

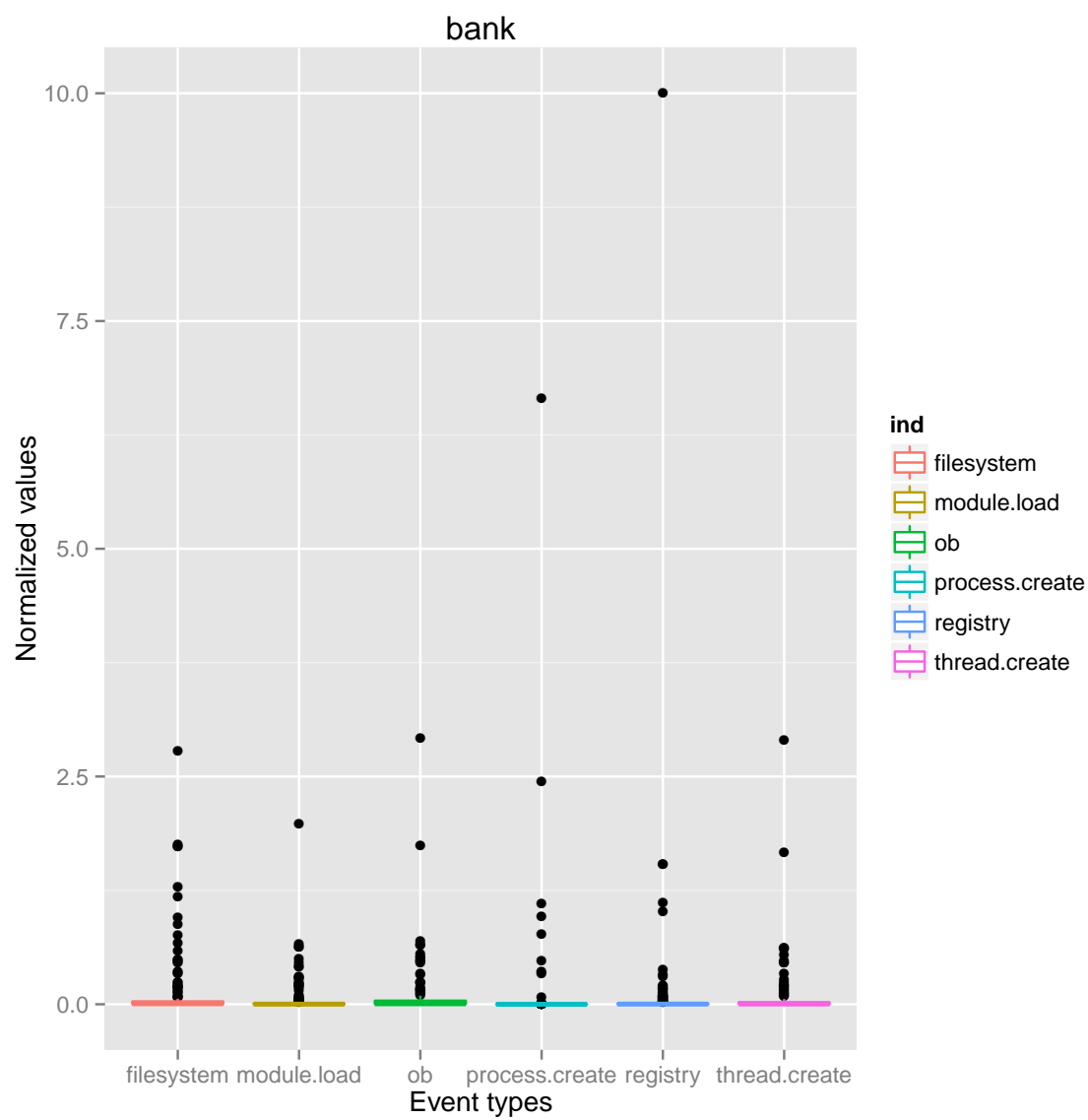


Figure C.5: Boxplot bank malware

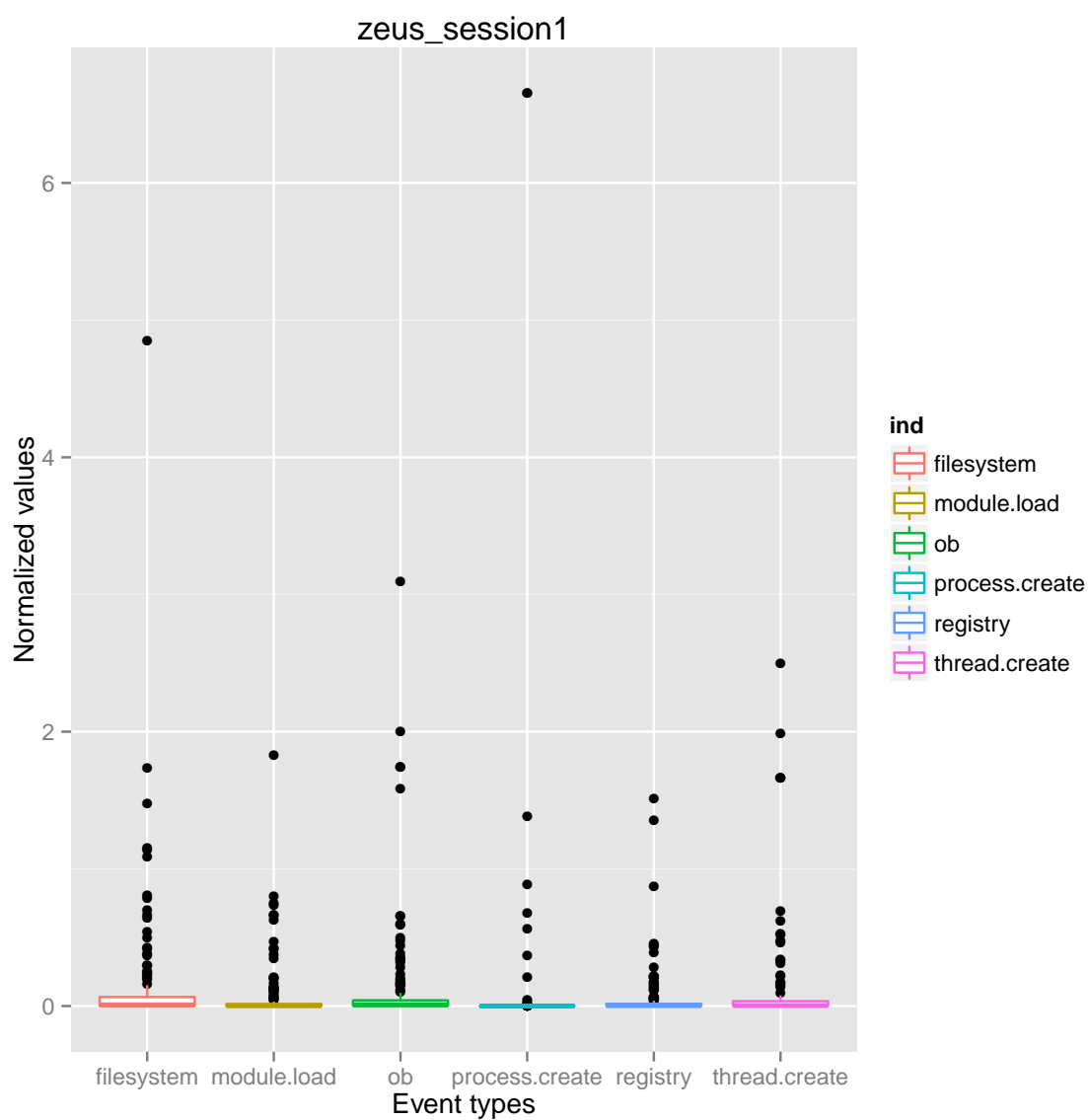


Figure C.6: Boxplot Zeus malware session 1

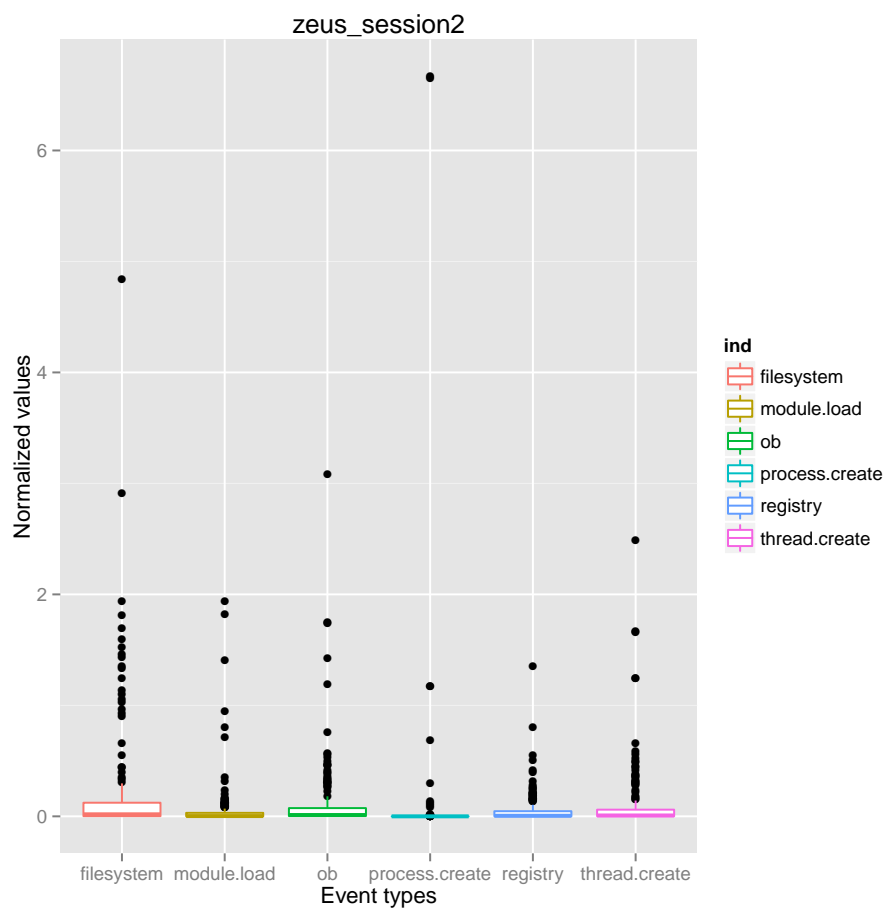


Figure C.7: Boxplot Zeus malware session 2

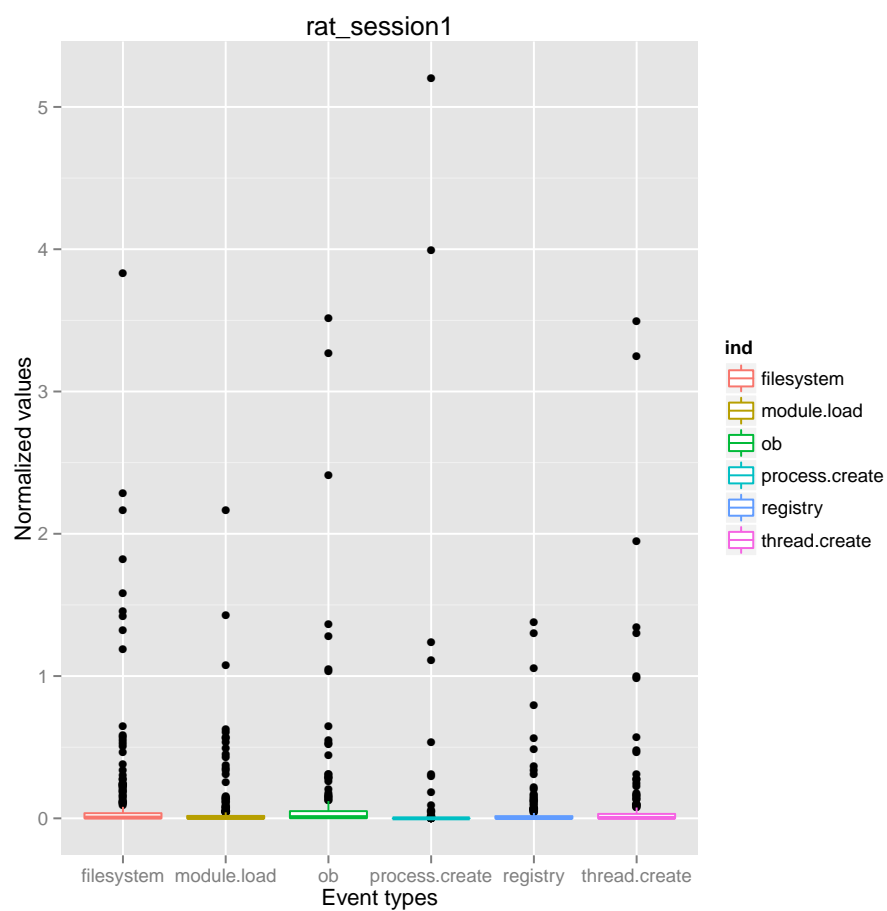


Figure C.8: Boxplot Rat malware session 1

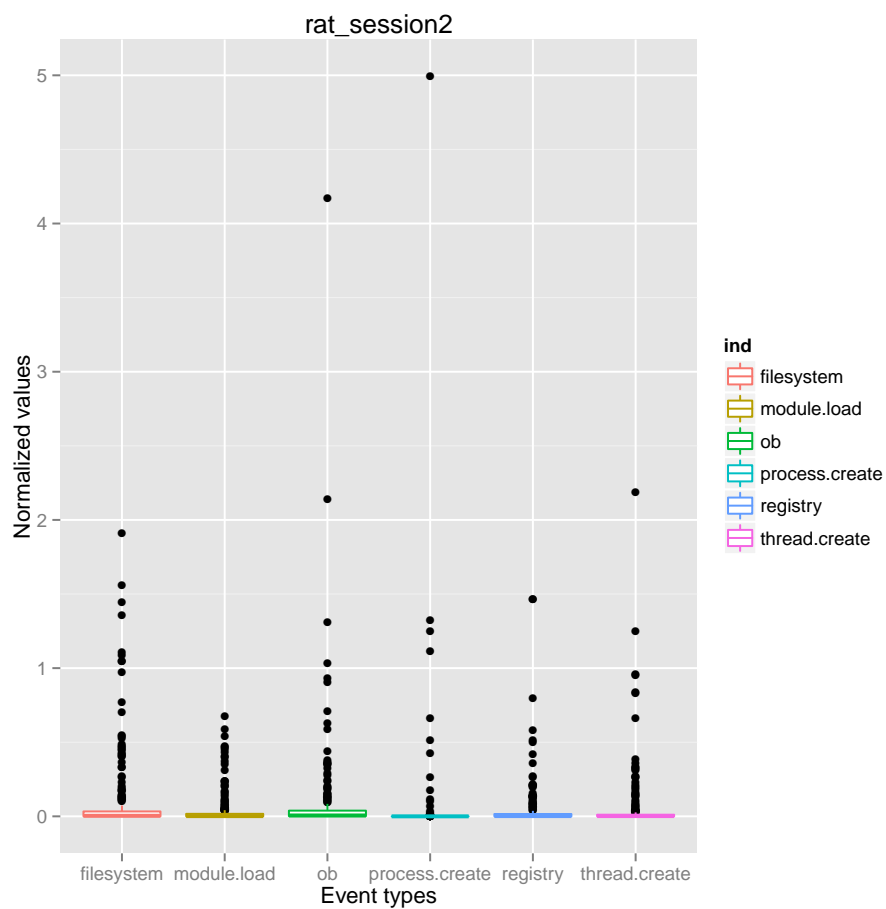


Figure C.9: Boxplot Rat malware session 2

D

Heatmaps

D.1 Heatmaps from the clean datasets

See seperate files for heatmaps

win8_1604_event_k8

0	0.07	0.384	0.242	0.289	0.259	460
0	0.07	0.382	0.241	0.288	0.258	761
0	0.07	0.387	0.243	0.292	0.261	755
0	0.069	0.377	0.237	0.284	0.254	718
0	0.068	0.376	0.236	0.283	0.253	766
0	0.068	0.372	0.238	0.28	0.251	455
0	0.066	0.363	0.228	0.274	0.245	723
0	0.066	0.363	0.228	0.273	0.244	739
0	0.065	0.357	0.224	0.269	0.24	733
0	0.063	0.349	0.221	0.263	0.235	703
0	0.06	0.328	0.206	0.247	0.221	750
0	0.059	0.327	0.206	0.246	0.22	713
0	0.061	0.336	0.211	0.253	0.226	708
0	0.058	0.32	0.201	0.241	0.215	468
0	0.058	0.318	0.2	0.239	0.214	749
0	0.072	0.397	0.25	0.299	0.267	728
0	0.072	0.394	0.248	0.297	0.265	478
0	0.075	0.411	0.259	0.31	0.277	473
0	0.079	0.436	0.274	0.328	0.293	465
0.085	0.074	0.422	0.401	0.149	0.171	824
0	0.047	0.282	0.23	0.172	0.188	604
0	0.043	0.258	0.209	0.157	0.172	219
0.067	0.047	0.291	0.232	0.152	0.193	199
0	0.015	0.35	0.277	0.1	0.146	919
0	0.05	0.315	0.095	0.125	0.135	738
0	0.141	0.173	0.211	0	0.018	790
0	0.122	0.15	0.183	0	0.016	386
0	0.164	0.201	0.241	0.021	0.021	546
0	0.002	0.272	0.304	0.006	0.007	272
0	0.244	0.298	0.366	0	0.032	851
0.023	0.019	0.114	0.438	0.064	0.085	918
0.022	0.018	0.174	0.399	0.028	0.031	182
0	0.237	0.179	0.499	0.227	0.281	574
0	0.232	0.181	0.488	0.222	0.275	576
0	0.227	0.17	0.477	0.217	0.269	580
0	0.296	0.205	0.477	0.208	0.238	60
0	0.176	0.215	0.561	0.244	0.22	202
0	0.183	0.138	0.385	0.175	0.217	568
0	0.133	0.122	0.375	0.176	0.218	528
0	0.127	0.192	0.402	0.267	0.255	201
0	0.211	0.086	0.429	0.103	0.137	789
0	0.19	0.077	0.385	0.093	0.123	383
0	0.168	0.055	0.429	0.081	0.092	543
0	0.251	0.101	0.509	0.122	0.163	543
0	0.085	0.153	0.549	0.27	0.334	265
0	0.047	0.096	0.543	0.219	0.276	399
0	0.048	0.099	0.556	0.184	0.388	792
0	0.04	0.083	0.467	0.154	0.31	601
0	0.036	0.073	0.414	0.137	0.39	881
0	0.017	0.034	0.193	0.064	0.189	882
0	0.015	0.031	0.176	0.09	0.176	644
0	0.017	0.035	0.199	0.08	0.15	651
0	0.016	0.033	0.185	0.068	0.146	837
0	0.02	0.041	0.231	0.076	0.177	860
0	0.019	0.039	0.219	0.088	0.157	600
0	0.02	0.042	0.235	0.078	0.14	250
0	0.084	0.029	0.197	0.138	0.171	777
0	0.073	0.031	0.173	0.121	0.161	524
0	0.025	0.052	0.297	0.098	0.217	802
0	0.025	0.051	0.286	0.105	0.206	687
0	0.019	0.04	0.226	0.116	0.218	643
0	0.02	0.04	0.228	0.075	0.223	884
0	0.144	0.094	0.317	0.119	0.136	129
0	0.157	0.105	0.263	0.104	0.119	128
0	0.03	0.062	0.349	0.115	0.22	680
0	0.028	0.059	0.331	0.109	0.22	683
0	0.031	0.064	0.36	0.119	0.201	545
0	0.032	0.066	0.37	0.122	0.259	625
0	0.031	0.063	0.357	0.105	0.162	420
0	0.079	0.053	0.185	0.258	0.312	17
0	0.077	0.052	0.181	0.253	0.305	548
0	0.072	0.048	0.168	0.234	0.282	791
0	0.069	0.046	0.162	0.226	0.273	853
0	0.093	0.072	0.208	0.277	0.264	204
0	0.062	0.052	0.175	0.271	0.248	788
0	0.093	0.135	0.242	0.231	0.264	195
0	0.067	0.12	0.206	0.198	0.245	383
0	0.116	0.04	0.274	0.192	0.238	814
0	0.115	0.04	0.271	0.19	0.235	821
0	0.112	0.039	0.264	0.185	0.229	225
0	0.116	0.056	0.268	0.198	0.214	522
0	0.104	0.036	0.246	0.172	0.213	841
0	0.125	0.043	0.293	0.206	0.255	621
0	0.119	0.092	0.234	0.357	0.374	161
0	0.119	0.092	0.234	0.357	0.374	128
0	0.117	0.091	0.254	0.35	0.367	847
0	0.113	0.071	0.3	0.357	0.367	166
0	0.104	0.081	0.204	0.312	0.327	398
0	0.161	0.077	0.364	0.21	0.481	77
0	0.104	0.06	0.107	0.416	0.396	675
0	0.09	0.05	0.109	0.357	0.34	71
0	0.07	0.04	0.071	0.277	0.264	467
0	0.209	0	0	0	0.476	464
0	0.209	0	0	0	0.476	472
0	0.209	0	0	0	0.476	707
0	0.209	0	0	0	0.476	732
0	0.209	0	0	0	0.476	754
0	0.209	0	0	0	0.476	760
0	0.208	0	0	0	0.473	749
0	0.208	0	0	0	0.473	744
0	0.208	0	0	0	0.473	477
0	0.06	0.315	0.163	0.095	0.621	126
0	0.007	0.005	0.017	0.03	0.031	453
0	0.007	0.005	0.017	0.03	0.031	682
0	0.007	0.005	0.017	0.03	0.031	619
0	0.007	0.005	0.017	0.03	0.031	483
0	0.007	0.005	0.017	0.03	0.031	538
0	0.007	0.005	0.017	0.03	0.031	512
0	0.007	0.005	0.017	0.03	0.031	527
0	0.007	0.005	0.017	0.03	0.031	809
0	0.007	0.005	0.017	0.03	0.031	633
0	0.007	0.005	0.017	0.03	0.031	567
0	0.007	0.005	0.017	0.03	0.031	569
0	0.007	0.005	0.017	0.03	0.031	529
0	0.007	0.005	0.017	0.03	0.031	557

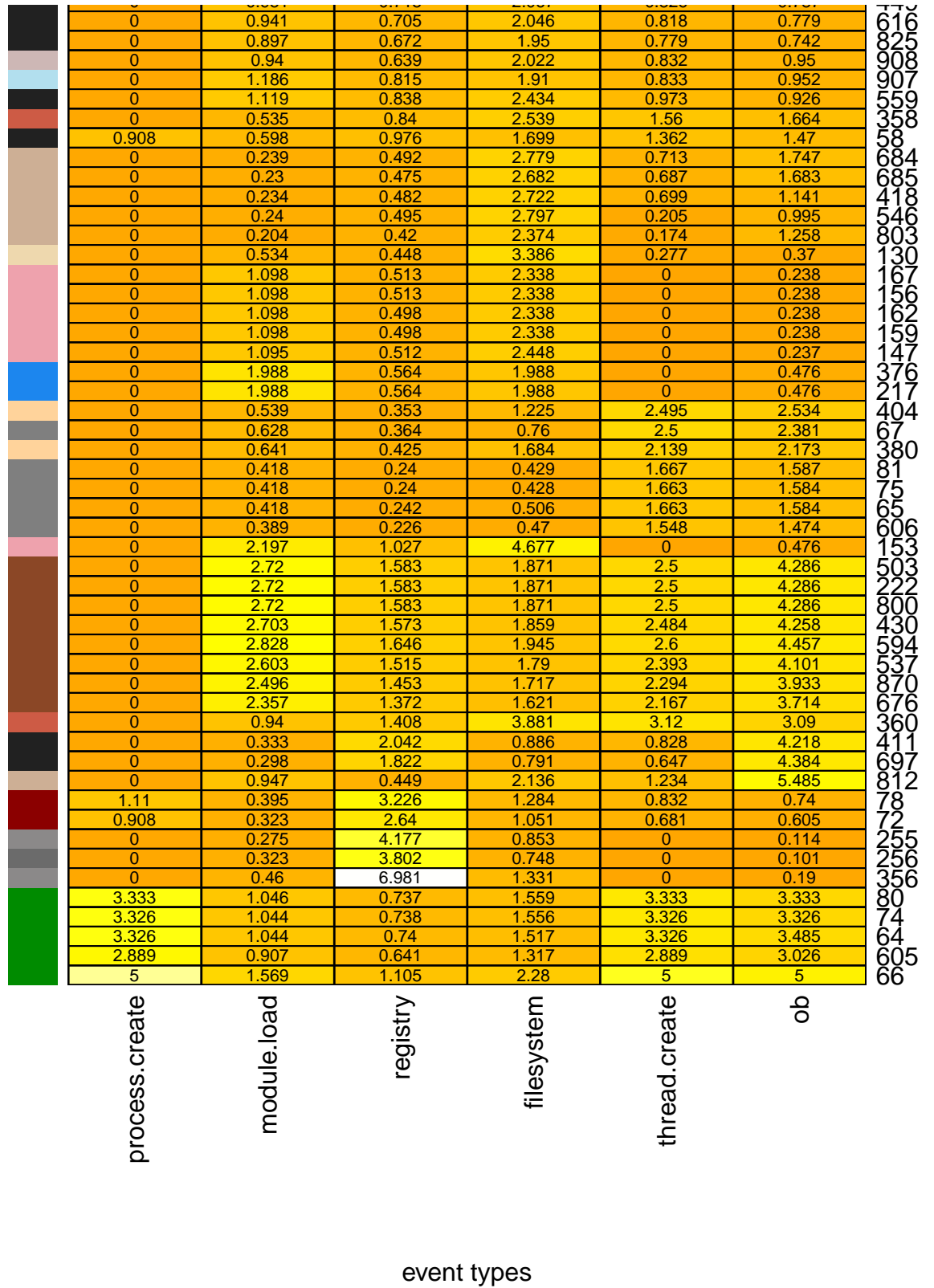


Figure D.2: Heatmap win8 1604 dataset split test

win8_1604_avond_event_k8

0.759	0.368	0.523	1.665	1.665	1.665	275
0.759	0.368	0.523	1.665	1.665	1.665	200
0.759	0.368	0.523	1.665	1.665	1.665	368
0.759	0.368	0.523	1.665	1.665	1.665	372
0.759	0.368	0.523	1.665	1.665	1.665	387
0.759	0.368	0.523	1.665	1.665	1.665	402
0.759	0.368	0.523	1.665	1.665	1.665	404
0.759	0.368	0.523	1.665	1.665	1.665	410
0.759	0.368	0.523	1.665	1.665	1.665	293
0.759	0.368	0.523	1.665	1.665	1.665	198
0.759	0.368	0.523	1.665	1.665	1.665	376
0.759	0.368	0.523	1.665	1.665	1.665	378
0.759	0.368	0.523	1.665	1.665	1.665	381
0.759	0.368	0.523	1.665	1.665	1.665	390
0.758	0.368	0.522	1.663	1.663	1.663	370
0.758	0.368	0.522	1.663	1.663	1.663	366
0.758	0.368	0.522	1.663	1.663	1.663	291
0.759	0.372	0.523	1.665	1.665	1.665	399
0.759	0.371	0.523	1.665	1.665	1.665	273
0.753	0.365	0.518	1.651	1.651	1.651	343
0.748	0.363	0.515	1.64	1.64	1.64	329
0.777	0.376	0.534	1.703	1.703	1.703	331
0.736	0.357	0.506	1.613	1.613	1.613	325
0.733	0.355	0.505	1.608	1.608	1.608	323
0.724	0.351	0.498	1.587	1.587	1.587	333
0.722	0.35	0.497	1.584	1.584	1.584	183
0.694	0.336	0.478	1.522	1.522	1.522	434
0.689	0.334	0.474	1.512	1.512	1.512	192
0.675	0.327	0.465	1.48	1.48	1.48	438
0.652	0.322	0.449	1.43	1.43	1.43	432
0.65	0.315	0.448	1.426	1.426	1.426	360
0.624	0.302	0.429	1.368	1.368	1.368	196
0.911	0.441	0.627	1.997	1.997	1.997	358
0.911	0.441	0.627	1.997	1.997	1.997	295
0.911	0.441	0.627	1.997	1.997	1.997	392
0.918	0.445	0.632	2.013	2.013	2.013	327
0.94	0.455	0.647	2.061	2.061	2.061	321
0.856	0.419	0.589	1.877	1.877	1.877	181
0.837	0.406	0.576	1.835	1.835	1.835	436
0.884	0.428	0.608	1.938	1.938	1.938	307
0.807	0.391	0.555	1.769	1.769	1.769	335
0.992	0.464	0.783	2.496	1.872	1.545	44
0.998	0.484	0.687	2.188	2.188	2.188	315
0.998	0.484	0.687	2.188	2.188	2.188	345
1.006	0.488	0.693	2.207	2.207	2.207	319
0.987	0.478	0.679	2.164	2.164	2.164	135
0.983	0.476	0.676	2.155	2.155	2.155	311
0.973	0.472	0.67	2.134	2.134	2.134	339
1.052	0.51	0.724	2.308	2.308	2.308	341
2.668	1.529	0.94	1.426	2.139	2.241	128
0.379	0.182	0.313	0	1.248	1.189	384
0.379	0.182	0.313	0	1.248	1.189	203
0.379	0.182	0.313	0	1.248	1.189	413
0.379	0.182	0.313	0	1.248	1.189	375
0.379	0.182	0.313	0	1.248	1.189	296
0.379	0.182	0.313	0	1.248	1.189	393
0.379	0.182	0.313	0	1.246	1.187	359
0.321	0.176	0.313	0	1.248	1.189	355
0.391	0.188	0.323	0	1.287	1.226	308
0.388	0.186	0.321	0	1.277	1.216	136
0.395	0.19	0.326	0	1.3	1.238	322
0.412	0.197	0.34	0	1.354	1.29	340
0.412	0.197	0.34	0	1.354	1.29	328
0.426	0.205	0.352	0	1.403	1.336	346
0.453	0.217	0.374	0	1.489	1.418	312
0.441	0.211	0.364	0	1.45	1.381	316
0.469	0.225	0.387	0	1.542	1.468	320
0.507	0.243	0.418	0	1.667	1.587	284
0.507	0.243	0.418	0	1.667	1.587	234
0.507	0.243	0.418	0	1.667	1.587	302
0.506	0.242	0.418	0	1.667	1.587	407

D.1.1 Analyzing the heatmaps

win8_1704_event_k8

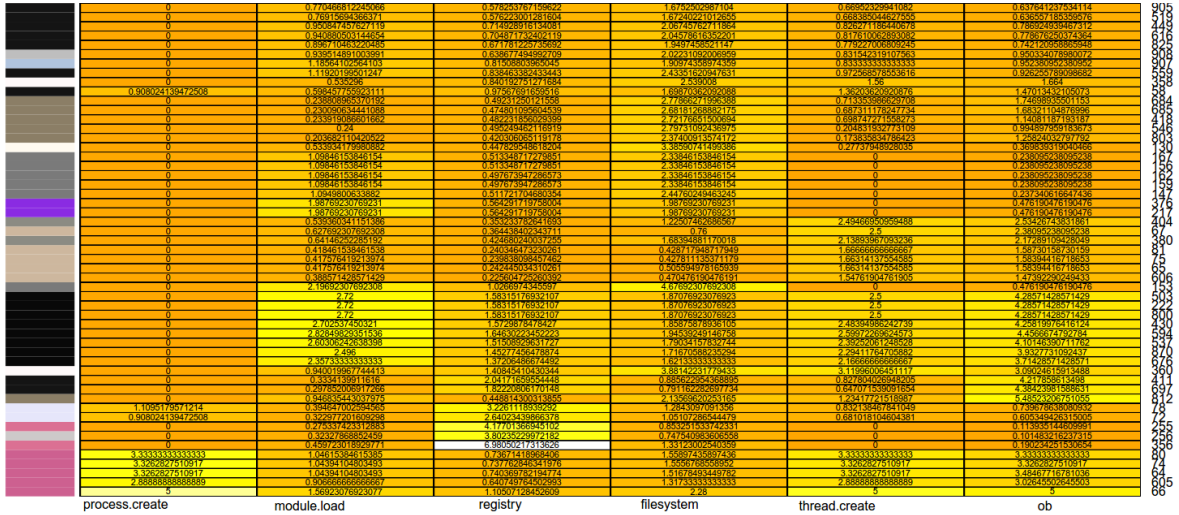
0	0.007	0.005	0.017	0.03	0.031	836
0	0.007	0.005	0.017	0.03	0.031	822
0	0.007	0.005	0.017	0.03	0.031	820
0	0.007	0.005	0.017	0.03	0.031	585
0	0.007	0.005	0.017	0.03	0.031	305
0	0.007	0.005	0.017	0.03	0.031	596
0	0.007	0.005	0.017	0.03	0.031	377
0	0.007	0.005	0.017	0.03	0.031	653
0	0.007	0.005	0.017	0.03	0.031	730
0	0.007	0.005	0.017	0.03	0.031	757
0	0.007	0.005	0.017	0.03	0.031	394
0	0.007	0.005	0.017	0.03	0.031	351
0	0.007	0.005	0.017	0.03	0.031	384
0	0.007	0.005	0.017	0.03	0.031	865
0	0.007	0.005	0.017	0.03	0.031	679
0	0.007	0.005	0.017	0.03	0.031	367
0	0.007	0.005	0.017	0.03	0.031	375
0	0.007	0.005	0.017	0.03	0.031	559
0	0.007	0.005	0.017	0.03	0.031	347
0	0.007	0.005	0.017	0.03	0.031	332
0	0.007	0.005	0.017	0.03	0.031	712
0	0.007	0.005	0.017	0.03	0.031	771
0	0.007	0.005	0.017	0.03	0.031	815
0	0.007	0.005	0.017	0.03	0.031	641
0	0.007	0.005	0.017	0.03	0.031	671
0	0.007	0.005	0.017	0.03	0.031	630
0	0.007	0.005	0.017	0.03	0.031	362
0	0.007	0.005	0.017	0.03	0.031	834
0	0.007	0.005	0.017	0.03	0.031	693
0	0.007	0.005	0.017	0.03	0.031	637
0	0.007	0.005	0.017	0.03	0.031	659
0	0.007	0.005	0.017	0.03	0.031	606
0	0.007	0.005	0.017	0.03	0.031	566
0	0.007	0.005	0.017	0.03	0.031	803
0	0.007	0.005	0.017	0.03	0.031	170
0	0.007	0.005	0.017	0.029	0.031	668
0	0.007	0.005	0.017	0.029	0.031	391
0	0.007	0.005	0.017	0.029	0.031	575
0	0.007	0.005	0.017	0.03	0.031	850
0	0.007	0.005	0.017	0.03	0.031	326
0	0.007	0.005	0.017	0.03	0.031	373
0	0.007	0.005	0.017	0.03	0.031	739
0	0.007	0.005	0.017	0.029	0.031	779
0	0.007	0.005	0.017	0.029	0.031	552
0	0.007	0.005	0.017	0.03	0.032	213
0	0.007	0.005	0.017	0.03	0.032	241
0	0.007	0.005	0.017	0.03	0.032	187
0	0.007	0.005	0.017	0.03	0.031	193
0	0.007	0.005	0.017	0.03	0.031	287
0	0.007	0.005	0.017	0.03	0.03	704
0	0.007	0.005	0.017	0.03	0.03	721
0	0.007	0.005	0.017	0.029	0.03	90
0	0.008	0.007	0.018	0.03	0.03	191
0	0.007	0.005	0.016	0.031	0.033	46
0	0.009	0.006	0.021	0.035	0.035	687
0	0.009	0.007	0.023	0.03	0.031	218
0	0.011	0.014	0.019	0.036	0.036	104
0	0.011	0.015	0.019	0.031	0.032	65
0	0.008	0.006	0.017	0.019	0.023	88
0	0.007	0.005	0.016	0.021	0.022	798
0	0.008	0.014	0.02	0.023	0.025	847
0.03	0.012	0.005	0.02	0.03	0.034	240
0.024	0.01	0.004	0.019	0.024	0.028	186
0	0.016	0.009	0.033	0.043	0.046	876
0	0.009	0.01	0.025	0.042	0.047	605
0	0.013	0.007	0.015	0.05	0.053	555
0	0.019	0.01	0.039	0.025	0.044	25
0	0.003	0.002	0.005	0.007	0.008	327
0	0.003	0.002	0.005	0.007	0.008	154
0	0.003	0.002	0.005	0.007	0.008	642
0	0.003	0.002	0.005	0.007	0.008	835
0	0.003	0.002	0.005	0.007	0.008	806
0	0.003	0.002	0.005	0.007	0.008	397
0	0.003	0.002	0.005	0.007	0.008	389
0	0.003	0.002	0.005	0.007	0.008	829
0	0.003	0.002	0.005	0.007	0.008	720
0	0.003	0.002	0.005	0.007	0.008	658
0	0.003	0.002	0.005	0.007	0.008	750
0	0.003	0.002	0.005	0.007	0.008	746
0	0.003	0.002	0.005	0.007	0.008	676
0	0.003	0.002	0.005	0.007	0.008	832
0	0.003	0.002	0.005	0.007	0.008	740
0	0.003	0.002	0.005	0.007	0.008	131
0	0.003	0.002	0.005	0.007	0.008	731
0	0.003	0.002	0.005	0.007	0.008	600
0	0.003	0.002	0.005	0.007	0.008	744
0	0.003	0.002	0.005	0.007	0.008	649
0	0.003	0.002	0.005	0.007	0.008	683
0	0.003	0.002	0.005	0.007	0.008	690
0	0.003	0.002	0.005	0.007	0.008	670
0	0.003	0.002	0.005	0.007	0.008	726
0	0.003	0.002	0.005	0.007	0.008	814
0	0.003	0.002	0.005	0.007	0.008	663
0	0.003	0.002	0.005	0.007	0.008	376
0	0.003	0.002	0.005	0.007	0.008	561
0	0.003	0.002	0.005	0.007	0.008	805
0	0.003	0.002	0.005	0.007	0.008	804
0	0.003	0.002	0.005	0.007	0.008	827
0	0.003	0.002	0.005	0.007	0.008	278
0	0.003	0.002	0.005	0.007	0.008	348
0	0.003	0.002	0.005	0.007	0.008	331
0	0.003	0.002	0.005	0.007	0.008	837
0	0.003	0.002	0.005	0.007	0.008	316
0	0.003	0.002	0.005	0.007	0.008	374
0	0.003	0.002	0.005	0.007	0.008	846
0	0.003	0.002	0.005	0.007	0.007	799
0	0.002	0.002	0.005	0.007	0.007	123
0	0.001	0.001	0.002	0.007	0.008	286
0	0.001	0.001	0.003	0.007	0.007	402
0	0.001	0.001	0.003	0.008	0.008	724

win8_1804_event_k8

0.759	0.368	0.523	1.665	1.665	1.665	249
0.759	0.368	0.523	1.665	1.665	1.665	247
0.759	0.368	0.523	1.665	1.665	1.665	258
0.759	0.368	0.523	1.665	1.665	1.665	262
0.759	0.368	0.523	1.665	1.665	1.665	267
0.759	0.368	0.523	1.665	1.665	1.665	271
0.759	0.368	0.523	1.665	1.665	1.665	278
0.759	0.368	0.523	1.665	1.665	1.665	305
0.759	0.368	0.523	1.665	1.665	1.665	309
0.759	0.368	0.523	1.665	1.665	1.665	318
0.759	0.368	0.523	1.665	1.665	1.665	322
0.759	0.368	0.523	1.665	1.665	1.665	269
0.759	0.368	0.523	1.665	1.665	1.665	251
0.759	0.368	0.523	1.665	1.665	1.665	273
0.759	0.368	0.523	1.665	1.665	1.665	290
0.759	0.368	0.523	1.665	1.665	1.665	297
0.759	0.368	0.523	1.665	1.665	1.665	301
0.759	0.368	0.523	1.665	1.665	1.665	303
0.759	0.368	0.523	1.665	1.665	1.665	307
0.759	0.368	0.523	1.665	1.665	1.665	311
0.758	0.368	0.522	1.663	1.663	1.663	255
0.758	0.368	0.522	1.663	1.663	1.663	237
0.758	0.368	0.522	1.663	1.663	1.663	292
0.758	0.368	0.522	1.663	1.663	1.663	299
0.758	0.368	0.522	1.663	1.663	1.663	260
0.758	0.368	0.522	1.663	1.663	1.663	241
0.758	0.368	0.522	1.663	1.663	1.663	229
0.758	0.368	0.522	1.663	1.663	1.663	224
0.758	0.368	0.522	1.663	1.663	1.663	231
0.651	0.315	0.448	1.427	1.427	1.427	243
0.794	0.371	0.627	1.997	1.498	1.236	43
0.57	0.278	0.392	1.249	1.249	1.308	125
0.506	0.245	0.348	1.11	1.11	1.11	253
0.379	0.184	0.261	0.832	0.832	0.832	137
1.334	0.766	0.47	0.713	1.069	1.12	120
1.248	0.35	0.466	1.536	0.768	0.732	121
0.758	0.363	0.626	0	2.492	2.373	329
0.758	0.363	0.626	0	2.492	2.373	189
0.758	0.363	0.626	0	2.492	2.373	291
0.76	0.364	0.628	0	2.5	2.381	282
0.641	0.359	0.626	0	2.492	2.373	142
0.38	0.182	0.314	0	1.25	1.19	236
0.379	0.182	0.313	0	1.248	1.189	315
0.409	0.18	0.313	0	1.248	1.426	44
0.507	0.243	0.418	0	1.667	1.587	246
0.506	0.242	0.418	0	1.663	1.584	128
1.516	0.689	0.605	0	1.496	1.805	226
0.304	0.146	0.251	0	0.999	0.951	250
0.304	0.146	0.251	0	0.999	0.951	248
0.304	0.146	0.251	0	0.999	0.951	259
0.304	0.146	0.251	0	0.999	0.951	263

	0.392364512456647	0.190171384115316	0.270048449688947	0.860448492229488	0.860448492229488	0.860448492229488	144
	0.282061855670103	0.138164209734563	0.194131641554322	0.618556701030628	0.618556701030628	0.618556701030628	223
	0.502372881355932	0.244353723751141	0.345762711864407	1.10169491525424	1.10169491525424	1.15415657788539	133
	0.29582485240451	0.0464421176937232	0.132342697933596	1.33162157363526	0.332905393408815	0.317052755627443	231
	0.038912	0.029952398591366	0.008704	0.832	0.208	0.217904761904762	2
	0.960596422468424	0.331876237642891	0.585050037490013	1.99741872942284	0.49935468010571	0.570991062977954	354
	1.19428571428571	2.99997681260235	0.369894126884127	1.03174803174803	0.77380592369524	0.687630887630688	146
	2.14465326801647	3.11852362107121	0.484474826308045	0	0.454020526384413	0.475640551450338	74
	8.08072667217176	0.225394380815809	0.13813377374071	0	0.241535920726672	0.536746490503716	460
	1.14	0.552539642263045	0.784615384615384	2.5	2.5	2.5	297
	1.14	0.552539642263045	0.784615384615384	2.5	2.5	2.5	283
	1.14	0.552539642263045	0.784615384615384	2.5	2.5	2.5	301
	1.14	0.552539642263045	0.784615384615384	2.5	2.5	2.5	374
	1.14	0.552539642263045	0.784615384615384	2.5	2.5	2.5	406
	1.13819056883928	0.551658646466099	0.783370027136347	2.49603194920895	2.49603194920895	2.49603194920895	383
	1.13819056883928	0.551658646466099	0.783370027136347	2.49603194920895	2.49603194920895	2.49603194920895	202
	1.13819056883928	0.551658646466099	0.783370027136347	2.49603194920895	2.49603194920895	2.49603194920895	412
	1.13816143153368	0.551644524185515	0.783349973120344	2.49596805160894	2.49596805160894	2.49596805160894	285
	1.13816143153368	0.551644524185515	0.783349973120344	2.49596805160894	2.49596805160894	2.49596805160894	163
	1.11672066912817	0.541252607144872	0.768593173084168	2.4489488358074	2.4489488358074	2.4489488358074	313
	1.10546950629204	0.579458159843465	0.822846079380445	2.62181348822201	2.62181348822201	2.62181348822201	337
	1.310035947905	0.63494871459095	0.90164417467146	2.87288585066887	2.87288585066887	2.87288585066887	317
	1.61672727272727	0.783596001754864	1.11272727272727	3.54545454545455	3.54545454545455	3.54545454545455	194
	1.52	0.73671418968406	1.04615384615385	3.33333333333333	3.33333333333333	3.33333333333333	233
	1.52	0.728676804687422	1.25538461538462	0	5	4.76190476190476	363
	1.52	0.728676804687422	1.25538461538462	0	5	4.76190476190476	300
	1.52	0.728676804687422	1.25538461538462	0	5	4.76190476190476	365
	1.17965454111279	0.560672918158749	0.97428960514208	0	3.88044256944997	3.6950958995235	310
	1.87076923076923	1.58315176932107	2.72	0	2.5	4.28571428571429	356
	1.87076923076923	1.58315176932107	2.72	0	2.5	4.28571428571429	188
	2.04413793103448	0.99426687595292	1.40689655172414	4.48275862068966	4.48275862068966	4.48275862068965	452
	1.98695030389703	0.963035644130263	1.36753684460686	4.35734715766893	4.35734715766893	4.35734715766893	148
	2.15553185300107	1.04844878076098	1.4835644332396	4.72704353728304	4.72704353728304	4.72704353728304	142
	2.71494870542257	1.31588239180867	1.8685881778212	5.95383488031265	5.95383488031265	5.95383488031265	309
	4.61820937281418	1.2961424913566	1.72417813010026	5.68314292375845	2.84157146187923	2.70825853512307	129
	4.56	2.21014259905218	3.13846153846154	10	10	10	362
	4.56	2.21014259905218	3.13846153846154	10	10	10	299
	4.56	2.21014259905218	3.13846153846154	10	10	10	364
	filesystem	registry	module.load	process.create	thread.create	ob	

Figure D.6: Part from 1604 avond



	0.58347573044221	1.32125590570675	0.18612475966738	0	0.325451292458878	0.227299315368104	330
	0.607599323664395	0.879153202850535	0.136579884266795	0	0.127994412879782	0.134089384921677	72
	2.17678216083343	0.287348230354209	0.343263490577081	0	0.178325822310664	0.237767763080885	100
	1.28432432432432	3.22614860644008	0.394651493598062	1.10953058321479	0.832147937411095	0.736687055476529	139
	1.52	0.728676804687422	1.25538461538462	0	5	4.76190476190476	223
	1.52	0.728676804687422	1.25538461538462	0	5	4.76190476190476	192
	1.52	0.728676804687422	1.25538461538462	0	5	4.76190476190476	240
	1.52	0.728676804687422	1.25538461538462	0	5	4.76190476190476	276
	1.52	0.728676804687422	1.25538461538462	0	5	4.76190476190476	284
	1.28615384615385	0.721039419690783	1.25538461538462	0	5	4.76190476190476	136
	1.87076923076923	1.58315176932107	2.72	0	2.5	4.28571428571429	204
	1.14	0.552535642263045	0.794615384615384	2.5	2.5	2.5	235
	1.13819056883928	0.551658646466099	0.783370027136347	2.49603194920895	2.49603194920895	2.49603194920895	245
	0.910822936512814	0.441458014222825	0.626882183025014	1.99741872042284	1.99741872042284	1.99741872042284	314
	1.51678493449782	0.735155910489177	1.04394104803493	3.3262827510917	3.3262827510917	3.3262827510917	127
	2.27277374501585	1.10156886045695	1.56425723341172	4.98415294959616	4.98415294959616	4.98415294959616	328
	2.27277374501585	1.10156886045695	1.56425723341172	4.98415294959616	4.98415294959616	4.98415294959616	141
	2.27277374501585	1.10938140556057	1.56425723341172	4.98415294959616	4.98415294959616	4.98415294959616	188
	4.56	2.21014256905218	3.13846153846154	10	10	10	239
	4.56	2.21014256905218	3.13846153846154	10	10	10	191
	4.56	2.21014256905218	3.13846153846154	10	10	10	275
	4.56	2.21014256905218	3.13846153846154	10	10	10	281
	4.56	2.21014256905218	3.13846153846154	10	10	10	283
	4.56	2.21797995404882	3.13846153846154	10	10	10	135
	4.56	2.22581733904546	3.13846153846154	10	10	10	222
filesystem	registry	module.load	process.create	thread.create	ob		

Figure D.9: Part from 1804

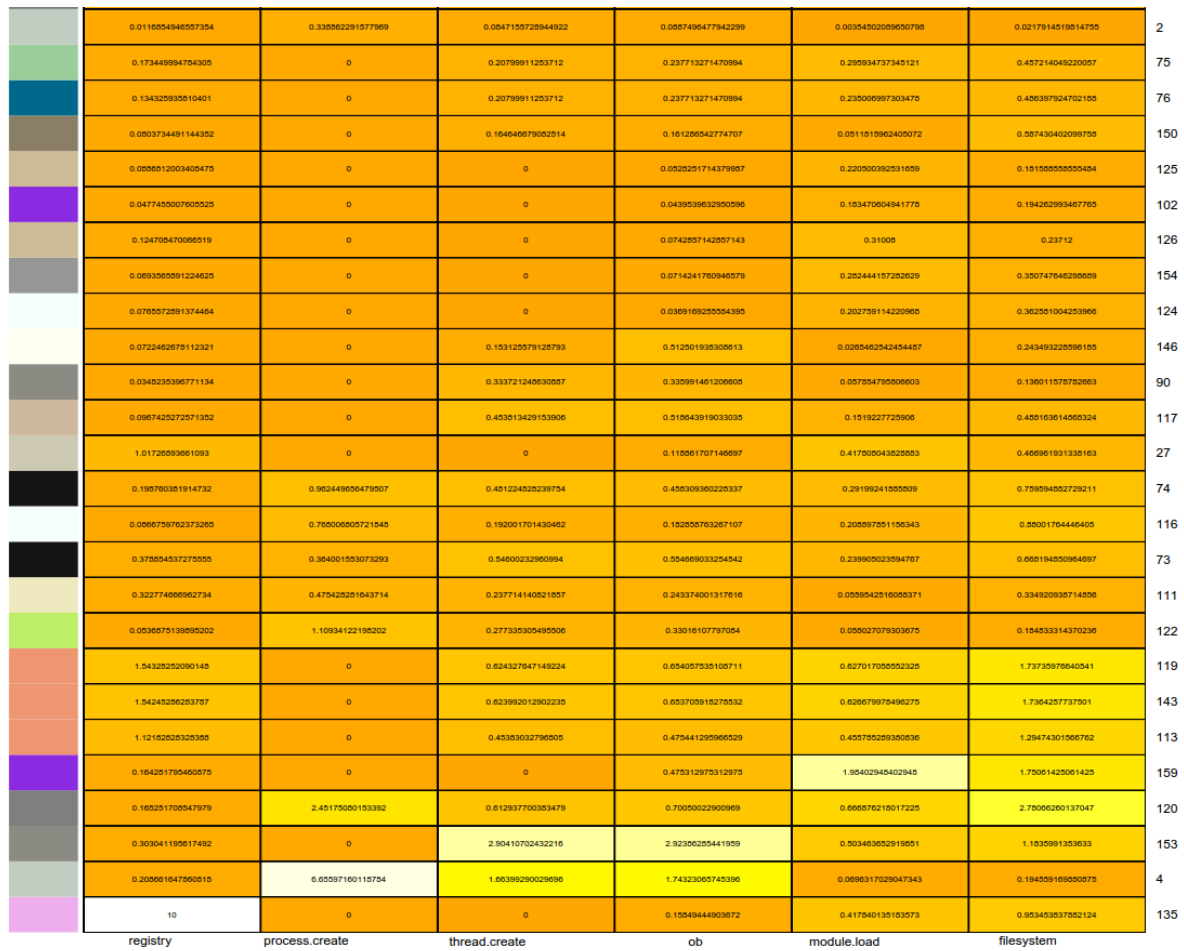


Figure D.10: Part from bank malware

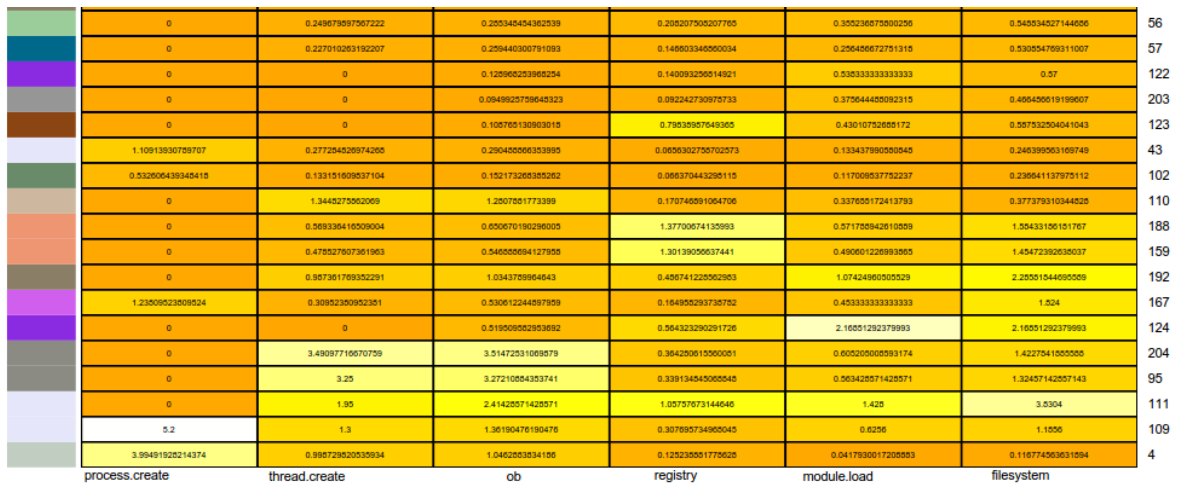


Figure D.11: Part from rat 1 malware

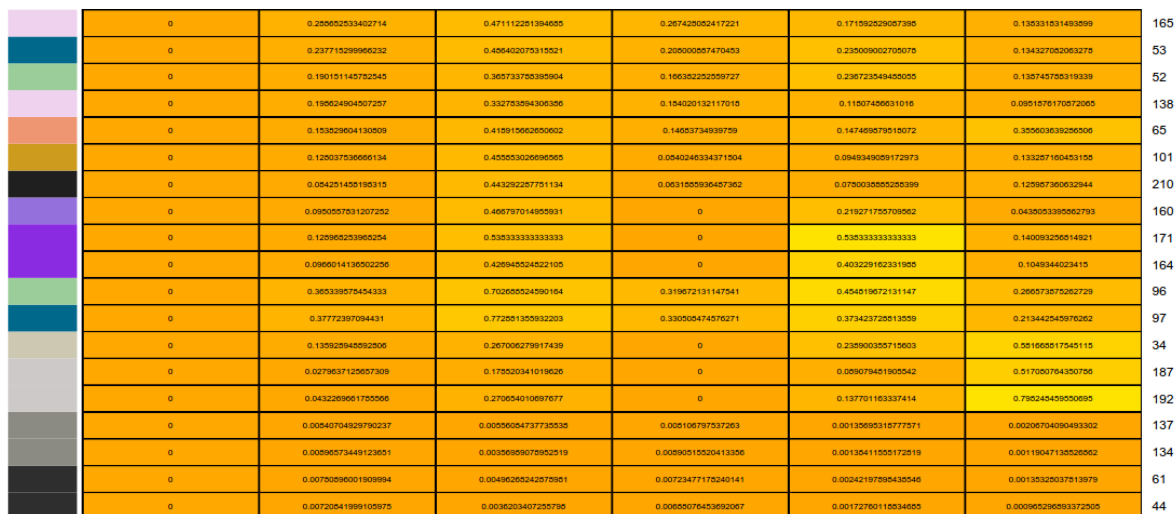


Figure D.12: Part from rat 2 malware upper part

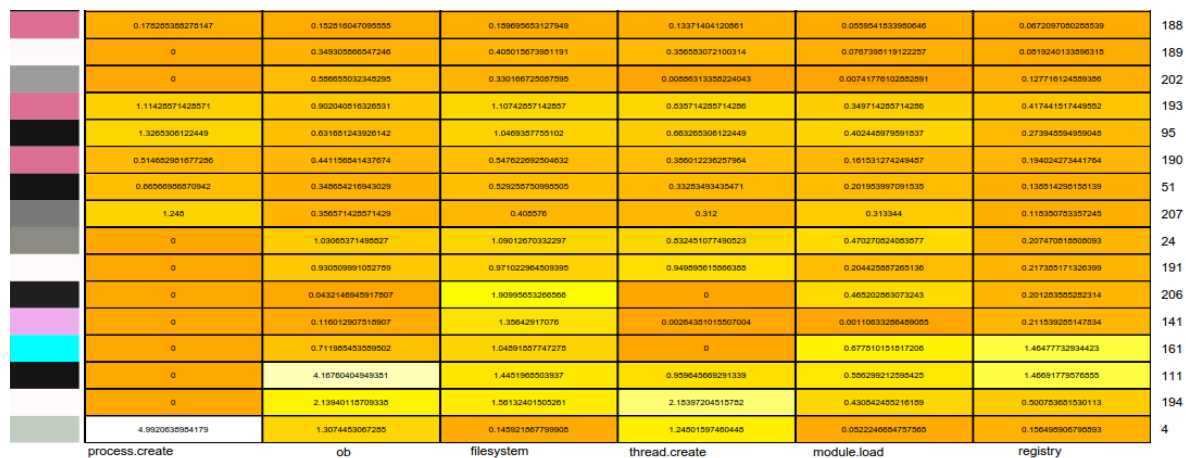


Figure D.13: Part from rat 2 malware bottom part

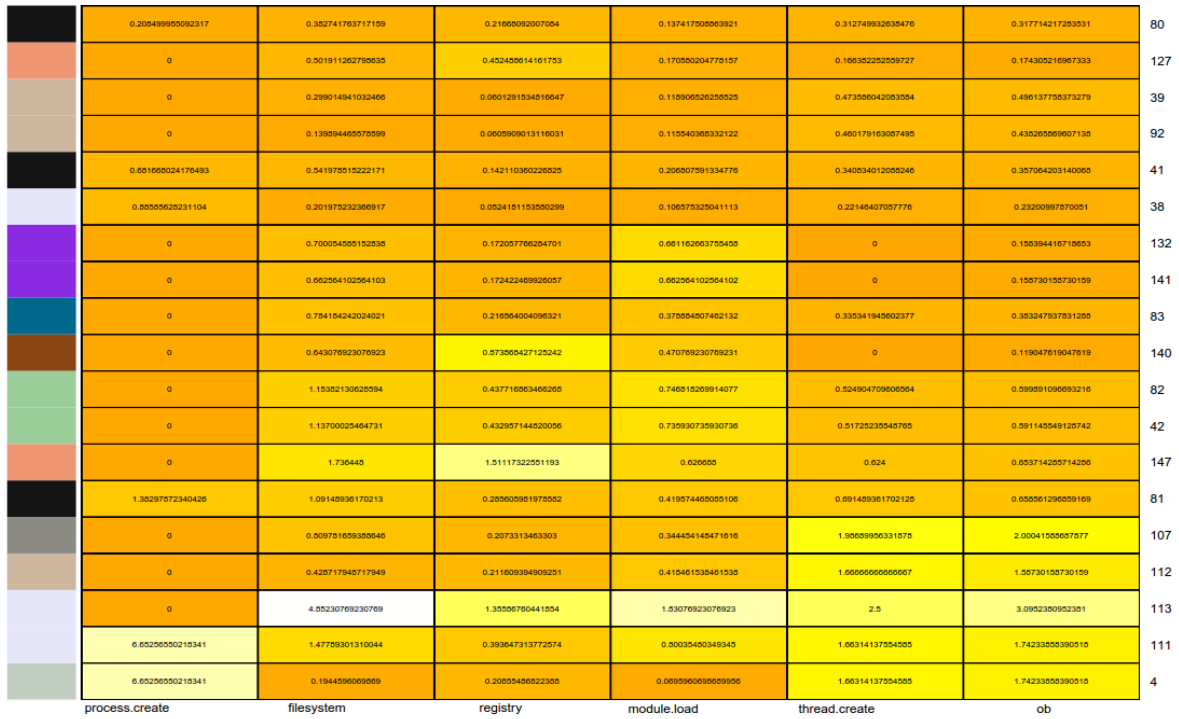


Figure D.14: Part from zeus 1 malware bottom part

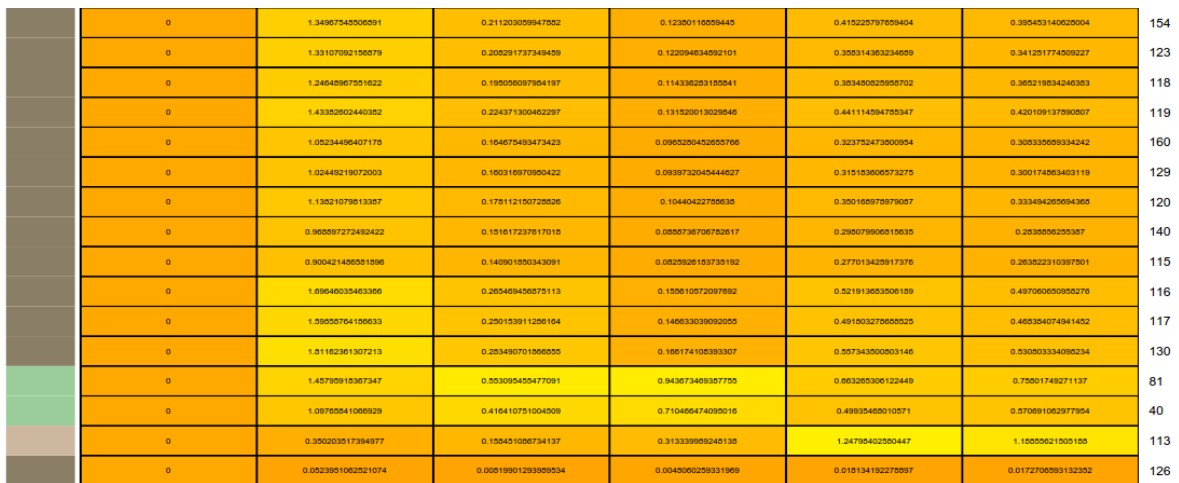


Figure D.15: Part from zeus 1 malware upper part

	0	0.123764102424917	0.0316077941999013	0.0526451612903226	0.303670745272525	0.30073653205531	96
	0	0.230926325095506	0.0553685309931455	0.125502991747062	0.222159435252105	0.275054538920705	22
	0	0.257939148720952	0.152120564305801	0.10660595735619	0.13903730097464	0.132418477118754	167
	0	0.254510050251256	0.157253320213956	0.091551256251407	0.117527939658432	0.111985513885564	92
	0	0.291004406199521	0.157020353697251	0.111304239352059	0.142491640490425	0.135706324276595	127
	0	0.250961300537166	0.139471190624421	0.0504436739560097	0.10295396475054	0.0950799654576574	176
	0	0.338310654923939	0.0529419670269356	0.0310325262610085	0.130104053266613	0.12390650730108	157
	0	0.30539084730793	0.0452583556981171	0.025257662265472	0.118595075299966	0.112947690781873	177
	0	0.253756260434057	0.0700785209382273	0.122604340967613	0.108514190317195	0.124016217505366	41
	0	0.443054918525536	0.069407503211797	0.0487180534951211	0.102420413211215	0.0942915088551053	137
	0	0.439594	0.178571070301291	0.120785	0.234	0.222557142557143	139
	0	0.39516150455457	0.20468732417433	0.115047260237169	0.151124022555935	0.143927665451543	121
	0	0.66050600605061	0.145263214987209	0.0537543754375438	0.107260726072607	0.102153072450102	164
	0.30250090500731	0.555298731357407	0.313659047008776	0.189371415465801	0.453751477600966	0.46095305200733	79
	1.17469879510072	0.92710843373484	0.242593032855422	0.356355542169675	0.587349397590361	0.559380375657487	80
	1.17469879510072	0.287831325301203	0.0690085798525403	0.141325301204819	0.293674698795181	0.3076959205281618	36
	0.605550499512691	0.0442515373532777	0.0237456259697311	0.00720369507182508	0.172147124575173	0.150344607015229	2
	0	2.91366554855355	0.504651012919783	1.40775954605109	1.24597454375163	1.423970907175	82
	0	1.935	0.504335724533716	1.935	0	0.464285714285714	163
	0	4.8366514905693	1.35150121587494	1.8248734566171	2.49194908756326	3.08527028585754	114
	6.66666666666667	1.52	0.394451711497493	0.502051252051252	1.66666666666667	1.74603174603175	112
	6.65250550210341	0.1944596069589	0.20855456522385	0.0895960650559956	1.66314137554555	1.74233558390515	4
	process.create	filesystem	registry	module.load	thread.create	ob	

Figure D.16: Part from zeus 2 malware bottom part

D.2 Benign process analysis

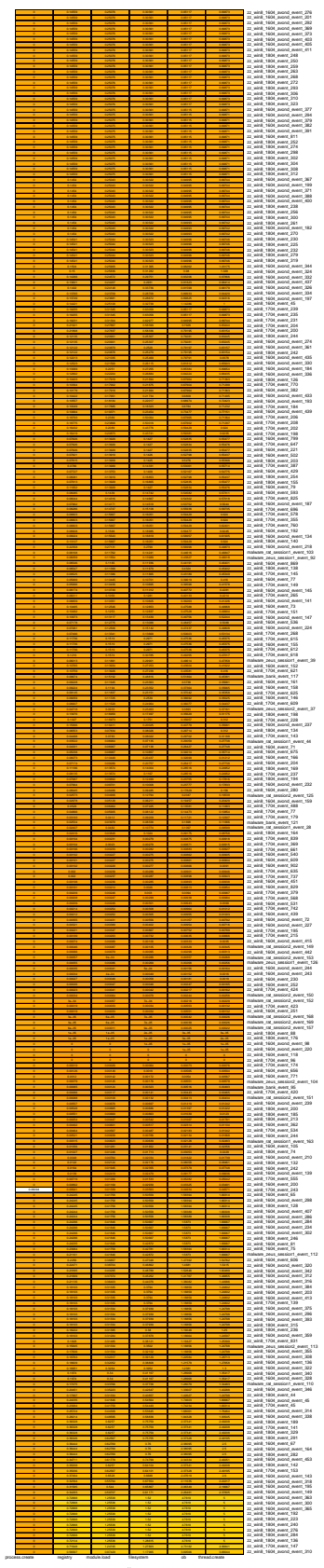


Figure D.17: Heatmap showing the process

E
INPUT IS HERE

F

Evaluation Algorithm 1

F.1 Banking malware

Table F.1: Outcome using different threshold values for bank malware using algorithm 1

thresh- old type	total nr pro- cesses	total nr mali- cious pro- cesses	total nr above thresh- old	number malicious above threshold	FPR	TPR	ACC	process ids	all processes
mean	161	12	19	8	0.074	0.667	0.907	111; 116; 117; 120; 122; 124; 125; 126	4; 27; 73; 74; 111; 113; 116; 117; 119; 120; 122; 124; 125; 126; 135; 143; 146; 153; 159
q0.75	161	12	35	11	0.161	0.917	0.845	111; 112; 116; 117; 118; 120; 121; 122; 124; 125; 126	1; 2; 4; 13; 27; 58; 65; 73; 74; 75; 76; 90; 111; 112; 113; 116; 117; 118; 119; 120; 121; 122; 124; 125; 126; 131; 134; 135; 143; 146; 150; 153; 154; 157; 159
q0.8	161	12	27	8	0.128	0.667	0.857	111; 116; 117; 120; 122; 124; 125; 126	1; 2; 4; 27; 65; 73; 74; 76; 90; 111; 113; 116; 117; 119; 120; 122; 124; 125; 126; 135; 143; 146; 150; 153; 154; 157; 159
q0.85	161	12	21	8	0.087	0.667	0.894	111; 116; 117; 120; 122; 124; 125; 126	4; 27; 73; 74; 111; 113; 116; 117; 119; 120; 122; 124; 125; 126; 135; 143; 146; 150; 153; 154; 159
q0.9	161	12	13	3	0.067	0.250	0.882	116; 120; 122	4; 27; 73; 74; 113; 116; 119; 120; 122; 135; 143; 153; 159
q0.95	161	12	3	1	0.013	0.083	0.919	120	4; 120; 135

F.2 Rat malware session 1

Table F.2: Outcome using different threshold values for rat1 malware using algorithm 1

thresh-old type	total nr pro- cesses	total nr mali- cious pro- cesses	total nr above thresh- old	number malicious above threshold	FPR	TPR	ACC	process ids	all processes
mean	224	20	21	1	0.098	0.050	0.826	167	1; 4; 43; 56; 57; 95; 109; 111; 120; 122; 124; 159; 167; 187; 188; 191; 192; 204; 203; 208; 222
q0.75	224	20	40	10	0.147	0.500	0.821	167; 171; 173; 175; 177; 179; 181; 183; 185; 186	1; 4; 26; 27; 43; 56; 57; 72; 86; 95; 97; 102; 107; 109; 111; 115; 120; 122; 123; 124; 159; 167; 171; 173; 175; 177; 179; 181; 183; 185; 186; 187; 188; 191; 192; 201; 204; 203; 208; 222
q0.8	224	20	27	2	0.123	0.100	0.808	167; 186	1; 4; 26; 43; 56; 57; 86; 95; 109; 111; 115; 120; 122; 123; 124; 159; 167; 186; 187; 188; 191; 192; 201; 204; 203; 208; 222
q0.85	224	20	23	1	0.108	0.050	0.817	167	1; 4; 43; 56; 57; 95; 109; 111; 120; 122; 123; 124; 159; 167; 187; 188; 191; 192; 201; 204; 203; 208; 222
q0.9	224	20	16	1	0.074	0.050	0.848	167	4; 43; 56; 57; 95; 109; 111; 124; 159; 167; 187; 188; 192; 204; 203; 208
q0.95	224	20	9	1	0.039	0.050	0.879	167	4; 95; 109; 111; 159; 167; 188; 192; 204

F.3 Rat malware session 2

Table F.3: Outcome using different threshold values for rat2 malware using algorithm 1

thresh-old type	total nr pro- cesses	total nr mali- cious pro- cesses	total nr above thresh- old	number malicious above threshold	FPR	TPR	ACC	process ids	all processes
mean	213	20	30	0	0.155	0	0.765		2; 4; 24; 34; 51; 52; 53; 65; 95; 96; 97; 101; 111; 124; 125; 141; 160; 161; 164; 171; 187; 189; 190; 191; 192; 193; 194; 202; 206; 207
q0.75	213	20	47	8	0.202	0.400	0.761	60; 72; 74; 76; 80; 82; 84; 85	1; 2; 4; 20; 24; 34; 51; 52; 53; 60; 65; 69; 72; 74; 76; 80; 82; 84; 85; 91; 94; 95; 96; 97; 101; 111; 114; 116; 124; 125; 141; 160; 161; 164; 165; 171; 187; 188; 189; 190; 191; 192; 193; 194; 202; 206; 207
q0.8	213	20	38	0	0.197	0	0.728		1; 2; 4; 24; 34; 51; 52; 53; 65; 69; 91; 94; 95; 96; 97; 101; 111; 114; 116; 124; 125; 141; 160; 161; 164; 165; 171; 187; 188; 189; 190; 191; 192; 193; 194; 202; 206; 207
q0.85	213	20	25	0	0.130	0	0.789		2; 4; 24; 34; 51; 52; 53; 95; 96; 97; 111; 124; 141; 160; 161; 164; 187; 189; 190; 192; 193; 194; 202; 206; 207
q0.9	213	20	11	0	0.057	0	0.854		4; 24; 34; 51; 111; 161; 187; 192; 194; 202; 206
q0.95	213	20	4	0	0.021	0	0.887		4; 24; 111; 161

F.4 Zeus malware session 1

Table F.4: Outcome using different threshold values for zeus1 malware using algorithm 1

thresh- old type	total nr pro- cesses	total nr malicious processes	total nr above thresh- old	number malicious above threshold	FPR	TPR	ACC	pro- cess ids	all processes
mean	156	5	17	0	0.113	0	0.859	4; 38; 41; 43; 80; 81; 82; 83; 107; 111; 113; 121; 127; 134; 140; 147; 150	
q0.75	156	5	35	0	0.232	0	0.744	1; 2; 4; 12; 38; 39; 41; 42; 43; 73; 75; 80; 81; 82; 83; 84; 91; 97; 107; 111; 112; 113; 117; 118; 121; 127; 130; 132; 134; 135; 140; 141; 147; 150; 152	
q0.8	156	5	26	0	0.172	0	0.801	2; 4; 38; 39; 41; 42; 43; 80; 81; 82; 83; 84; 107; 111; 112; 113; 117; 121; 127; 130; 132; 134; 140; 147; 150; 152	
q0.85	156	5	19	0	0.126	0	0.846	4; 38; 39; 41; 43; 80; 81; 82; 83; 84; 107; 111; 113; 121; 127; 134; 140; 147; 150	
q0.9	156	5	8	0	0.053	0	0.917	4; 38; 41; 80; 107; 111; 113; 147	
q0.95	156	5	5	0	0.033	0	0.936	4; 107; 111; 113; 147	

F.5 Zeus malware session 2

Table F.5: Outcome using different threshold values for zeus2 malware using algorithm 1

thresh- old type	total nr pro- cesses	total nr mali- cious pro- cesses	total nr above thresh- old	number malicious above threshold	FPR	TPR	ACC	pro- cess ids	all processes
mean	191	2	20	0	0.106	0	0.885	4; 59; 79; 80; 81; 112; 114; 115; 116; 117; 118; 119; 120; 123; 129; 130; 140; 154; 160; 164	
q0.75	191	2	38	0	0.201	0	0.791	1; 4; 26; 36; 37; 40; 59; 79; 80; 81; 82; 85; 92; 112; 114; 115; 116; 117; 118; 119; 120; 121; 123; 127; 129; 130; 137; 139; 140; 148; 151; 154; 160; 163; 164; 167; 176; 187	
q0.8	191	2	28	0	0.148	0	0.843	1; 4; 36; 37; 40; 59; 79; 80; 81; 82; 112; 114; 115; 116; 117; 118; 119; 120; 121; 123; 127; 129; 130; 139; 140; 154; 160; 164	
q0.85	191	2	21	0	0.111	0	0.880	4; 79; 80; 81; 82; 112; 114; 115; 116; 117; 118; 119; 120; 121; 123; 129; 130; 140; 154; 160; 164	
q0.9	191	2	13	0	0.069	0	0.921	4; 112; 114; 116; 117; 118; 119; 120; 123; 129; 130; 154; 160	
q0.95	191	2	6	0	0.032	0	0.958	4; 112; 114; 116; 117; 130	

G

Evaluation Algorithm 2

G.1 Banking malware

Table G.1: Outcome using different threshold values for bank malware using algorithm 2

thresh- old type	total nr pro- cesses	total nr mali- cious pro- cesses	total nr above thresh- old	number malicious above threshold	FPR	TPR	ACC	process ids	all processes
mean	161	12	18	5	0.087	0.417	0.876	111; 116; 120; 122; 126	4; 27; 73; 74; 111; 113; 116; 119; 120; 122; 126; 131; 135; 143; 146; 153; 154; 159
q0.75	161	12	32	9	0.154	0.750	0.839	111; 116; 117; 118; 120; 122; 124; 125; 126	1; 2; 4; 27; 65; 73; 74; 76; 84; 90; 111; 113; 116; 117; 118; 119; 120; 122; 124; 125; 126; 131; 134; 135; 143; 146; 147; 150; 153; 154; 157; 159
q0.8	161	12	24	8	0.107	0.667	0.876	111; 116; 117; 120; 122; 124; 125; 126	1; 4; 27; 65; 73; 74; 111; 113; 116; 117; 119; 120; 122; 124; 125; 126; 131; 135; 143; 146; 150; 153; 154; 159
q0.85	161	12	17	4	0.087	0.333	0.870	111; 116; 120; 122	4; 27; 73; 74; 111; 113; 116; 119; 120; 122; 131; 135; 143; 146; 153; 154; 159
q0.9	161	12	10	3	0.047	0.250	0.901	111; 120; 122	4; 111; 113; 119; 120; 122; 135; 143; 153; 159
q0.95	161	12	3	0	0.020	0	0.907		4; 135; 159

G.2 Rat malware session 1

Table G.2: Outcome using different threshold values for rat1 malware using algorithm 2

thresh- old type	to- tal nr pro- cesses	total nr ma- licious pro- cesses	total nr above thresh- old	number mali- cious above thresh- old	FPR	TPR	ACC	process ids	all processes
mean	224	20	17	1	0.078	0.050	0.844	167	4; 43; 45; 56; 95; 109; 110; 111; 159; 162; 163; 167; 188; 192; 204; 203; 208
q0.75	224	20	46	10	0.176	0.500	0.795	167; 171; 173; 175; 177; 179; 181; 183; 185; 186	1; 4; 26; 43; 45; 56; 57; 72; 75; 86; 93; 95; 96; 103; 109; 110; 111; 115; 120; 122; 123; 124; 159; 162; 163; 167; 171; 173; 175; 177; 179; 181; 183; 185; 186; 187; 188; 191; 192; 194; 201; 204; 203; 208; 213; 222
q0.8	224	20	36	1	0.172	0.050	0.759	167	1; 4; 26; 43; 45; 56; 57; 72; 75; 86; 93; 95; 103; 109; 110; 111; 115; 120; 122; 123; 124; 159; 162; 163; 167; 187; 188; 191; 192; 194; 201; 204; 203; 208; 213; 222
q0.85	224	20	25	1	0.118	0.050	0.808	167	1; 4; 43; 45; 56; 57; 95; 109; 110; 111; 120; 124; 159; 162; 163; 167; 187; 188; 191; 192; 194; 204; 203; 208; 222
q0.9	224	20	17	1	0.078	0.050	0.844	167	4; 43; 45; 56; 95; 109; 110; 111; 159; 162; 163; 167; 188; 192; 204; 203; 208
q0.95	224	20	4	0	0.020	0	0.893		4; 109; 110; 111

G.3 Rat malware session 2

Table G.3: Outcome using different threshold values for rat2 malware using algorithm 2

thresh- old type	total nr pro- cesses	total nr mali- cious pro- cesses	total nr above thresh- old	number malicious above threshold	FPR	TPR	ACC	pro- cess ids	all processes
mean	213	20	20	0	0.104	0	0.812		4; 24; 34; 51; 53; 65; 95; 111; 124; 125; 161; 164; 187; 189; 192; 194; 199; 202; 206; 207
q0.75	213	20	41	1	0.207	0.050	0.723	60	1; 2; 4; 24; 34; 51; 52; 53; 60; 65; 69; 91; 94; 95; 96; 97; 101; 105; 111; 116; 124; 125; 141; 160; 161; 164; 165; 171; 187; 188; 189; 190; 191; 192; 193; 194; 195; 199; 202; 206; 207
q0.8	213	20	33	0	0.171	0	0.751		1; 2; 4; 24; 34; 51; 52; 53; 65; 69; 95; 96; 97; 101; 111; 124; 125; 141; 160; 161; 164; 165; 187; 188; 189; 190; 192; 193; 194; 199; 202; 206; 207
q0.85	213	20	22	0	0.114	0	0.803		4; 24; 34; 51; 53; 65; 95; 96; 111; 124; 125; 161; 164; 187; 189; 190; 192; 194; 199; 202; 206; 207
q0.9	213	20	9	0	0.047	0	0.864		4; 24; 34; 111; 161; 192; 194; 202; 206
q0.95	213	20	3	0	0.016	0	0.892		4; 111; 206

G.4 Zeus malware session 1

Table G.4: Outcome using different threshold values for zeus1 malware using algorithm 2

thresh- old type	total nr pro- cesses	total nr malicious processes	total nr above thresh- old	number malicious above threshold	FPR	TPR	ACC	pro- cess ids	all processes
mean	156	5	10	0	0.066	0	0.904	4; 38; 40; 80; 107; 111; 112; 113; 134; 147	
q0.75	156	5	34	0	0.225	0	0.750	1; 2; 4; 38; 39; 40; 41; 42; 43; 75; 80; 81; 82; 83; 84; 92; 107; 111; 112; 113; 117; 118; 121; 127; 130; 132; 134; 135; 140; 142; 147; 150; 152; 153	
q0.8	156	5	21	0	0.139	0	0.833	4; 38; 39; 40; 41; 43; 75; 80; 81; 82; 107; 111; 112; 113; 118; 121; 127; 134; 135; 147; 150	
q0.85	156	5	14	0	0.093	0	0.878	4; 38; 40; 41; 80; 107; 111; 112; 113; 118; 127; 134; 147; 150	
q0.9	156	5	8	0	0.053	0	0.917	4; 38; 40; 107; 111; 112; 113; 147	
q0.95	156	5	4	0	0.026	0	0.942	4; 111; 112; 113	

G.5 Zeus malware session 2

Table G.5: Outcome using different threshold values for zeus2 malware using algorithm 2

thresh- old type	total nr pro- cesses	total nr mali- cious processes	total nr above thresh- old	number malicious above threshold	FPR	TPR	ACC	pro- cess ids	all processes
mean	191	2	20	0	0.106	0	0.885	4; 36; 38; 53; 54; 79; 80; 112; 113; 114; 116; 117; 118; 119; 120; 123; 129; 130; 154; 160	
q0.75	191	2	37	0	0.196	0	0.796	1; 4; 36; 37; 38; 40; 53; 54; 59; 79; 80; 81; 82; 92; 109; 112; 113; 114; 115; 116; 117; 118; 119; 120; 121; 123; 127; 129; 130; 139; 140; 154; 160; 164; 167; 176; 188	
q0.8	191	2	30	0	0.159	0	0.832	4; 36; 37; 38; 53; 54; 79; 80; 81; 82; 112; 113; 114; 115; 116; 117; 118; 119; 120; 121; 123; 127; 129; 130; 139; 140; 154; 160; 164; 167	
q0.85	191	2	22	0	0.116	0	0.874	4; 36; 38; 53; 54; 79; 80; 81; 112; 113; 114; 116; 117; 118; 119; 120; 123; 129; 130; 140; 154; 160	
q0.9	191	2	12	0	0.063	0	0.927	4; 36; 38; 112; 113; 114; 116; 117; 119; 123; 130; 154	
q0.95	191	2	5	0	0.026	0	0.963	4; 36; 112; 113; 114	

H

Evaluation Algorithm 3

H.1 Banking malware

Table H.1: Outcome using different threshold values for bank malware using algorithm d3

thresh- old type	total nr pro- cesses	total nr mali- cious pro- cesses	total nr above thresh- old	number malicious above threshold	FPR	TPR	ACC	process ids	all processes
mean	161	12	18	7	0.074	0.583	0.901	116; 117; 120; 122; 124; 125; 126	4; 27; 73; 74; 113; 116; 117; 119; 120; 122; 124; 125; 126; 135; 143; 146; 153; 159
q0.75	161	12	35	11	0.161	0.917	0.845	111; 112; 116; 117; 118; 120; 121; 122; 124; 125; 126	1; 2; 4; 13; 27; 58; 65; 73; 74; 75; 76; 90; 111; 112; 113; 116; 117; 118; 119; 120; 121; 122; 124; 125; 126; 131; 134; 135; 143; 146; 150; 153; 154; 157; 159
q0.8	161	12	27	8	0.128	0.667	0.857	111; 116; 117; 120; 122; 124; 125; 126	1; 2; 4; 27; 65; 73; 74; 76; 90; 111; 113; 116; 117; 119; 120; 122; 124; 125; 126; 135; 143; 146; 150; 153; 154; 157; 159
q0.85	161	12	21	8	0.087	0.667	0.894	111; 116; 117; 120; 122; 124; 125; 126	4; 27; 73; 74; 111; 113; 116; 117; 119; 120; 122; 124; 125; 126; 135; 143; 146; 150; 153; 154; 159
q0.9	161	12	13	3	0.067	0.250	0.882	116; 120; 122	4; 27; 73; 74; 113; 116; 119; 120; 122; 135; 143; 153; 159
q0.95	161	12	3	1	0.013	0.083	0.919	120	4; 120; 135

H.2 Rat malware session 1

Table H.2: Outcome using different threshold values for rat1 malware using algorithm 3

thresh-old type	total nr pro- cesses	total nr mali- cious pro- cesses	total nr above thresh- old	number malicious above threshold	FPR	TPR	ACC	process ids	all processes
mean	224	20	18	1	0.083	0.050	0.839	167	4; 43; 56; 57; 95; 109; 111; 124; 159; 167; 187; 188; 191; 192; 204; 203; 208; 222
q0.75	224	20	40	10	0.147	0.500	0.821	167; 171; 173; 175; 177; 179; 181; 183; 185; 186	1; 4; 26; 27; 43; 56; 57; 72; 86; 95; 97; 102; 107; 109; 111; 115; 120; 122; 123; 124; 159; 167; 171; 173; 175; 177; 179; 181; 183; 185; 186; 187; 188; 191; 192; 201; 204; 203; 208; 222
q0.8	224	20	27	2	0.123	0.100	0.808	167; 186	1; 4; 26; 43; 56; 57; 86; 95; 109; 111; 115; 120; 122; 123; 124; 159; 167; 186; 187; 188; 191; 192; 201; 204; 203; 208; 222
q0.85	224	20	23	1	0.108	0.050	0.817	167	1; 4; 43; 56; 57; 95; 109; 111; 120; 122; 123; 124; 159; 167; 187; 188; 191; 192; 201; 204; 203; 208; 222
q0.9	224	20	16	1	0.074	0.050	0.848	167	4; 43; 56; 57; 95; 109; 111; 124; 159; 167; 187; 188; 192; 204; 203; 208
q0.95	224	20	9	1	0.039	0.050	0.879	167	4; 95; 109; 111; 159; 167; 188; 192; 204

H.3 Rat malware session 2

Table H.3: Outcome using different threshold values for rat2 malware using algorithm 3

thresh-old type	total nr pro- cesses	total nr mali- cious pro- cesses	total nr above thresh- old	number malicious above threshold	FPR	TPR	ACC	process ids	all processes
mean	213	20	28	0	0.145	0	0.775		2; 4; 24; 34; 51; 52; 53; 65; 95; 96; 97; 111; 124; 141; 160; 161; 164; 171; 187; 189; 190; 191; 192; 193; 194; 202; 206; 207
q0.75	213	20	47	8	0.202	0.400	0.761	60; 72; 74; 76; 80; 82; 84; 85	1; 2; 4; 20; 24; 34; 51; 52; 53; 60; 65; 69; 72; 74; 76; 80; 82; 84; 85; 91; 94; 95; 96; 97; 101; 111; 114; 116; 124; 125; 141; 160; 161; 164; 165; 171; 187; 188; 189; 190; 191; 192; 193; 194; 202; 206; 207
q0.8	213	20	38	0	0.197	0	0.728		1; 2; 4; 24; 34; 51; 52; 53; 65; 69; 91; 94; 95; 96; 97; 101; 111; 114; 116; 124; 125; 141; 160; 161; 164; 165; 171; 187; 188; 189; 190; 191; 192; 193; 194; 202; 206; 207
q0.85	213	20	25	0	0.130	0	0.789		2; 4; 24; 34; 51; 52; 53; 95; 96; 97; 111; 124; 141; 160; 161; 164; 187; 189; 190; 192; 193; 194; 202; 206; 207
q0.9	213	20	11	0	0.057	0	0.854		4; 24; 34; 51; 111; 161; 187; 192; 194; 202; 206
q0.95	213	20	4	0	0.021	0	0.887		4; 24; 111; 161

H.4 Zeus malware session 1

Table H.4: Outcome using different threshold values for zeus1 malware using algorithm 3

thresh- old type	total nr pro- cesses	total nr malicious processes	total nr above thresh- old	number malicious above threshold	FPR	TPR	ACC	pro- cess ids	all processes
mean	156	5	12	0	0.079	0	0.891	4; 38; 41; 80; 81; 82; 107; 111; 113; 121; 134; 147	
q0.75	156	5	35	0	0.232	0	0.744	1; 2; 4; 12; 38; 39; 41; 42; 43; 73; 75; 80; 81; 82; 83; 84; 91; 97; 107; 111; 112; 113; 117; 118; 121; 127; 130; 132; 134; 135; 140; 141; 147; 150; 152	
q0.8	156	5	26	0	0.172	0	0.801	2; 4; 38; 39; 41; 42; 43; 80; 81; 82; 83; 84; 107; 111; 112; 113; 117; 121; 127; 130; 132; 134; 140; 147; 150; 152	
q0.85	156	5	19	0	0.126	0	0.846	4; 38; 39; 41; 43; 80; 81; 82; 83; 84; 107; 111; 113; 121; 127; 134; 140; 147; 150	
q0.9	156	5	8	0	0.053	0	0.917	4; 38; 41; 80; 107; 111; 113; 147	
q0.95	156	5	5	0	0.033	0	0.936	4; 107; 111; 113; 147	

H.5 Zeus malware session 2

Table H.5: Outcome using different threshold values for zeus2 malware using algorithm 3

thresh- old type	total nr pro- cesses	total nr mali- cious pro- cesses	total nr above thresh- old	number malicious above threshold	FPR	TPR	ACC	pro- cess ids	all processes
mean	191	2	19	0	0.101	0	0.890	4; 79; 80; 81; 112; 114; 115; 116; 117; 118; 119; 120; 123; 129; 130; 140; 154; 160; 164	
q0.75	191	2	38	0	0.201	0	0.791	1; 4; 26; 36; 37; 40; 59; 79; 80; 81; 82; 85; 92; 112; 114; 115; 116; 117; 118; 119; 120; 121; 123; 127; 129; 130; 137; 139; 140; 148; 151; 154; 160; 163; 164; 167; 176; 187	
q0.8	191	2	28	0	0.148	0	0.843	1; 4; 36; 37; 40; 59; 79; 80; 81; 82; 112; 114; 115; 116; 117; 118; 119; 120; 121; 123; 127; 129; 130; 139; 140; 154; 160; 164	
q0.85	191	2	21	0	0.111	0	0.880	4; 79; 80; 81; 82; 112; 114; 115; 116; 117; 118; 119; 120; 121; 123; 129; 130; 140; 154; 160; 164	
q0.9	191	2	13	0	0.069	0	0.921	4; 112; 114; 116; 117; 118; 119; 120; 123; 129; 130; 154; 160	
q0.95	191	2	6	0	0.032	0	0.958	4; 112; 114; 116; 117; 130	

H.5.1 Malicious heatmaps

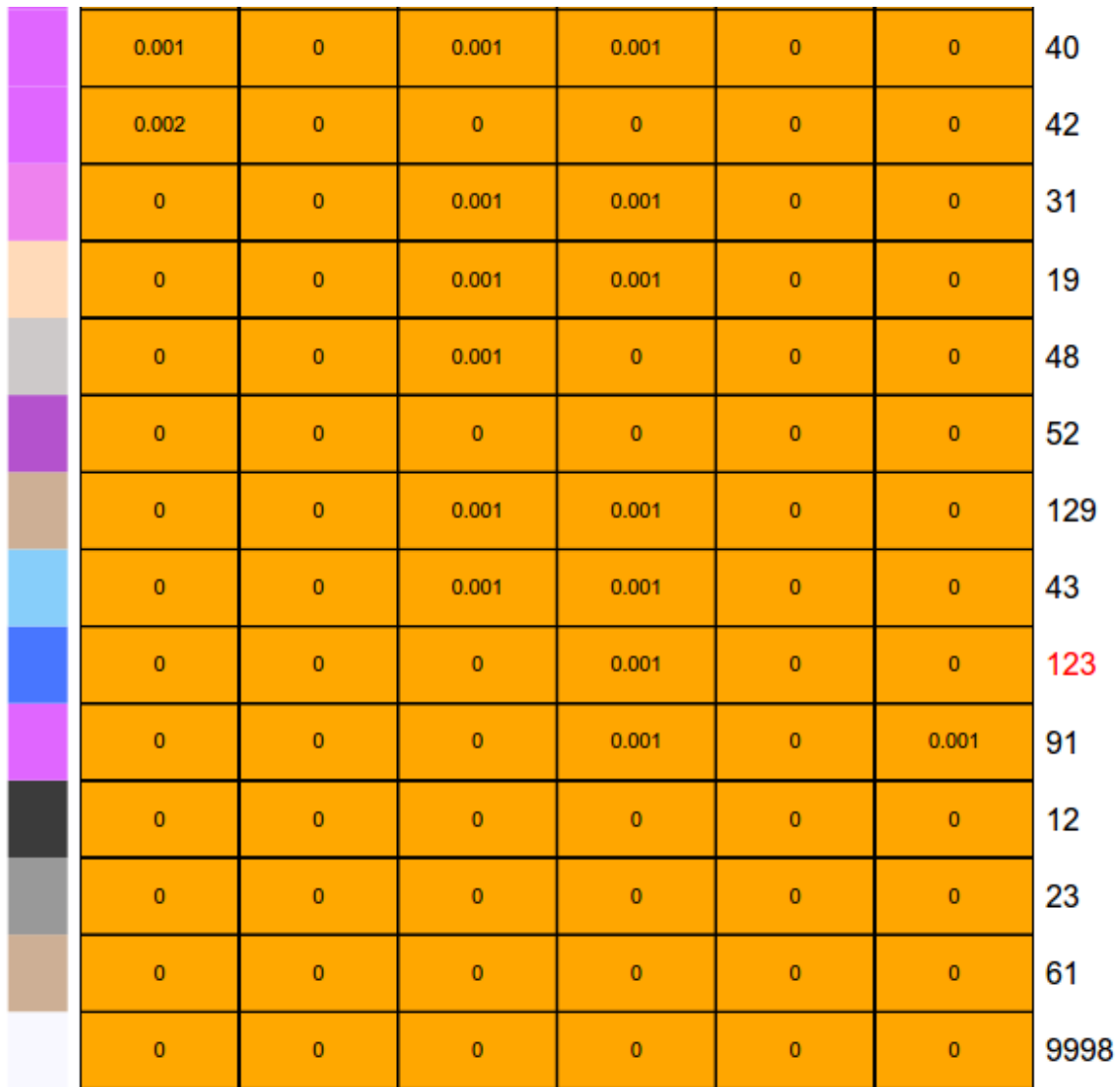


Figure H.1: Part 1 from the heatmap from the banking malware with the malicious processes

With the following column order: Registry, process create, thread create, ob, module load, filesystem

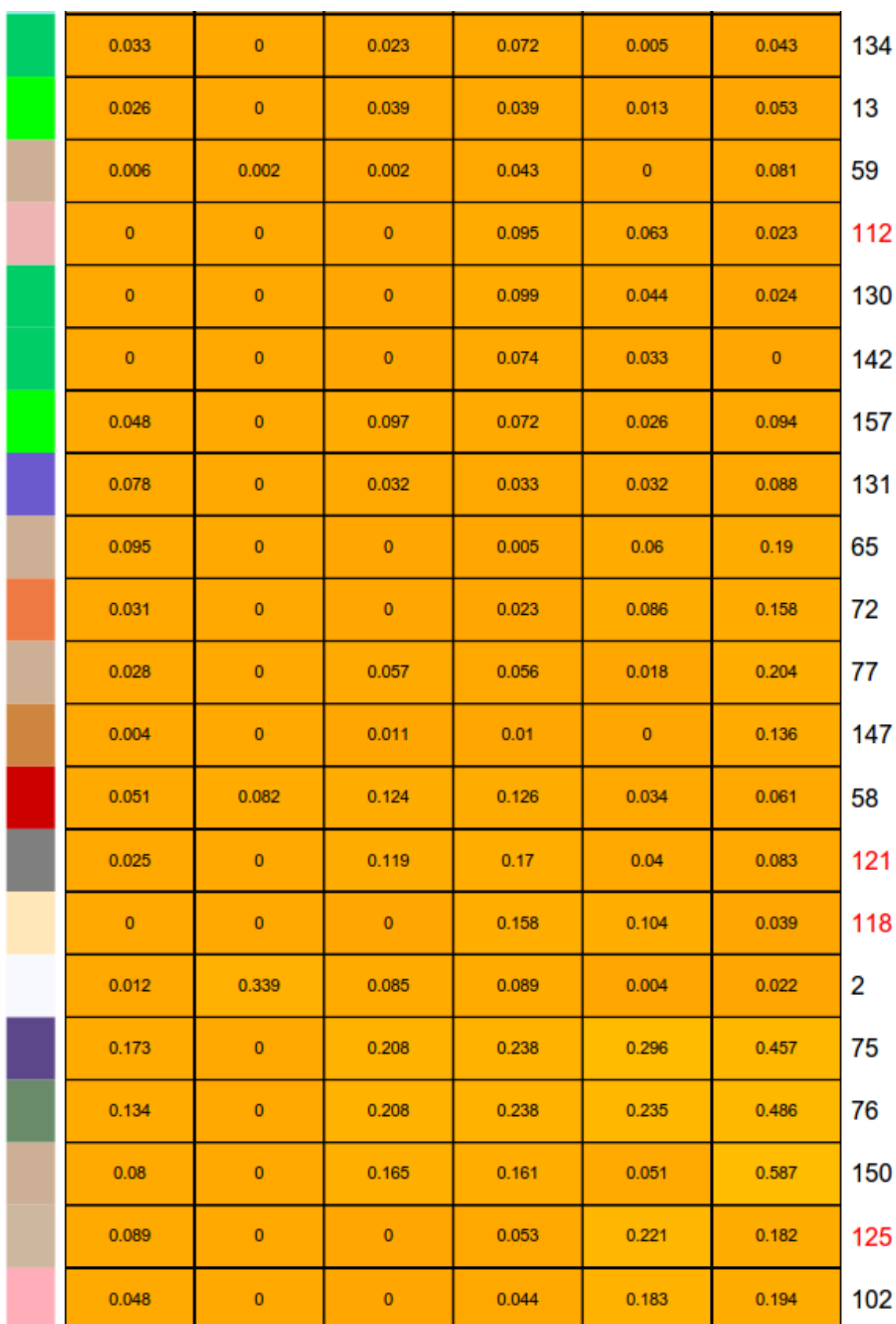


Figure H.2: Part 2 from the heatmap from banking malware with the malicious processes

With the following column order: Registry, process create, thread create, ob, module load, filesystem

	0	0.044	0.033	0.022	0.012	0.042	221
	0	0.015	0.022	0.038	0.011	0.037	142
	0	0.061	0.065	0.022	0.006	0.034	88
	0	0.062	0.063	0.012	0.012	0.028	144
	0	0.016	0.072	0.026	0.01	0.026	55
	0	0	0.071	0	0.047	0	173
	0	0	0.071	0	0.047	0	171
	0	0	0.071	0	0.047	0	175
	0	0	0.071	0	0.047	0	177
	0	0	0.071	0	0.047	0	179
	0	0	0.071	0	0.047	0	181
	0	0	0.071	0	0.047	0	185
	0	0	0.071	0	0.047	0	183
	0	0	0.073	0	0.048	0	186
	0	0.002	0.002	0.001	0.001	0.002	211
	0	0.002	0.002	0.001	0	0.002	84
	0	0.002	0.002	0.001	0.001	0.001	83
	0	0.002	0.002	0.001	0.001	0.001	210
	0	0.002	0.002	0.001	0.001	0.002	59

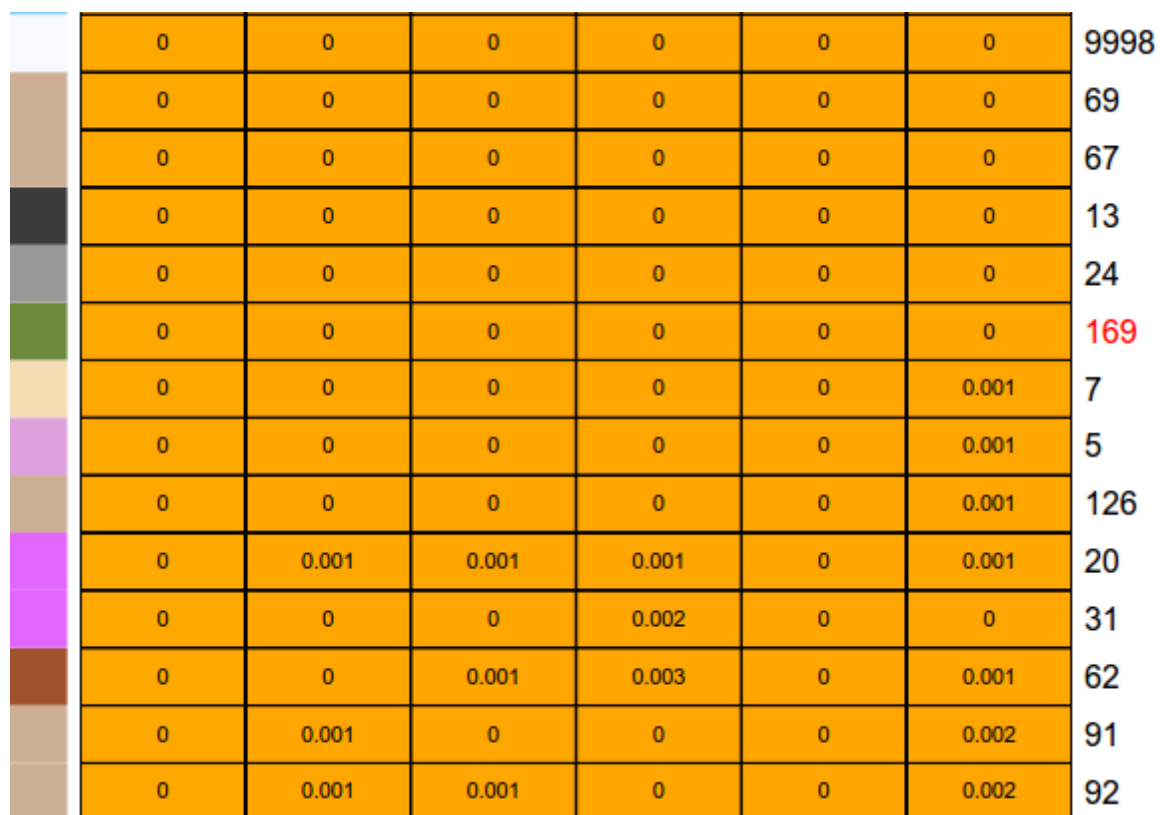
Figure H.3: Part 1 from the heatmap from rat session 1 with the malicious processes

With the following column order: process create, thread create, ob, registry, module load, filesystem

	0	0.003	0.004	0.001	0.001	0.002	214
	0	0.003	0.003	0.001	0.001	0.003	118
	0.002	0.002	0.003	0.001	0.001	0.001	58
	0.002	0.002	0.002	0.002	0	0	8
	0.003	0.002	0.002	0	0	0	121
	0.003	0.001	0.001	0	0	0	168
	0.003	0.001	0.004	0.001	0	0	10
	0	0.001	0.001	0	0	0.001	156
	0	0.001	0.001	0	0	0	196
	0	0.001	0.001	0	0	0	199

Figure H.4: Part 2 from the heatmap from rat session 1 with the malicious processes

With the following column order: process create, thread create, ob, registry, module load, filesystem



0	0	0	0	0	0	9998
0	0	0	0	0	0	69
0	0	0	0	0	0	67
0	0	0	0	0	0	13
0	0	0	0	0	0	24
0	0	0	0	0	0	169
0	0	0	0	0	0.001	7
0	0	0	0	0	0.001	5
0	0	0	0	0	0.001	126
0	0.001	0.001	0.001	0	0.001	20
0	0	0	0.002	0	0	31
0	0	0.001	0.003	0	0.001	62
0	0.001	0	0	0	0.002	91
0	0.001	0.001	0	0	0.002	92

Figure H.5: Part 3 from the heatmap from rat session 1 with the malicious processes

With the following column order: process create, thread create, ob, registry, module load, filesystem

	0	0.007	0.008	0.001	0.002	0.005	105
	0	0.007	0.008	0.001	0.002	0.005	131
	0	0.007	0.007	0.001	0.002	0.005	73
	0	0.007	0.007	0.001	0.002	0.005	147
	0	0.008	0.009	0.002	0.003	0.007	35
	0	0.009	0.006	0.001	0	0.004	9999
	0	0.005	0.006	0.001	0	0.001	30
	0.004	0.004	0.011	0.001	0.001	0.002	215
	0	0.002	0.011	0.003	0	0.002	25
	0	0	0.012	0	0.008	0	176
	0	0	0.012	0	0.008	0	174
	0	0	0.012	0	0.008	0	180
	0	0	0.012	0	0.008	0	178
	0	0	0.012	0	0.008	0	184
	0	0	0.012	0	0.008	0	182
	0	0	0.012	0	0.008	0	172
	0	0	0.007	0	0.004	0.002	170
	0	0.009	0.01	0.007	0.012	0.019	81
	0	0.009	0.01	0.006	0.01	0.02	82
	0	0.015	0.016	0.012	0.006	0.023	136
	0	0.006	0.006	0.003	0.002	0.023	116
	0	0.006	0.006	0.003	0.002	0.022	202
	0	0.005	0.004	0.002	0.001	0.016	68
	0	0.004	0.004	0.002	0.001	0.015	117
	0	0.004	0.003	0.002	0.001	0.013	114

Figure H.6: Part 4 from the heatmap from rat session 1 with the malicious processes

With the following column order: process create, thread create, ob, registry, module load, filesystem

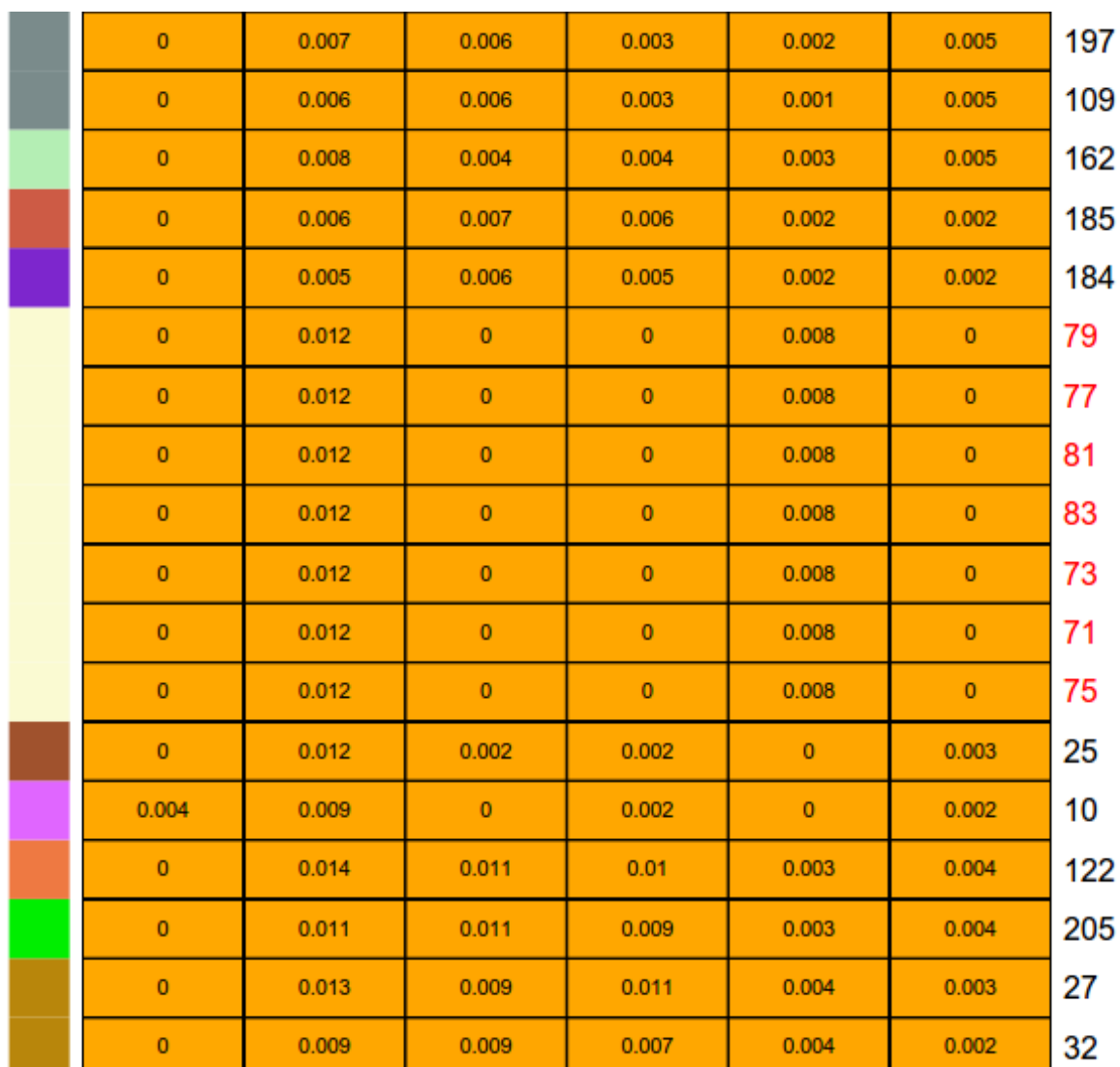


Figure H.7: Part 1 from the heatmap from rat session 2 with the malicious processes

With the following column order: process create, ob, filesystem, thread create, module load, registry

	0	0	0	0	0	0	23
	0	0	0	0	0	0	89
	0	0	0	0	0	0	41
	0	0	0	0	0	0	13
	0	0	0	0	0	0	180
	0	0	0	0	0	0	178
	0	0	0	0	0	0	181
	0	0	0	0	0	0	64
	0	0	0	0	0	0	182
	0	0.001	0.001	0	0	0	127
	0	0	0.001	0	0	0	99
	0.001	0.001	0.002	0	0	0	7
	0	0	0.002	0.001	0	0	5

Figure H.8: Part 2 from the heatmap from rat session 2 with the malicious processes

With the following column order: process create, ob, filesystem, thread create, module load, registry

	0	0.004	0.001	0.004	0.001	0.001	151
	0	0.004	0.003	0.003	0.001	0.001	153
	0	0.005	0.002	0.003	0	0.001	3
	0.004	0.005	0.002	0.005	0.001	0.001	145
	0.003	0.003	0.002	0.003	0.001	0.001	54
	0	0.006	0.001	0	0.004	0	66
	0.004	0.002	0.001	0.002	0	0.001	166
	0.003	0.002	0.001	0.002	0	0	167
	0.004	0.001	0.001	0.001	0	0	176
	0.003	0.001	0.001	0.001	0	0	100
	0.003	0.001	0	0.001	0	0	62
	0.002	0.001	0	0.001	0	0	163
	0.006	0.002	0.002	0.001	0.001	0	135
	0.007	0.002	0	0.002	0	0	147
	0.005	0.003	0.003	0.003	0.001	0.005	36
	0.004	0.004	0.002	0.004	0	0.005	8

Figure H.9: Part 3 from the heatmap from rat session 2 with the malicious processes

With the following column order: process create, ob, filesystem, thread create, module load, registry

	0	0.001	0.006	0	0.003	0.016	175
	0	0.001	0.015	0	0.002	0.013	186
	0.002	0.007	0.003	0.003	0.001	0.017	98
	0.001	0.007	0.006	0.006	0	0.012	39
	0	0.001	0.009	0	0.003	0.026	174
	0	0.022	0	0	0.01	0.01	1
	0	0.024	0.006	0	0.011	0	63
	0	0.04	0	0	0.026	0	70
	0	0.039	0.02	0.039	0.007	0.006	67
	0	0.037	0.019	0.037	0.007	0.005	200
	0	0.037	0.021	0.036	0.008	0.006	46
	0	0.04	0.022	0.039	0.009	0.007	110
	0	0.039	0.022	0.037	0.008	0.007	129
	0	0.039	0.016	0.039	0.007	0.005	68
	0	0.037	0.016	0.037	0.006	0.005	33

Figure H.10: Part 4 from the heatmap from rat session 2 with the malicious processes

With the following column order: process create, ob, filesystem, thread create, module load, registry

	0	0.018	0.036	0.016	0.017	0.008	93
	0	0.012	0.051	0.012	0.004	0.007	92
	0	0.009	0.033	0.009	0.003	0.005	103
	0	0.068	0	0	0.045	0	74
	0	0.068	0	0	0.045	0	72
	0	0.068	0	0	0.045	0	85
	0	0.068	0	0	0.045	0	82
	0	0.068	0	0	0.045	0	80
	0	0.068	0	0	0.045	0	84
	0	0.059	0	0	0.039	0	76
	0	0.053	0	0	0.035	0	78
	0	0.052	0.013	0	0.034	0	198
	0	0.028	0.1	0.028	0.009	0.014	115
	0.002	0.039	0.073	0.003	0	0.006	88
	0	0.016	0.069	0	0.031	0.02	139
	0	0.002	0.071	0	0.023	0.036	117
	0	0.025	0.053	0.025	0.006	0.085	195
	0.036	0.054	0.066	0.054	0.024	0.037	94
	0	0.079	0.045	0.074	0.009	0.031	196
	0.071	0.02	0.115	0.018	0.027	0.01	60
	0	0.005	0.13	0	0.057	0.087	91
	0	0.001	0.169	0	0.015	0.025	116
	0	0.03	0.204	0	0.111	0.04	130

Figure H.11: Part 5 from the heatmap from from rat session 2 with the malicious processes

With the following column order: process create, ob, filesystem, thread create, module load, registry

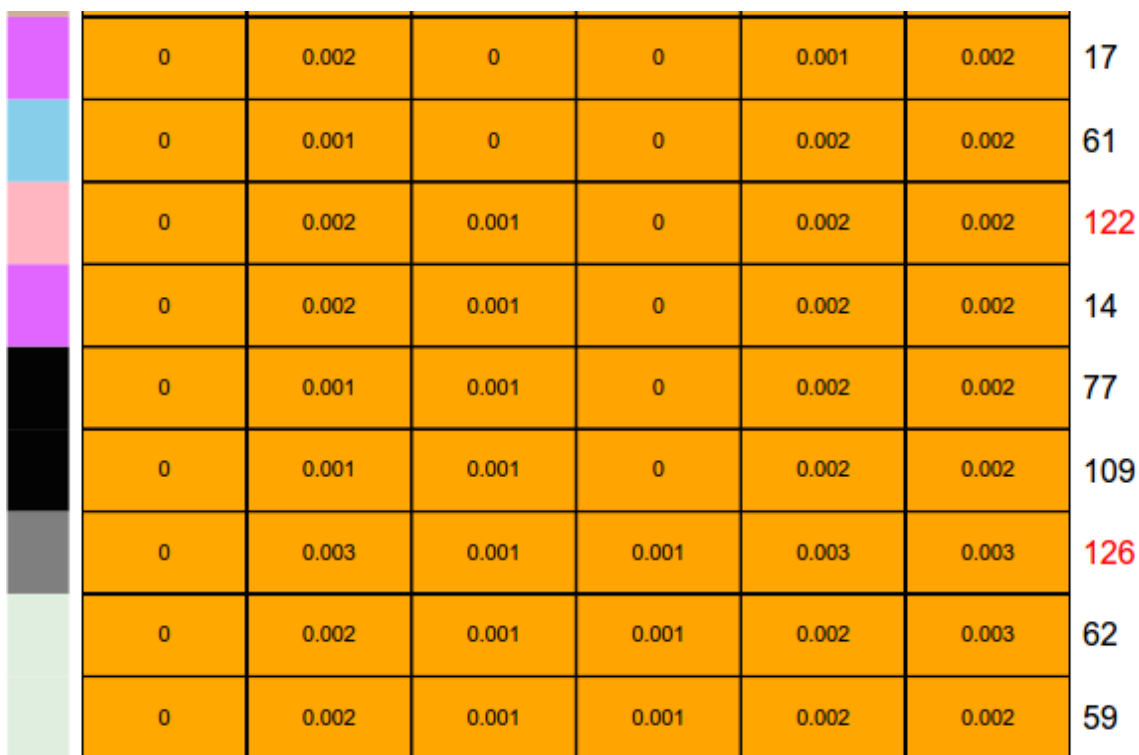


Figure H.12: Part 1 from the heatmap from zeus session 1 with the malicious processes

With the following column order: process create, filesystem, registry, module load, thread create, ob

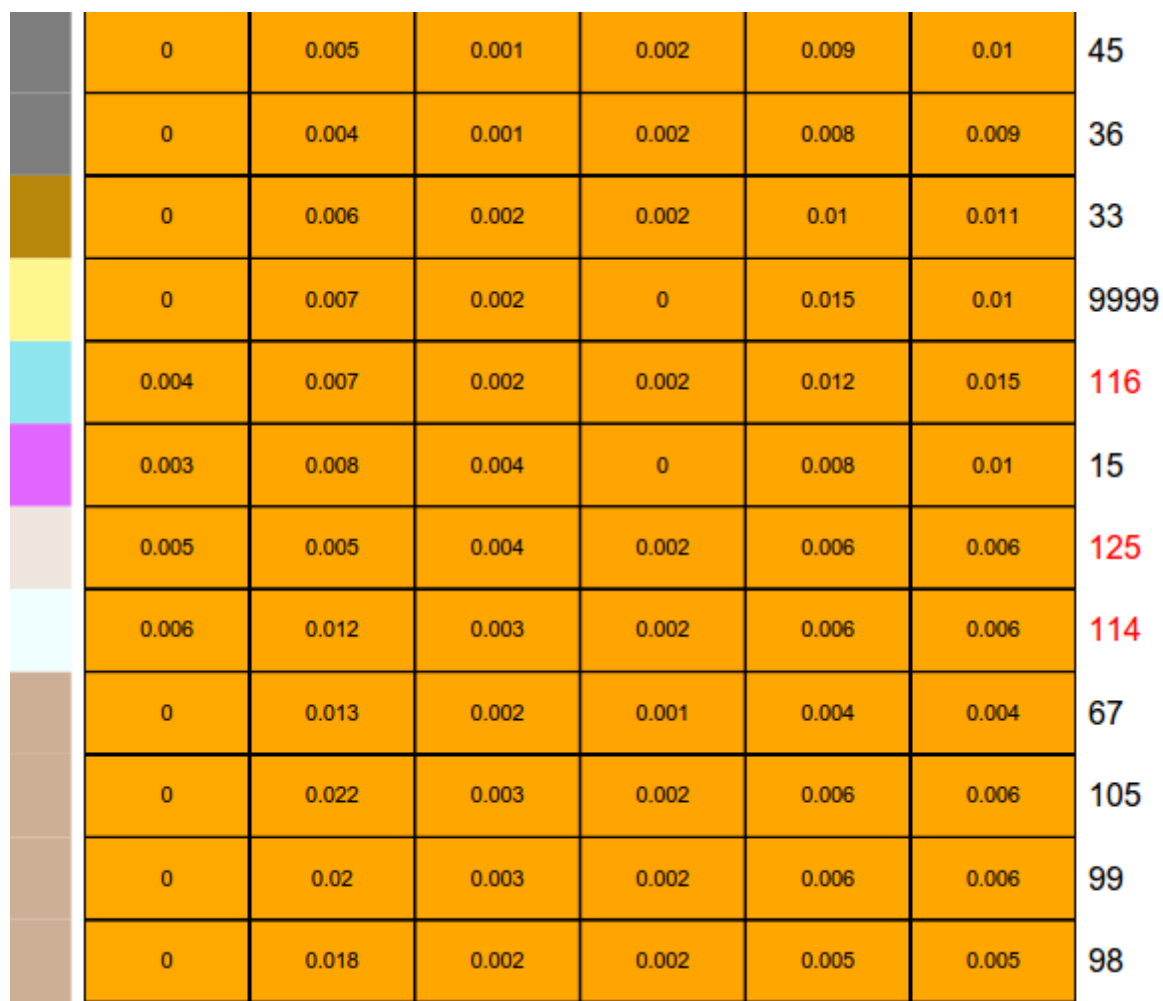


Figure H.13: Part 2 from the heatmap from zeus session 1 with the malicious processes

With the following column order: process create, filesystem, registry, module load, thread create, ob

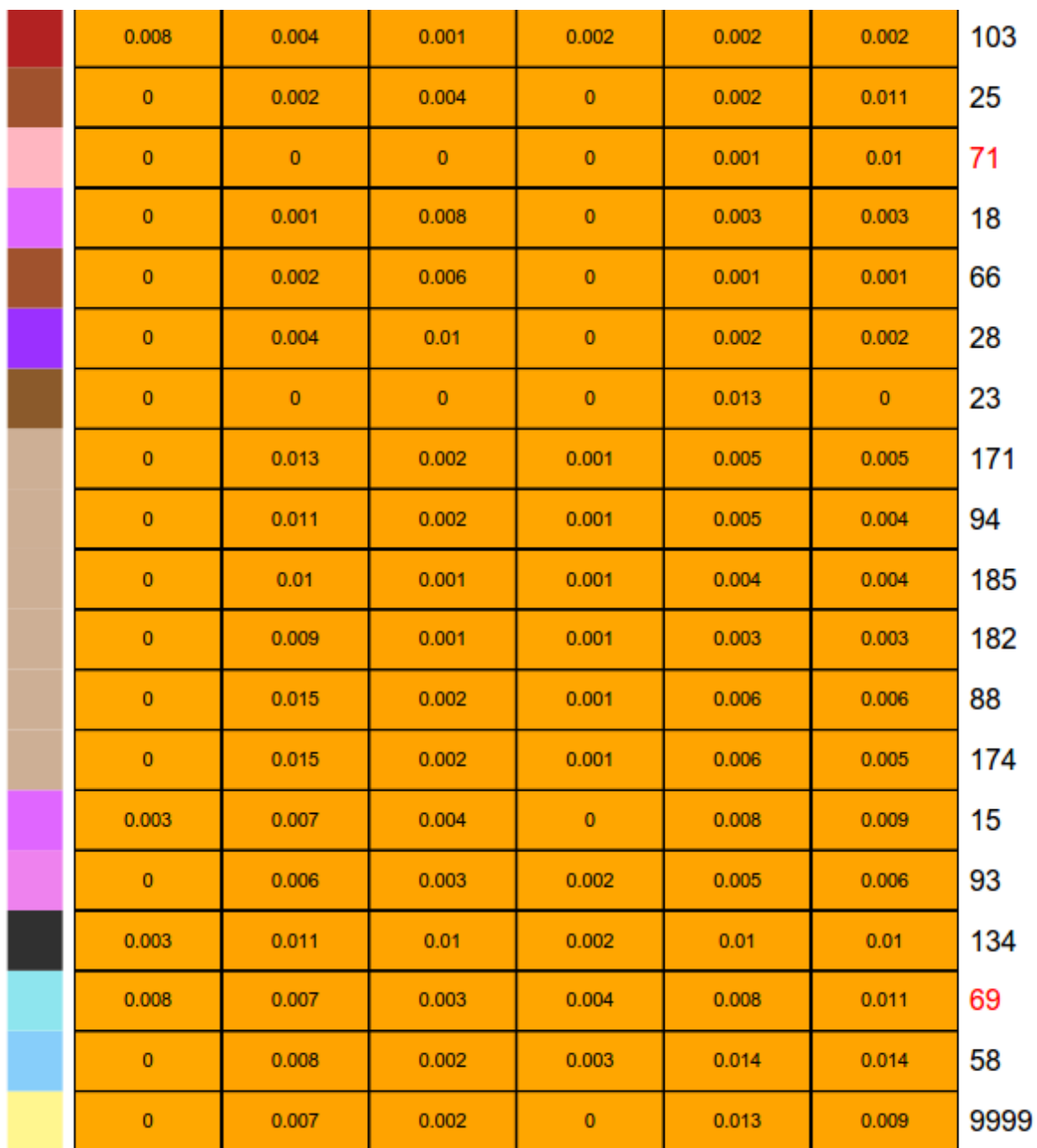


Figure H.14: Part 1 from the heatmap from zeus session 2 with the malicious processes

process create, filesystem, registry, module load, thread create, ob

I

Ranked malicious processes

The tables in this chapter show for every malware dataset and all three algorithms the process which are marked malicious, the sum of the distances and if the process is really malicious. In every table lines are drawn for each threshold type. Furthermore for every ranking the top five non-malicious are taken and shown in which datasets the executable is present.

I.1 Algorithm 1: Ranked malicious marked processes

Table I.1: Ranking of the processes based on the distance for banking malware using algorithm 1. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.

	unique ids	distance sum	malicious
1	4	23.489	No
2	135	21.236	No
3	120	7.946	Yes
4	27	5.383	No
5	119	4.587	No
6	143	4.584	No
7	153	4.438	No
8	159	4.007	No
9	73	3.600	No
10	122	3.293	Yes
11	113	2.770	No
12	74	2.471	No
13	116	2.378	Yes
14	111	1.613	Yes
15	126	1.426	Yes
16	124	1.338	Yes
17	125	1.013	Yes
18	146	0.987	No
19	150	0.968	No
20	117	0.960	Yes
21	76	0.938	No
22	75	0.922	No
23	1	0.755	No
24	154	0.692	No
25	90	0.480	No
26	157	0.435	No
27	2	0.422	No
28	121	0.407	Yes
29	65	0.402	No
30	118	0.399	Yes
31	58	0.289	No
32	112	0.265	Yes
33	13	0.238	No
34	131	0.183	No
35	134	0.151	No

Table I.2: The five highest ranked benign processes for banking malware using algorithm 1. Also showing the clean datasets in which the executable is present.

unique ids	dis- tance sum	ma- li- cious	dfs			
4	23.489	No	malware_rat_session1 malware_zeus_session2	malware_rat_session2 zz_win8_1604_avond	malware_zeus_session1 zz_win8_1604	zz_win8_1704 zz_win8_1804
135	21.236	No	malware_rat_session1 malware_zeus_session2	malware_rat_session2	malware_zeus_session1	
27	5.383	No	malware_rat_session2	zz_win8_1604_avond	zz_win8_1604	zz_win8_1704 zz_win8_1804
119	4.587	No	malware_rat_session1 malware_zeus_session2	malware_rat_session2 zz_win8_1604_avond	malware_zeus_session1 zz_win8_1604	zz_win8_1704 zz_win8_1804
143	4.584	No	malware_rat_session1 malware_zeus_session2	malware_rat_session2 zz_win8_1604_avond	malware_zeus_session1 zz_win8_1604	zz_win8_1704 zz_win8_1804

Table I.3: Ranking of the processes based on the distance for rat session 1 using algorithm 1. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.

	unique ids	distance sum	malicious
1	4	18.626	No
2	109	10.674	No
3	111	8.652	No
4	95	7.347	No
5	204	6.634	No
6	167	4.996	Yes
7	192	4.535	No
8	188	4.051	No
9	159	3.804	No
10	124	3.709	No
11	43	2.562	No
12	56	1.429	No
13	208	1.386	No
14	122	1.315	No
15	123	1.266	No
16	57	1.130	No
17	203	1.126	No
18	102	1.095	No
19	191	0.927	No
20	187	0.896	No
21	201	0.830	No
22	222	0.789	No
23	1	0.720	No
24	120	0.662	No
25	27	0.656	No
26	72	0.590	No
27	115	0.494	No
28	86	0.394	No
29	26	0.361	No
30	107	0.328	No
31	186	0.272	Yes
32	171	0.267	Yes
33	173	0.267	Yes
34	175	0.267	Yes
35	177	0.267	Yes
36	179	0.267	Yes
37	181	0.267	Yes
38	185	0.267	Yes
39	183	0.264	Yes
40	97	0.230	No

Table I.4: The five highest ranked benign processes for rat session 1 using algorithm 1. Also showing the clean datasets in which the executable is present.

unique ids	dis- tance sum	ma- li- cious	dfs			
4	18.626	No	malware_bank malware_zeus_session2	malware_rat_session2 zz_win8_1604_avond	malware_zeus_session1 zz_win8_1604	zz_win8_1704 zz_win8_1804
109	10.674	No	malware_zeus_session1	malware_zeus_session2	zz_win8_1604_avond zz_win8_1604	zz_win8_1704 zz_win8_1804
111	8.652	No	malware_zeus_session1	malware_zeus_session2	zz_win8_1604_avond zz_win8_1604	zz_win8_1704 zz_win8_1804
95	7.347	No	malware_bank	malware_rat_session2	malware_zeus_session1 malware_zeus_session2	
204	6.634	No	malware_bank	malware_rat_session2	malware_zeus_session1 malware_zeus_session2	

Table I.5: Ranking of the processes based on the distance for rat session 2 using algorithm 1. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third q85, fourth the mean, the fifth q80 and the bottom line is q75.

	unique ids	distance sum	malicious
1	4	20.393	No
2	111	8.395	No
3	161	4.538	No
4	206	4.142	No
5	194	3.791	No
6	193	3.480	No
7	24	3.464	No
8	95	3.126	No
9	191	2.780	No
10	207	2.615	No
11	192	2.438	No
12	34	2.156	No
13	141	1.750	No
14	189	1.711	No
15	51	1.640	No
16	202	1.639	No
17	190	1.632	No
18	187	1.488	No
19	97	1.361	No
20	171	1.361	No
21	96	1.360	No
22	164	1.162	No
23	53	1.048	No
24	65	0.941	No
25	52	0.939	No
26	160	0.927	No
27	2	0.802	No
28	101	0.795	No
29	1	0.740	No
30	124	0.700	No
31	165	0.662	No
32	125	0.584	No
33	188	0.533	No
34	69	0.485	No
35	91	0.379	No
36	94	0.348	No
37	116	0.342	No
38	114	0.337	No
39	60	0.313	Yes
40	20	0.287	No
41	84	0.253	Yes
42	80	0.252	Yes
43	82	0.252	Yes
44	72	0.252	Yes
45	74	0.252	Yes
46	85	0.252	Yes
47	76	0.218	Yes

Table I.6: The five highest ranked benign processes for rat session 2 using algorithm 1. Also showing the clean datasets in which the executable is present.

unique ids	dis- tance sum	ma- li- cious	dfs			
4	20.393	No	malware_bank malware_zeus_session2	malware_rat_session1 zz_win8_1604_avond zz_win8_1804	malware_zeus_session1 zz_win8_1604	zz_win8_1704
111	8.395	No	malware_bank	malware_rat_session1 malware_zeus_session2	malware_zeus_session1	
161	4.538	No		no		
206	4.142	No		no		
194	3.791	No		zz_win8_1604	zz_win8_1704	

Table I.7: Ranking of the processes based on the distance for zeus session 1 using algorithm 1. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.

	unique ids	distance sum	malicious
1	4	23.482	No
2	111	14.668	No
3	113	12.345	No
4	147	4.503	No
5	107	4.418	No
6	81	3.265	No
7	80	3.196	No
8	42	2.784	No
9	38	2.616	No
10	82	2.035	No
11	41	1.690	No
12	132	1.507	No
13	141	1.484	No
14	83	1.387	No
15	140	1.375	No
16	134	1.341	No
17	91	1.121	No
18	121	1.023	No
19	43	1.002	No
20	127	0.980	No
21	150	0.841	No
22	112	0.831	No
23	84	0.799	No
24	39	0.670	No
25	1	0.669	No
26	2	0.553	No
27	117	0.506	No
28	75	0.405	No
29	152	0.389	No
30	130	0.381	No
31	97	0.310	No
32	73	0.301	No
33	135	0.291	No
34	118	0.273	No
35	12	0.263	No

Table I.8: The five highest ranked benign processes for zeus session 1 using algorithm 1. Also showing the clean datasets in which the executable is present.

unique ids	dis- tance sum	ma- li- cious	dfs			
4	23.482	No	malware.bank	malware_rat_session1	malware_rat_session2	
			malware_zeus_session2	zz_win8_1604_avond	zz_win8_1604	zz_win8_1704
				zz_win8_1804		
111	14.668	No	malware_rat_session1	malware_zeus_session2	zz_win8_1604_avond	
				zz_win8_1604	zz_win8_1704	zz_win8_1804
113	12.345	No	malware_rat_session1	malware_zeus_session2	zz_win8_1604_avond	
				zz_win8_1604	zz_win8_1704	zz_win8_1804
147	4.503	No	malware.bank	malware_rat_session1	malware_rat_session2	
			malware_zeus_session2	zz_win8_1604_avond	zz_win8_1604	zz_win8_1704
				zz_win8_1804		
107	4.418	No	malware.bank	malware_rat_session1	malware_rat_session2	
				malware_zeus_session2		

Table I.9: Ranking of the processes based on the distance for zeus session 2 using algorithm 1. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.

	unique ids	distance sum	malicious
1	4	23.482	No
2	112	14.718	No
3	114	12.306	No
4	116	5.340	No
5	130	5.331	No
6	117	3.692	No
7	82	3.517	No
8	79	3.310	No
9	119	3.250	No
10	163	3.092	No
11	154	2.985	No
12	81	2.970	No
13	123	2.889	No
14	80	2.786	No
15	118	2.670	No
16	40	2.652	No
17	36	2.550	No
18	120	2.342	No
19	160	2.091	No
20	129	2.019	No
21	140	1.845	No
22	115	1.619	No
23	164	1.196	No
24	59	0.957	No
25	139	0.930	No
26	121	0.858	No
27	37	0.781	No
28	1	0.683	No
29	137	0.679	No
30	127	0.676	No
31	151	0.550	No
32	167	0.548	No
33	92	0.533	No
34	26	0.518	No
35	176	0.509	No
36	85	0.453	No
37	187	0.432	No
38	148	0.409	No

Table I.10: The five highest ranked benign processes for zeus session 2 using algorithm 1. Also showing the clean datasets in which the executable is present.

unique ids	dis- tance sum	ma- li- cious	dfs			
4	23.482	No	malware.bank malware_zeus_session1	malware_rat_session1 zz_win8_1604_avond	malware_rat_session2 zz_win8_1604	zz_win8_1704 zz_win8_1804
112	14.718	No	malware_rat_session1	malware_zeus_session1 zz_win8_1604	zz_win8_1604_avond zz_win8_1704	zz_win8_1804
114	12.306	No	malware_rat_session1	malware_zeus_session1 zz_win8_1604	zz_win8_1604_avond zz_win8_1704	zz_win8_1804
116	5.340	No	malware.bank malware_zeus_session1	malware_rat_session1 zz_win8_1604_avond	malware_rat_session2 zz_win8_1604	zz_win8_1704
130	5.331	No	malware.bank malware_zeus_session1	malware_rat_session1 zz_win8_1604_avond	malware_rat_session2 zz_win8_1604	zz_win8_1704

I.2 Algorithm 2: Ranked malicious marked processes

Table I.11: Ranking of the processes based on the distance for banking malware using algorithm 2. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third q85, fourth the mean, the fifth q80 and the bottom line is q75.

	unique ids	distance sum	malicious
1	4	32.514	No
2	135	24.814	No
3	159	10.431	No
4	27	9.995	No
5	153	9.905	No
6	120	7.946	Yes
7	119	7.285	No
8	143	7.281	No
9	131	7.088	No
10	111	5.875	Yes
11	84	4.023	No
12	113	3.646	No
13	73	3.600	No
14	122	3.293	Yes
15	74	2.471	No
16	116	2.378	Yes
17	146	2.041	No
18	154	1.523	No
19	126	1.426	Yes
20	124	1.338	Yes
21	117	1.079	Yes
22	125	1.013	Yes
23	150	0.968	No
24	76	0.938	No
25	1	0.934	No
26	65	0.501	No
27	157	0.482	No
28	90	0.480	No
29	147	0.451	No
30	2	0.422	No
31	118	0.399	Yes
32	134	0.304	No

Table I.12: The five highest ranked benign processes for banking malware using algorithm 2. Also showing the clean datasets in which the executable is present.

unique ids	dis- tance sum	ma- li- cious	dfs			
4	32.514	No	malware_rat_session1 malware_zeus_session2	malware_rat_session2 zz_win8_1604_avond	malware_zeus_session1 zz_win8_1604	zz_win8_1704 zz_win8_1804
135	24.814	No	malware_rat_session1	malware_rat_session2	malware_zeus_session1 malware_zeus_session2	
159	10.431	No	malware_rat_session1	malware_rat_session2	malware_zeus_session1 malware_zeus_session2	
27	9.995	No	malware_rat_session2	zz_win8_1604_avond	zz_win8_1604	zz_win8_1704 zz_win8_1804
153	9.905	No	malware_rat_session1	malware_rat_session2	malware_zeus_session1 malware_zeus_session2	

Table I.13: Ranking of the processes based on the distance for rat session 1 using algorithm 2. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90 and the mean, third q85, the fourth q80 and the bottom line is q75.

	unique ids	distance sum	malicious
1	109	25.014	No
2	4	24.599	No
3	111	20.723	No
4	110	15.170	No
5	204	11.975	No
6	95	11.378	No
7	43	10.722	No
8	45	10.618	No
9	162	9.716	No
10	167	8.773	Yes
11	163	7.118	No
12	188	6.630	No
13	124	6.248	No
14	159	6.012	No
15	192	5.543	No
16	93	3.643	No
17	103	3.488	No
18	203	2.131	No
19	122	1.775	No
20	208	1.521	No
21	56	1.429	No
22	222	1.284	No
23	123	1.266	No
24	187	1.198	No
25	57	1.130	No
26	191	1.100	No
27	1	0.885	No
28	201	0.830	No
29	72	0.718	No
30	213	0.695	No
31	120	0.671	No
32	75	0.657	No
33	115	0.636	No
34	194	0.611	No
35	86	0.453	No
36	96	0.405	No
37	26	0.361	No
38	186	0.272	Yes
39	171	0.267	Yes
40	173	0.267	Yes
41	175	0.267	Yes
42	177	0.267	Yes
43	179	0.267	Yes
44	181	0.267	Yes
45	185	0.267	Yes
46	183	0.264	Yes

Table I.14: The five highest ranked benign processes for rat session 1 using algorithm 2. Also showing the clean datasets in which the executable is present.

unique ids	dis- tance sum	ma- li- cious	dfs			
109	25.014	No	malware_zeus_session1	malware_zeus_session2	zz_win8_1604_avond	
			zz_win8_1604	zz_win8_1704	zz_win8_1804	
4	24.599	No	malware_bank	malware_rat_session2	malware_zeus_session1	
			malware_zeus_session2	zz_win8_1604_avond	zz_win8_1604	zz_win8_1704
				zz_win8_1804		
111	20.723	No	malware_zeus_session1	malware_zeus_session2	zz_win8_1604_avond	
			zz_win8_1604	zz_win8_1704	zz_win8_1804	
110	15.170	No	malware_bank	malware_rat_session2	malware_zeus_session1	
			malware_zeus_session2	zz_win8_1604_avond	zz_win8_1604	zz_win8_1704
				zz_win8_1804		
204	11.975	No	malware_bank	malware_rat_session2	malware_zeus_session1	
				malware_zeus_session2		

Table I.15: Ranking of the processes based on the distance for rat session 2 using algorithm 2. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.

	unique ids	distance sum	malicious
1	4	27.284	No
2	111	8.395	No
3	199	6.871	No
4	125	6.690	No
5	124	6.588	No
6	165	6.279	No
7	202	5.817	No
8	206	5.254	No
9	34	4.852	No
10	24	4.572	No
11	161	4.538	No
12	105	4.019	No
13	194	3.791	No
14	193	3.480	No
15	207	3.293	No
16	95	3.126	No
17	192	3.059	No
18	191	2.780	No
19	141	2.747	No
20	187	1.892	No
21	171	1.775	No
22	189	1.711	No
23	65	1.699	No
24	51	1.640	No
25	190	1.632	No
26	97	1.361	No
27	96	1.360	No
28	164	1.212	No
29	53	1.048	No
30	52	0.939	No
31	160	0.927	No
32	1	0.914	No
33	2	0.802	No
34	101	0.795	No
35	69	0.592	No
36	188	0.533	No
37	91	0.396	No
38	60	0.390	Yes
39	116	0.357	No
40	195	0.349	No
41	94	0.348	No

Table I.16: The five highest ranked benign processes for rat session 2 using algorithm 2. Also showing the clean datasets in which the executable is present.

unique ids	dis- tance sum	ma- li- cious	dfs			
4	27.284	No	malware_bank malware_zeus_session2	malware_rat_session1 zz_win8_1604_avond	malware_zeus_session1 zz_win8_1604	zz_win8_1704 zz_win8_1804
111	8.395	No	malware_bank	malware_rat_session1 malware_zeus_session2	malware_zeus_session1	
199	6.871	No	malware_bank malware_zeus_session2	malware_rat_session1 zz_win8_1604_avond	malware_zeus_session1 zz_win8_1604	zz_win8_1704 zz_win8_1804
125	6.690	No	malware_bank malware_zeus_session2	malware_rat_session1 zz_win8_1604_avond	malware_zeus_session1 zz_win8_1604	zz_win8_1704 zz_win8_1804
124	6.588	No	malware_bank malware_zeus_session2	malware_rat_session1 zz_win8_1604	malware_zeus_session1 zz_win8_1704	zz_win8_1804

Table I.17: Ranking of the processes based on the distance for zeus session 1 using algorithm 2. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.

	unique ids	distance sum	malicious
1	4	32.503	No
2	111	29.827	No
3	113	21.907	No
4	112	15.990	No
5	40	10.839	No
6	38	10.514	No
7	107	7.764	No
8	147	7.242	No
9	118	6.929	No
10	134	6.744	No
11	135	6.643	No
12	92	3.523	No
13	81	3.265	No
14	80	3.196	No
15	42	2.784	No
16	132	2.092	No
17	82	2.035	No
18	41	1.690	No
19	127	1.543	No
20	83	1.387	No
21	140	1.375	No
22	121	1.246	No
23	150	1.206	No
24	43	1.002	No
25	1	0.810	No
26	84	0.799	No
27	39	0.779	No
28	75	0.729	No
29	142	0.719	No
30	117	0.669	No
31	2	0.553	No
32	153	0.473	No
33	152	0.407	No
34	130	0.381	No

Table I.18: The five highest ranked benign processes for zeus session 1 using algorithm 2. Also showing the clean datasets in which the executable is present.

unique ids	dis- tance sum	ma- li- cious	dfs			
4	32.503	No	malware.bank malware_zeus_session2	malware_rat_session1 zz_win8_1604_avond	malware_rat_session2 zz_win8_1604	zz_win8_1704 zz_win8_1804
111	29.827	No	malware_rat_session1	malware_zeus_session2	zz_win8_1604_avond	zz_win8_1604 zz_win8_1704 zz_win8_1804
113	21.907	No	malware_rat_session1	malware_zeus_session2	zz_win8_1604_avond	zz_win8_1604 zz_win8_1704 zz_win8_1804
112	15.990	No	malware.bank malware_zeus_session2	malware_rat_session1 zz_win8_1604_avond	malware_rat_session2 zz_win8_1604	zz_win8_1704 zz_win8_1804
40	10.839	No	malware_rat_session1	malware_zeus_session2	zz_win8_1604_avond	zz_win8_1604 zz_win8_1704 zz_win8_1804

Table I.19: Ranking of the processes based on the distance for zeus session 2 using algorithm 2. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.

	unique ids	distance sum	malicious
1	4	32.503	No
2	112	29.887	No
3	114	21.837	No
4	113	14.955	No
5	38	10.848	No
6	36	10.794	No
7	54	7.169	No
8	53	7.166	No
9	116	5.355	No
10	130	5.331	No
11	117	3.810	No
12	82	3.517	No
13	79	3.310	No
14	119	3.274	No
15	154	2.988	No
16	81	2.970	No
17	123	2.939	No
18	80	2.786	No
19	118	2.670	No
20	40	2.652	No
21	120	2.342	No
22	160	2.091	No
23	129	2.019	No
24	140	1.845	No
25	115	1.619	No
26	164	1.206	No
27	59	1.143	No
28	37	0.930	No
29	139	0.930	No
30	121	0.892	No
31	109	0.860	No
32	1	0.830	No
33	127	0.795	No
34	167	0.699	No
35	92	0.651	No
36	176	0.597	No
37	188	0.596	No

Table I.20: The five highest ranked benign processes for zeus session 2 using algorithm 2. Also showing the clean datasets in which the executable is present.

unique ids	dis- tance sum	ma- li- cious	dfs			
4	32.503	No	malware.bank malware_zeus_session1	malware_rat_session1 zz_win8_1604_avond	malware_rat_session2 zz_win8_1604	zz_win8_1704 zz_win8_1804
112	29.887	No	malware_rat_session1	malware_zeus_session1 zz_win8_1604	zz_win8_1604_avond zz_win8_1704	zz_win8_1804
114	21.837	No	malware_rat_session1	malware_zeus_session1 zz_win8_1604	zz_win8_1604_avond zz_win8_1704	zz_win8_1804
113	14.955	No	malware.bank malware_zeus_session1	malware_rat_session1 zz_win8_1604_avond	malware_rat_session2 zz_win8_1604	zz_win8_1704 zz_win8_1804
38	10.848	No	malware_rat_session1	malware_zeus_session1 zz_win8_1604	zz_win8_1604_avond zz_win8_1704	zz_win8_1804

I.3 Algorithm 3: Ranked malicious marked processes

Table I.21: Ranking of the processes based on the distance for banking malware using algorithm 3. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.

	unique ids	distance sum	malicious
1	4	32.514	No
2	135	24.814	No
3	120	8.284	Yes
4	27	7.616	No
5	153	6.880	No
6	73	6.591	No
7	119	4.606	No
8	143	4.603	No
9	159	4.007	No
10	122	3.413	Yes
11	113	2.770	No
12	74	2.471	No
13	116	2.378	Yes
14	111	1.613	Yes
15	126	1.426	Yes
16	124	1.338	Yes
17	125	1.013	Yes
18	146	0.987	No
19	150	0.968	No
20	117	0.960	Yes
21	76	0.938	No
22	75	0.922	No
23	1	0.755	No
24	154	0.692	No
25	90	0.480	No
26	157	0.435	No
27	2	0.422	No
28	121	0.407	Yes
29	65	0.402	No
30	118	0.399	Yes
31	58	0.289	No
32	112	0.265	Yes
33	13	0.238	No
34	131	0.183	No
35	134	0.151	No

Table I.22: The five highest ranked benign processes for banking malware using algorithm 3. Also showing the clean datasets in which the executable is present.

unique ids	dis- tance sum	ma- li- cious	dfs			
4	32.514	No	malware_rat_session1 malware_zeus_session2	malware_rat_session2 zz_win8_1604_avond zz_win8_1804	malware_zeus_session1 zz_win8_1604	zz_win8_1704
135	24.814	No	malware_rat_session1	malware_rat_session2 malware_zeus_session2	malware_zeus_session1	
27	7.616	No	malware_rat_session2	zz_win8_1604_avond zz_win8_1804	zz_win8_1604	zz_win8_1704
153	6.880	No	malware_rat_session1	malware_rat_session2 malware_zeus_session2	malware_zeus_session1	
73	6.591	No	malware_rat_session1	malware_rat_session2 malware_zeus_session2	malware_zeus_session1	

Table I.23: Ranking of the processes based on the distance for rat session 1 using algorithm 3. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.

	unique ids	distance sum	malicious
1	4	24.599	No
2	109	12.328	No
3	95	10.132	No
4	204	10.043	No
5	111	8.817	No
6	192	5.445	No
7	167	5.266	Yes
8	188	4.051	No
9	159	3.804	No
10	124	3.709	No
11	43	2.653	No
12	56	1.429	No
13	208	1.386	No
14	122	1.315	No
15	123	1.266	No
16	57	1.130	No
17	203	1.126	No
18	102	1.095	No
19	191	0.927	No
20	187	0.896	No
21	201	0.830	No
22	222	0.789	No
23	1	0.720	No
24	120	0.662	No
25	27	0.656	No
26	72	0.590	No
27	115	0.494	No
28	86	0.394	No
29	26	0.361	No
30	107	0.328	No
31	186	0.272	Yes
32	171	0.267	Yes
33	173	0.267	Yes
34	175	0.267	Yes
35	177	0.267	Yes
36	179	0.267	Yes
37	181	0.267	Yes
38	185	0.267	Yes
39	183	0.264	Yes
40	97	0.230	No

Table I.24: The five highest ranked benign processes for rat session 1 using algorithm 3. Also showing the clean datasets in which the executable is present.

unique ids	dis- tance sum	ma- li- cious	dfs			
4	24.599	No	malware_bank	malware_rat_session2	malware_zeus_session1	
			malware_zeus_session2	zz_win8_1604_avond	zz_win8_1604	zz_win8_1704
				zz_win8_1804		
109	12.328	No	malware_zeus_session1	malware_zeus_session2	zz_win8_1604_avond	
				zz_win8_1604	zz_win8_1704	zz_win8_1804
95	10.132	No	malware_bank	malware_rat_session2	malware_zeus_session1	
				malware_zeus_session2		
204	10.043	No	malware_bank	malware_rat_session2	malware_zeus_session1	
				malware_zeus_session2		
111	8.817	No	malware_zeus_session1	malware_zeus_session2	zz_win8_1604_avond	
				zz_win8_1604	zz_win8_1704	zz_win8_1804

Table I.25: Ranking of the processes based on the distance for rat session 2 using algorithm 3. The first dark line shows the q95 border. The second line is the q90, third q85, fourth is the mean, the fifth q80 and the bottom line is q75.

	unique ids	distance sum	malicious
1	4	27.284	No
2	111	11.374	No
3	191	4.836	No
4	161	4.538	No
5	206	4.142	No
6	194	3.791	No
7	189	3.699	No
8	193	3.660	No
9	24	3.468	No
10	95	3.126	No
11	207	2.679	No
12	192	2.438	No
13	34	2.156	No
14	141	1.750	No
15	51	1.640	No
16	202	1.639	No
17	190	1.632	No
18	187	1.488	No
19	97	1.361	No
20	171	1.361	No
21	96	1.360	No
22	164	1.162	No
23	53	1.048	No
24	65	0.941	No
25	52	0.939	No
26	160	0.927	No
27	2	0.802	No
28	101	0.795	No
29	1	0.740	No
30	124	0.700	No
31	165	0.662	No
32	125	0.584	No
33	188	0.533	No
34	69	0.485	No
35	91	0.379	No
36	94	0.348	No
37	116	0.342	No
38	114	0.337	No
39	60	0.313	Yes
40	20	0.287	No
41	84	0.253	Yes
42	80	0.252	Yes
43	82	0.252	Yes
44	72	0.252	Yes
45	74	0.252	Yes
46	85	0.252	Yes
47	76	0.218	Yes

Table I.26: The five highest ranked benign processes for rat session 2 using algorithm 3. Also showing the clean datasets in which the executable is present.

unique ids	dis- tance sum	ma- li- cious	dfs		
4	27.284	No	malware_bank malware_zeus_session2	malware_rat_session1 zz_win8_1604_avond zz_win8_1804	malware_zeus_session1 zz_win8_1604 zz_win8_1704
111	11.374	No	malware_bank	malware_rat_session1 malware_zeus_session2	malware_zeus_session1
191	4.836	No		zz_win8_1604	zz_win8_1704
161	4.538	No			no
206	4.142	No			no

Table I.27: Ranking of the processes based on the distance for zeus session 1 using algorithm 3. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.

	unique ids	distance sum	malicious
1	4	32.503	No
2	111	15.913	No
3	113	15.806	No
4	80	6.663	No
5	107	6.191	No
6	147	4.524	No
7	81	3.265	No
8	42	2.784	No
9	38	2.616	No
10	82	2.035	No
11	41	1.690	No
12	132	1.507	No
13	141	1.484	No
14	83	1.387	No
15	140	1.375	No
16	134	1.341	No
17	91	1.121	No
18	121	1.023	No
19	43	1.002	No
20	127	0.980	No
21	150	0.841	No
22	112	0.831	No
23	84	0.799	No
24	39	0.670	No
25	1	0.669	No
26	2	0.553	No
27	117	0.506	No
28	75	0.405	No
29	152	0.389	No
30	130	0.381	No
31	97	0.310	No
32	73	0.301	No
33	135	0.291	No
34	118	0.273	No
35	12	0.263	No

Table I.28: The five highest ranked benign processes for zeus session 1 using algorithm 3. Also showing the clean datasets in which the executable is present.

unique ids	dis- tance sum	ma- li- cious	dfs			
4	32.503	No	malware.bank	malware_rat_session1	malware_rat_session2	
			malware_zeus_session2	zz_win8_1604_avond	zz_win8_1604	zz_win8_1704
				zz_win8_1804		
111	15.913	No	malware_rat_session1	malware_zeus_session2	zz_win8_1604_avond	
				zz_win8_1604	zz_win8_1704	zz_win8_1804
113	15.806	No	malware_rat_session1	malware_zeus_session2	zz_win8_1604_avond	
				zz_win8_1604	zz_win8_1704	zz_win8_1804
80	6.663	No	malware.bank	malware_rat_session1	malware_rat_session2	
				malware_zeus_session2		
107	6.191	No	malware.bank	malware_rat_session1	malware_rat_session2	
				malware_zeus_session2		

Table I.29: Ranking of the processes based on the distance for zeus session 2 using algorithm 3. The first dark line shows the q95 border. All processes above this line are marked as malicious. The second line is the q90, third the mean, fourth q85, the fifth q80 and the bottom line is q75.

	unique ids	distance sum	malicious
1	4	32.503	No
2	112	15.942	No
3	114	15.749	No
4	79	6.511	No
5	130	5.498	No
6	116	5.462	No
7	117	3.763	No
8	82	3.637	No
9	119	3.250	No
10	163	3.092	No
11	154	2.985	No
12	81	2.970	No
13	123	2.889	No
14	80	2.786	No
15	36	2.698	No
16	118	2.670	No
17	40	2.652	No
18	120	2.342	No
19	160	2.091	No
20	129	2.019	No
21	140	1.845	No
22	115	1.619	No
23	164	1.196	No
24	59	0.957	No
25	139	0.930	No
26	121	0.858	No
27	37	0.781	No
28	1	0.683	No
29	137	0.679	No
30	127	0.676	No
31	151	0.550	No
32	167	0.548	No
33	92	0.533	No
34	26	0.518	No
35	176	0.509	No
36	85	0.453	No
37	187	0.432	No
38	148	0.409	No

Table I.30: The five highest ranked benign processes for zeus session 2 using algorithm 3. Also showing the clean datasets in which the executable is present.

unique ids	dis- tance sum	ma- li- cious	dfs			
4	32.503	No	malware.bank malware_zeus_session1	malware_rat_session1 zz_win8_1604_avond	malware_rat_session2 zz_win8_1604	zz_win8_1704 zz_win8_1804
112	15.942	No	malware_rat_session1	malware_zeus_session1 zz_win8_1604	zz_win8_1604_avond zz_win8_1704	zz_win8_1804
114	15.749	No	malware_rat_session1	malware_zeus_session1 zz_win8_1604	zz_win8_1604_avond zz_win8_1704	zz_win8_1804
79	6.511	No	malware.bank	malware_rat_session1 malware_zeus_session1	malware_rat_session2	
130	5.498	No	malware.bank malware_zeus_session1	malware_rat_session1 zz_win8_1604_avond	malware_rat_session2 zz_win8_1604	zz_win8_1704

J

Algorithm 3: Process trees

Figure J.1: The process tree of the banking malware dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.

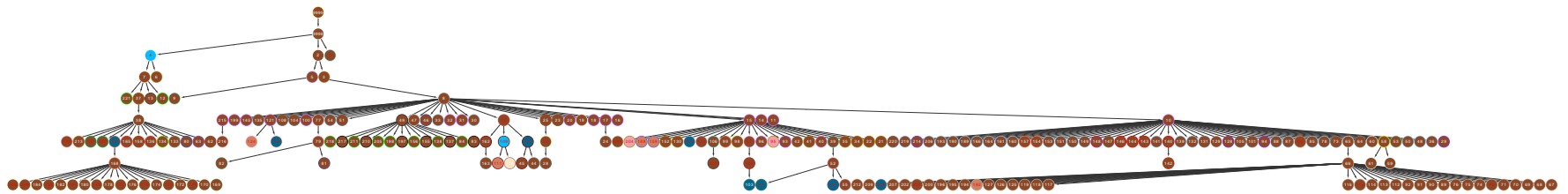


Figure J.2: The process tree of the rat session 1 malware dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.

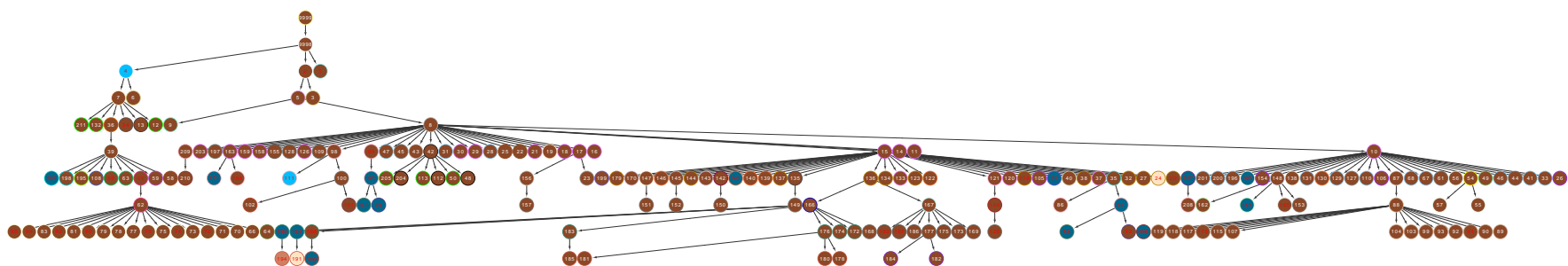


Figure J.3: The process tree of the rat session 2 malware dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.

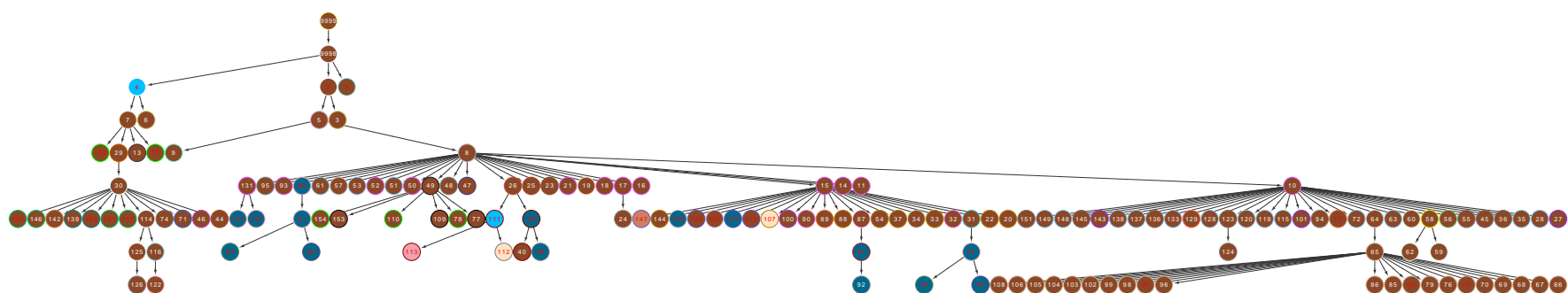


Figure J.4: The process tree of the zeus session 1 malware dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.

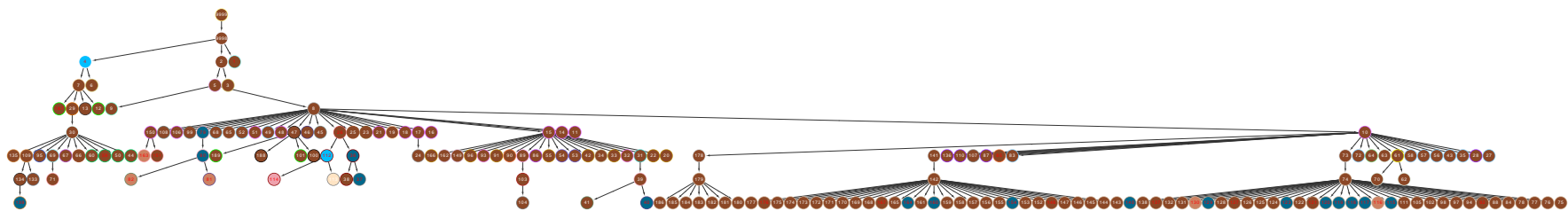


Figure J.5: The process tree of the zeus session 1 malware dataset set showing the malicious marked process ids, using the 75% quantile as threshold, with a red label.