Countering Ransomware Using Anomaly Detection of Endpoint Events

V.A. (Vincent) van der Eijk

Computer Engineering
The Hague University of Applied Sciences

Graduation report

December 21, 2017

Abstract

Ransomware gained a lot of media attention in 2017 because of the many large-scale attacks that occurred which managed to shut down multiple companies, or even hospitals, and resulted in significant costs [4, 18]. Because this specific type of malware appears to be a lucrative business for criminals, it is of even greater importance for cybersecurity vendors to provide the required measures to prevent ransomware infections by implementing an anti-virus (AV) solution which is capable of mitigating a potential threat. However, the AV solutions which are currently implemented are not build to act upon unknown malware. The Next-Generation Anti-virus (NGAV) solution should therefore be able to dynamically analyze system events to classify specific actions as malware. Although various cybersecurity vendors claim to have a solution, in practice a flawless NGAV is yet nowhere to be found. This research focuses on the implementation of a behavioral and probability based model to enhance the current AV solution of Fox-IT with the capability to detect unknown malware processes. The main research question of this research is:

"Is it possible to accurately classify system processes on a Windows endpoint as ransomware based on its features?"

A prototype is presented to extract potentially malicious features from Windows processes in order to classify a process as benign or malicious. These features are recommended by various renowned security specialists [2, 3, 14]. A machine learning classification algorithm is trained with testing data of both malicious and benign processes in order to make actual predictions for the testing data set. This is a novel approach where both security intelligence is combined with machine learning whilst not performing an in-depth analysis of system calls or machine code.

The data samples used as input for classification were obtained in collaboration with the Threat Intelligence (TI) department of Fox-IT to provide a sample set representing real-world ransomware for testing. The classification algorithm provided exceedingly well results, which is partially due to the fact that the chosen features did show very strong characteristics of ransomware. The algorithm which was used in the final prototype even managed to classify every single malware process correctly, directly at the root of the processes. With a True Positive Rate (TPR), or recall, of 100%, the algorithm did alert on various benign processes. These processes all contained features and behavior which indeed appeared malicious such as crash reporters which hooked in on a failing process, or installers containing unsigned executables. However, the amount of false positives remained relatively low overall and they could be easily prevented by whitelisting this small set of specific processes.

Vincent (V.A.) van der Eijk, "Countering Ransomware Using Anomaly Detection of Endpoint Events". Graduation report to obtain the Bachelor's degree for the education of Computer Engineering, The Hague University of Applied Sciences, 2017.

Keywords: anomaly detection, classification, Fox-IT, malware, next-generation anti-virus, ransomware

Preface

This research has been commissioned by Fox-IT, a major cybersecurity firm in Delft, The Netherlands. The research is a part of the graduation internship in order to obtain the Bachelor's degree of Computer Engineering at The Hague University of Applied Sciences. I would like to thank my supervisors Hans de Vreught and Pieter Burghouwt for their valuable feedback during this research.

This research would not have been the same without the expertise from several employees of Fox-IT. Their knowledge about malware and system processes provided useful insights for this research. Their help to provide a testing environment and malware samples was of great value to achieve the results presented in this research.

Thank you.

Vincent van der Eijk
Delft, December 2017

Contents

List of Figures

List of Tables

1 Introduction

In the past year, a major surge of ransomware has taken place. This specific type of malware, meant to extort its victims or to wreck systems completely, has had a big impact on society [4]. Ransomware has received a lot of media attention since the large-scale attack of the WannaCry variant in May 2017, when it managed to encrypt over 160,000 computers in only a couple of days. The WannaCry ransomware was particularly successful because it managed to spread quickly through internal networks. The months following, new variants of the WannaCry ransomware were released, which exploited the same vulnerabilities. Some of these did even greater harm than the earlier WannaCry attack [4]. Ransomware infections are therefore not expected to diminish in the future. Instead, according to the 2017 annual security report by Cisco, ransomware is growing at a yearly rate of 350% [6]. The total cost of ransomware is predicted to be $5 billion for 2017 [7].

Due to the increase of this specific type of malware with disastrous consequences, it is crucial to have a solution in place to detect and prevent new ransomware outbreaks. Traditional anti-virus (AV) solutions rely on signature-based detection. This means that the hash, or the signature, of executables is compared against a database of known hashes of malicious files, and therefore flagged as malicious. This technique works very well for known malware, but has the downside that it fails to detect new malware variants which have a different hash. The slightest change to the binary of the malware means that it can stay undetected by AV. This traditional approach of AV is found on almost every endpoint, and provided a good solution for malware attacks in the past. As malware evolved, the traditional AV solutions remained stagnant and therefore a new and innovative approach is required.

To avoid the challenges of signature based detection that traditional AV has to cope with, the next-generation anti-virus (NGAV) has been introduced. This type of endpoint protection takes a fundamentally different approach as it examines system events from a system-centric point of view [9]. By doing so, a behavioral pattern is created in order to classify an event as malicious or benign. This results in a more dynamic AV solution which is also able to flag potentially malicious files that haven't been seen before, independent of the signature hash of the file. The NGAV is a relatively new concept and still has to mature. There are currently no known vendors of NGAV who offer a solution which combines both security intelligence and logic to provide a state-of-the-art security mechanism.

This approach of behavioral detection is often related to machine learning. When using machine learning as a method for malware classification, processes can be evaluated using a predictive algorithm to determine the probability of a process being malicious or benign. Creating a model that is accurate for all cases is a challenging task. The recent BadRabbit ransomware of October 2017 was classified as legitimate by Microsoft because the malware probability score was only a little below the threshold required to block its execution [9]. After evaluation, the ransomware received a higher classification and was blocked by the Windows operating system eventually. In the meantime eight other endpoints got infected.

Although a machine learning approach is not 100% safe, as the BadRabbit example shows, it is able to learn quickly when deployed properly so actual threats can still be blocked faster than with traditional AV. The only difference between 'known' and 'unknown' threats is time, and it is the duty of AV providers to reduce this as much as possible [10].

1.1 Fox-IT CTM Endpoint Module

The CTM Endpoint Module is the AV solution provided by Fox-IT. The endpoint module provides system hardening and protection which still relies for a large part upon signature-based detection for ransomware and static rule-based mechanisms. Although the Endpoint Module provides a good solution for endpoint protection, it faces the same challenges as other signature-based AV solutions.

1.2 Research Outline

This research, commissioned by Fox-IT, is focused on the development of a new concept in cybersecurity event monitoring to improve the detection capabilities of the CTM Endpoint Module. Additional background information of Fox-IT is provided in Chapter 2. The description of the research assignment and the goal of the research is given in Chapter 3. Chapter 4 states the approach of the research and provides a more detailed overview of the project including the separate tasks and activities during the length of the research.

Before specifying the requirements of the research a literature review is performed to provide additional information about common ransomware characteristics and different classification techniques. Based on these results, multiple exploratory prototypes are developed to establish the exact requirements of the research in accordance with the product owner in Chapter 5.

The development of the final prototype is described in Chapter 6 which also included the designs of the operation of the prototype. An agile approach was used during the development phase which is described in three separate sprints. The tests that have been performed for the software are included in a separate document, the Test Report, which is included in Appendix E.

The results of the final prototype are included in Chapter 7, and leads to a conclusion which is drawn in Chapter 8 which also provides recommendations for future work.

2 Company Background

Fox-IT is a renowned provider of cybersecurity solutions. The company has customers worldwide, but is well-known in the Netherlands because of the services they provide for the Dutch government. Customers of Fox-IT are known for their vital infrastructures, such as the Dutch government. The motto of Fox-IT is: "For a more secure society" [11].

2.1 Goals & Activities

The mission of Fox-IT has been formulated as: "To derive satisfaction from helping to create a more secure society with the help of our technical and innovative solutions".

The Development CTMp (Cyber Threat Management platform) department is responsible for the development of a platform to be deployed in the Security Operations Center (SOC) of Fox-IT. The network infrastructure of customers of Fox-IT is continuously monitored by the SOC for any security incidents.

CTMp has an Endpoint department, which is responsible for the development of the CTM Endpoint Module. The alerts generated by the CTM Endpoint Module enhance the other monitoring services that Fox-IT has to offer. More relevant reporting of security incidents will result in a faster response to serious threats and therefore add value to the Managed Security Services of Fox-IT.

2.2 Organizational Structure

In November 2015 Fox-IT has been acquired by NCC group. Because of this acquisition, Fox-IT is now part of a global team of more than 2,000 employees spread across 3 continents. This acquisition has had little impact on the day-to-day activities of Fox-IT employees.

The technology portfolio of Fox-IT includes advanced tools to secure infrastructures and to operate advanced security processes [12]. The CTMp department is included in the technology portfolio of Fox-IT. The intelligence which is incorporated in the different modules used by CTMp is obtained by the various departments of Fox-IT which are in the field every day, seeing new attacks and context of their findings. In particular, the Threat Intelligence (TI) department of Fox-IT is responsible for malware analysis, which also includes ransomware samples.

The departments of Fox-IT which are relevant for this research are shown in Figure 1 on the following page. The Endpoint Module department of CTMp is located under the technology portfolio of Fox-IT.

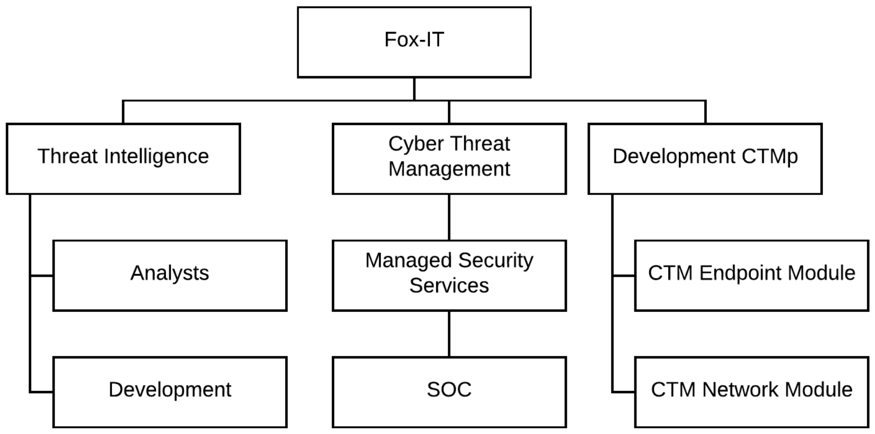


Fig. 1: Fox-IT organizational diagram

3 Research Description

This research has been introduced because there is not sufficient knowledge to detect and prevent an advanced ransomware attack on Windows endpoint devices.

The primary goal of this research is to propose a solution in the form of a prototype that is able to classify Windows processes that are retrieved from Windows endpoints into potentially malicious and benign events.

The main research question formulated to reach the goal of the research is therefore:

Is it possible to accurately classify system processes on a Windows endpoint as ransomware based on its features?

In order to classify system processes, a selection of features has to be defined to serve as input for the classification algorithm. The selection of these features and the type of algorithm used for classification is determined by the following subquestions:

1. *How are the features of system processes on a Windows endpoint being logged?* In order to make predictions about the nature of process logs these events have to be collected for further analysis.
2. *What are the strongest features of ransomware noticeable on an endpoint?* When the large amount of system events is reduced to a minimum it is possible to make fast predictions with little overhead. This requires a selection of only the most relevant features of processes that are related to ransomware.
3. *Which type of algorithm is most suitable for classification?* Different types of predictive classification algorithms may provide different results. This includes the convergence rate for a self-learning classification algorithm which might make the classifier more suitable to predict new situations.
4. *What is a reasonable accuracy for the classification of system processes?* The answer to this question is of great importance in order to verify that the classification algorithm performs according to the stakeholders' needs. This also includes determining which metrics are relevant for the calculation of the performance of the classifier.

In order to determine whether the results of the classification algorithm are sufficient, the desired accuracy of the algorithm has to be defined beforehand. It is recommended to implement a machine learning algorithm for the classification of processes in the prototype.

4 Initiation

This chapter provides a global overview of the research and several considerations which have been taken into account at the start of the research.

Originally, the goal of the research has been formulated to detect anomalous patterns in the Windows kernel in real-time to determine if an endpoint is being infected with ransomware. However, quickly after the start of the research the description has been altered in a way to focus more on the feasibility of anomaly detection itself. Both the real-time constraint and the direct interface to the Windows kernel are not applicable anymore in the newly formulated goal, as mentioned in Chapter 3. Instead, event logs of endpoint devices will be used for static classification. If the results of this research show that anomaly detection and classification of system events is feasible, this technique might be incorporated in future releases of the CTM Endpoint Module.

4.1 Approach

To reach the goal of the research as stated in Chapter 3, ransomware samples have to be collected and analyzed to retrieve process information. Additionally, a dataset consisting of benign processes has to be generated to serve as a baseline for trusted processes. A set of the most specific features extracted from these processes will be used as the input for the classification algorithm. The feature set which will be used for classification is determined by performing literature research about malware characteristics, which provides an answer to the research question:

- *'What are the strongest features of ransomware noticeable on an endpoint?'*

To determine which type of classification algorithm is most suitable for the dataset different exploratory prototypes will be developed and tested for their accuracy. This will also be the main focus of the definition phase. This approach of exploratory structural prototyping allows to quickly receive feedback in a short time span to determine and verify the stakeholders' needs and to refine the actual software specification, if necessary. The prototyping phase will provide answers to the research questions:

- *'Which type of algorithm is most suitable for classification?'*
- *'What is a reasonable accuracy for the classification algorithm?'*

To determine whether a classification algorithm performs according to the expectations it will be tested for a variety of metrics including accuracy, speed and convergence of a machine learning algorithm.

The metric for the accuracy will be used to determine whether the chosen classification algorithm performs according to the expectations. During this phase the requirements of the prototype will be validated in accordance with the product owner of the CTM Endpoint Module. When the actual software specification are in place, a global design of the software architecture will be developed that will act as a guidance during the development phase.

The dataset containing process information is required prior to the development of the classification algorithm for testing of the classification algorithm. A reliable dataset with accurate features of both malicious and benign data is required to train and test the algorithm. The Threat Intelligence (TI) department of Fox-IT provides malware samples which will be run on a virtualized environment to collect information of system processes. A virtual environment using its own internal network is used in order to reduce the risk of the malware to spread. Additionally, this ensures that known-good snapshots of the environment can be restored quickly after a malware infection to repeat experiments. The process information of benign processes is obtained by simulating multiple uninfected Windows endpoints in the virtualized environment. Prior to the collection process events, insight must be created in the logging of system processes and how these events could be used as input for the classification algorithm, which answers the final question:

- *'How are the features of system processes on a Windows endpoint being logged?'*.

With the proper datasets obtained, the classification algorithm can be tested and adjusted where required during the development phase.

The last phase of the assignment, the development phase, consists of three sprints of three weeks each. This time span of three weeks creates enough time to be able to develop significant increments, while still having sufficient iterations to adjust if necessary. The epics written during the definition phase will be used as an initial starting point. At the end of the first two sprints new user-stories will be written for the following sprint, based on remaining system requirements. At the end of this phase the development report and the test report are completed.

By performing thorough tests of the classification algorithm it is possible to ultimately draw valid conclusions from the results. These results have to provide an answer whether it is possible to accurately classify system processes on a Windows endpoint as ransomware by monitoring process features, which is the answer to the primary research question.

4.2 Research outline

The global overview of each phase in this research and corresponding tasks are shown in Table 1. Every phase will provide its own deliverables. The development phase will provide both a development report and a test report. All deliverables are attached as an appendix. The final deliverables are the following:

- Plan of Action
- Definition Report
- Development Report
- Test Report

Chapter 5 provides background information about characteristics of ransomware and malware in general. This is the basis for the research to determine which features should be used for classification. The actual methods for classification are reviewed to determine a set of possibilities for the structural exploratory prototyping phase. Based on the results of the exploratory prototyping a classification algorithm is chosen to be implemented during the actual prototype development which is described in Chapter 6. The results of the prototype and an explanation for these results is provided in Chapter 7. Chapter 8 concludes this research with answers to the research questions and recommendations for future work.

Tab. 1: Project planning

Phase	**Task**
Initiation (1 week)	Assignment description
	Analyzing project risks
	Create planning
Definition (7 weeks)	Read about malware characteristics
	Read about machine learning possibilities
	Exploratory structural prototyping
	Determining the stakeholders' needs
	Write epics
	Software architecture development
Development (3 x 3 weeks)	Software development
	Software testing

4.3 Risks

Because the research is exploratory by nature, there are no components of the CTM Endpoint Module which can be negatively impacted based on the outcome of the research. The introduced risks are therefore only related to the execution of the research and its corresponding prototype development, but not to the CTM Endpoint Module itself.

4.3.1 Insufficient access to internal resources

Due to only a moderate screening level as an intern at Fox-IT, there is no full access to internal resources. If specific restricted resources are required, this could become a problem. This risk is described in Table 2. It is also relevant for the access to malware samples, as there might be no authorization to run real malware in an environment set up for this research.

Especially for the training data set it is required to have access to internal research about malware samples and existing log files. However, if access is restricted there are various technical analyses made available publicly which could be used instead.

Tab. 2: Insufficient access to internal resources

Description	Insufficient access to internal resources
Probability	Medium
Impact	Research cannot be performed or validated without the required test data, or tools to generate this test data
Risk reduction	Discuss the obtaining of internal resources in a timely manner
Impact reduction	Obtain similar resources from a public source
Contingency Plan	Implement a workaround for the specific situation which does not require this restricted resource

4.3.2 Ransomware encrypts endpoint

Ransomware samples have to be monitored for their activity in order to make predictions based on their behavior. If the ransomware manages to break out of its contained network environment it will encrypt the (log) files on the system, resulting in a loss of work. The setup of the network is described in more detail in Appendix G.1 on page 77.

Tab. 3: Ransomware encrypts endpoint

Description	Ransomware encrypts endpoints (on the network)
Probability	Medium
Impact	Log files or other work might get encrypted and therefore lost
Risk reduction	The ransomware samples are executed in a virtualized internal network environment disconnected from an outside network. All machines except the targeted machine are provided with the latest updates and security patches
Impact reduction	Regular snapshots of known-good points in time are made to create a system restore. Only log files are stored on the machine
Contingency Plan	Restore the most recent snapshot

5 Definition

This chapter, which describes the Definition phase of the research provides insights in the execution of ransomware and different classification techniques such as supervised and unsupervised machine learning which will be used for the final prototype to classify system events.

5.1 Literature Research

The literature research has been divided into four parts. To start with, a brief introduction is provided about endpoint security and its known flaws in Section 5.1.1. Section 5.1.2 provides detailed background information about ransomware and its common characteristics. These characteristics are expressed as features that will be used for classification. Section 5.1.3 provides information and possible approaches for classification. This includes machine learning in particular, which is the desired approach for the problem of anomaly detection. The metrics to calculate the precision of the classification algorithm are discussed which are required to provide reliable results at the end of the research. This literature research is concluded in Section 5.1.4 with previous work regarding this subject to provide background information about the current situation of anomaly detection of endpoint devices.

5.1.1 Introduction to Endpoint Security

Traditional consumer anti-virus (AV) solutions which provide endpoint security are not capable of preventing advanced threats. However, neither do enterprise-grade intrusion detection systems. Currently, most malware detection approaches use common signature based detection algorithms. These algorithms scan the file system for hashes (signatures) of known malicious files, or scan files for known malicious byte patterns. This approach works pretty well, although this is only the case if the signatures are updated regularly. In the event that a malicious file is altered, its signature changes and the file is not detected anymore. Because the traditional AV approach can only detect known signatures, it is impossible to detect zero-day exploits which can be the initial attack vector of a ransomware attack.

While this technique of signature-based malware detection is used for most AV solutions for personal use, it is not advanced enough to provide proper protection for high-value targets with a vital infrastructure such as the customers of Fox-IT.

5.1.2 Ransomware Analysis

The Threat Intelligence (TI) department of Fox-IT has provided several malware samples to be used for generating test data for this research. The following paragraphs discuss the malware characteristics found by literature research and relates these characteristics to the features obtained from the samples provided by Fox-IT.

Windows processes can be used to model the behavior of an endpoint. This model, or process tree creates an overview of the relationship between parent and child processes instead of just an instance of a single process. With the model of a process tree in place, anomalies can be detected in its branches based on the features of the processes.

Ransomware Features Ransomware, and malware in general, is known to show certain anomalies from regular system processes. For regular users, processes are created in a predictable manner [3]. Most processes are started by *explorer.exe* and not by *cmd.exe*. Furthermore certain operating system services are known not to create new processes, which makes it suspicious and anomalous behavior if they do. This is a major indicator which should be used when evaluating process creation events.

Russinovich, a renowned security researcher at Microsoft, states that ransomware has several characteristics that can be observed during execution [2]. Ransomware is typically not a type of malware that is designed to stay on a system undetected as it is meant to extort its victim. The best way to detect ransomware would be to monitor processes on the system with a tool such as Process Monitor [13]. Russinovich states that processes that are worth investigating are processes that contain features such as the lack of an icon, have no description, are unsigned, or live in the Windows directory or user profile. These specific features can be explained due to the fact that malware is developed with the purpose to extort its victim and not to be a high-quality piece of software. Another recommendation given by the security researcher from Microsoft is to be especially wary of items residing the the Windows directory or the AppData directory of the user profile.

According to Cisco, one of the most important characteristics of ransomware is the deletion of system shadow copies, which prevent a system backup restore [14]. In 2014, the CryptoWall ransomware was the first ransomware family which incorporated this technique after it had been disclosed that a system could be restored with a shadow copy [15]. The process to delete a shadow copy from the file system should therefore also be a major indicator of ransomware. Because administrative privileges are required to perform this action, an elevated process should be hijacked first.

The recent WannaCry ransomware from May 2017 did show traces of all features mentioned in this section which eventually led to the successful encryption of over 160,000 computers worldwide [16]. Using a publicly available exploit the process *lsass.exe*, which has administrative privileges, was hijacked to eventually start the process *tasksche.exe* which launches a command prompt to delete shadow copies and create persistence at boot to show the ransom note by adding a key to the registry. Under normal circumstances, the process *lsass.exe* should never start *tasksche.exe*.

To summarize, the features of system processes that will be used for classification are the following:

- Residing directory
- Unsigned executables
- Deletion of shadow copies
- Lack of process meta-data
- Unknown process relationship
- Scripting files

The first subquestion of this research can therefore already be answered, as this is the list of features that will be used for classification.

Ransomware Samples The following ransomware samples have been provided by Fox-IT. These samples have been harvested from infected machines. The malware has not been altered and should therefore be representative for this research.

The ransomware samples have been run on a network of virtualized endpoints. Both the CTM Endpoint Module and the tool ProcessMonitor collected logs of system process events during the execution of the ransomware samples. Both monitoring tools were required to create a complete overview of the system, by merging the features of both logs. The process and the setup used for collecting process event logs is described in more detail in Appendix G.

- Locky: Ransomware family released in 2016. The ransomware became active again in the summer of 2017 when it was spread through multiple SPAM emails containing an invoice that required payment. The attached document was a Microsoft Word document that contained malicious macros [17].
- Bitpaymer: A ransomware variant targeted at hospitals. The ransomware unpacks several malicious unsigned executables into the user profile folder [18].
- GlobeImposter: This ransomware variant is obtained through a drive-by executed by a malicious Javascript file. This ransomware deletes shadowcopies from the system [19].
- Jaff: A ransomware variant that has been primarily spread through SPAM emails and spearphishing attacks [20].
- WannaCry: The infamous ransomware which gained a lot of publicity because it managed to disrupt infrastructure on a global level [21].

In addition to the ransomware samples mentioned above, two additional malware samples have been provided. Note that these samples are not ransomware and are only being tested in order to see if the classification algorithm is also able to detect other kinds of malware instead of only ransomware which it will initially be trained for.

- Geodo: Botnet used as a loader for more malware and SPAM [22].
- Trickbot: Banking malware which injects malicious code into the browser when specific websites are accessed [23].
- Kronos: Banking malware which injects malicious code into the browser when specific websites are accessed [24].

5.1.3 Machine Learning

The current implementation of rule-based classification has several shortcomings by nature which can be bypassed relatively easily. Because of these shortcomings, a new approach is desired. At the start of this research, the suggestion was made to implement a classification method using behavioral patterns and anomaly detection, which leads towards a solution implementing machine learning.

A self-learning classification algorithm is a solution which is likely to solve the problems introduced by static signature-based detection as described in Section 5.1.1. This type of classification algorithm is able to evaluate events and to classify them either as malicious or benign. The main strength of self-learning classification is that it is able to classify events which have not been hard-coded into the detection which is the case for signature-based detection. This means that new variants of ransomware families, or even completely new ransomware families, could still be detected dynamically based on their features instead of a static hash of the executable. With the self-learning capabilities of the classification algorithm it is even able to make even more accurate predictions after it has seen a specific sample multiple times. The learning curve, or the convergence rate, determines how well the classification algorithm is able to adjust itself. This way, false positives can be used as input for the algorithm to reduce the probability of occurrence in the future.

Four different approaches for a machine learning algorithm are chosen for the exploratory prototyping. These different algorithms will be tested and evaluated based on their performance. The algorithm which performs the best will be chosen for the implementation of the actual prototype. The metrics that will be tested on are the following:

- Accuracy - The accuracy of the classifier based on a manually crafted data set for training and testing
- Noise - How well the algorithm is able to handle irregularities in the dataset
- Convergence - How well the classifier is able to learn from new data samples
- Execution time - How fast the classifier is able to make a prediction
- Complexity - The complexity of the algorithm behind the classification

Classification Algorithms A classification algorithm will be implemented to test which type of algorithm is most suitable for the classification of system processes. A variety of different approaches for machine learning implementation has been chosen in order to test the various different characteristics of these algorithms. A brief description of the algorithms which will be incorporated in the prototyping phase is presented below:

- Counting

This is the most trivial approach for the classification problem. Given the set of features, the amount of features that indicate ransomware can be counted and if a certain threshold is reached the process is being classified as malicious. Although this solution is very easy to implement it is expected to generate a lot of false positives and false negatives. Another downside is that using this approach shows many similarities to a rule-based AV solution, which makes it a less feasible solution. This is the only approach not implementing a machine learning algorithm.

- Naive Bayes

The Naive Bayes algorithm is a supervised classification algorithm that assumes independence among data features. This algorithm is known to perform relatively well in general while being easy to implement. It is a probabilistic algorithm that is trained by using already classified data. To train the classifier, both benign and malicious data has to be collected for training [25].

- K-Modes Clustering

The algorithm will generate clusters based on the features of the input data. When using a unsupervised learning algorithm data does not have to be labeled to train the classification algorithm. However, to test the accuracy of the algorithm labeled test data is still required to determine if the predicted clusters of the classification algorithm are correct. This unsupervised learning algorithm is the only one suitable for discrete values which are used for the data set of the CTM Endpoint Module. Because it is different from the other supervised learning algorithms it is interesting to try it during the exploratory prototyping phase. Because not all data sets are suitable for clustering it is not always an effective approach [27, 26].

- Deep Neural Network

A Deep Neural Network is another approach to supervised machine learning and approximates a 'black-box' approach where the models are determined empirically, instead of theoretically [42]. The approach of a neural network makes it an interesting option to try during the prototyping phase.

The exploratory prototypes developed for the definition phase are described in more detail in Appendix C.5 on page 52.

Dimensionality Reduction Machine learning implementations often suffer from the *curse of dimensionality*, meaning the dataset contains too many features to make proper predictions. Datasets with too many features are often noisy and contain features which are not relevant. Because of this characteristic of datasets, the feature set has been reduced to only 6 features which are the most expressive, as discussed in Paragraph 5.1.2 on page 10.

Enhancing the dataset by removing redundant or irrelevant features will significantly improve the performance of the algorithm in the following ways:

- Better understanding of the data
- Improving prediction accuracy
- Faster predictions

The most expressive features of the dataset will be selected to optimize the classification. The features which have been mentioned in Section 5.1.2 on page 10 are the features that will remain after applying dimensionality reduction to the dataset.

After applying feature selection and extraction as described in Section C.3.2 on page 49, a 15-fold increase in performance could be obtained by discarding irrelevant features in the CTM Endpoint Module.

The remaining dataset will in turn be used to extract the 6 features which have been mentioned in Section 5.1.2. These features have been set by applying boolean logic to check for the presence of specific values which apply to the corresponding feature. This results in a 1-dimensional dataset consisting of only 6 binary features per process.

The reduced feature set will provide more understanding of the data and result in significantly faster predictions.

Classifier Accuracy Brownlee, the author of various books regarding machine learning, provided a writeup on the pitfalls of calculating the accuracy of classification algorithms and how this result can often be misleading [41]. The results are misleading because of the class imbalance in the dataset which is used for classification, which is also referred to as the *Accuracy Paradox*. Additional measures are required to properly evaluate the classification algorithm.

A measure which is commonly used which provides a good insight in the performance of the classification algorithm is the precision. This value is the number of True Positives divided by the number of True positives and False Positives. This results in a score which represents the overall ability of the classification algorithm to correctly predict True Positives. For the ransomware classification problem where the amount of malicious processes is significantly less compared to benign processes, it is desired to know the ability to correctly predict True Positive results instead of True Negatives.

To provide a clean and unambiguous solution to present the predictions made by the classification algorithm, a confusion matrix is used. This table shows a clear overview of the correctly and incorrectly predicted results which will be used to calculate the metrics to determine classifier performance. An example of a confusion matrix is provided in Table 4.

Tab. 4: Confusion matrix

Predicted \ Actual	**Positive**	**Negative**
Positive	True Positive (TP)	False Positive (FP)
Negative	False Negative (FN)	True Negative (TN)

The following can therefore be said about the last subquestion of this research regarding the accuracy of the classification algorithm:

"What is a reasonable accuracy for the implementation of the classification algorithm?"

The accuracy of the algorithm itself is not relevant as this does not say much about its predictive capabilities. Instead, the precision metric is the most important for the malware classification problem. With the formula to calculate the precision of the classification algorithm, its is also significantly easier to assign a specific value to the metric of the required performance. The total amount of True and False positives have to be tracked down in order to set a minimum requirement for the classification algorithm to achieve. The exact interpretation of this requirement is provided in Section 5.3 on page 20 in collaboration with the actors involved in the CTM Endpoint Module. The formula for the precision, also referred to as the Positive Predictive Value (PPV) is as follows:

$$PPV = \frac{TP}{TP + FP} \tag{1}$$

Additional metrics that provide insight in the performance of the algorithm are the False Postive Ratio (FPR) and the True Positive Ratio (TPR), which provide information about the amount of incorrectly classified negatives and correctly classified positives. The following formulas are used to calculate these metrics:

$$FPR = \frac{FP}{FP + TN} \tag{2}$$

$$TPR = \frac{TP}{TP + FN} \tag{3}$$

5.1.4 Related Work

At the time this research was initiated, researchers at Fox-IT were looking into the deployment of the X-Pack plug-in for Elasticsearch [34]. This plug-in supports several different features to enhance an Elasticsearch deployment, which is currently used in the CTM Endpoint Module, by including machine learning capabilities. The results are not impressive so far, as it only makes use of a simple probabilistic algorithm to determine feature importance and to generate a subset of events of interest. The problem with this approach is that, even if it succeeds to provide decent results, the false negatives will still not be detected because the plug-in is only used together with the data that was already present. Furthermore the plug-in comes with a significant price tag, which doesn't make it an attractive solution.

Malware classification by means of anomaly detection is not a new concept. There are plenty of research papers that write about the possibilities of classifying malware types to ease the process of detection. Although most of these researches, of which some even date back to 2008 [35], claim to have outstanding results, anomaly based classification implementations –which should be the holy grail of cybersecurity– are yet nowhere to be found. The results are often not as promising as they appear because of the *Accuracy Paradox* described in the previous section.

Malware classification is often performed by analyzing binary files in the sample data set on a system call level [35, 37, 36]. This approach seems to be reliable, but has many familiarities with traditional AV solutions using signature-based detection. The research mentioned in [36] uses a clustering algorithm to automatically group malware into families to ease detection and further research.

In the past years several researchers have been performed and published regarding the topic of anomaly detection by using machine learning. These researches often rely on complex neural networks for the classification of malware samples. The main focus of the research appears to be the implementation of a machine learning algorithm and does not take the threat intelligence and malware characteristics into account as a recent study from [1] shows. The research is focused on the early stage malware detection using a recurrent neural network. It is said claims to detect whether an executable is malicious of benign within the first 4 seconds of execution with 93% accuracy. The neural network used in this research used features of the actual machine running the malware. Features such as the memory usage, CPU usage and other generic hardware monitoring approaches were used for malware classification. As the CTM Endpoint Module of Fox-IT contains more detailed process information, it will be more likely to develop a successful classification algorithm for ransomware by using better indicators.

5.2 Prototyping

The various approaches that have been used in the previous work mentioned in Section 5.1.4 show interesting approaches and results. These approaches (a simple probabilistic algorithm, clustering and a neural network) will be developed as an exploratory prototype to discover which approach is most suitable for the classification of the current feature set provided by the CTM Endpoint Module.

The structural exploratory prototyping technique is used to verify the initial needs of the stakeholders involved in the development of the final prototype. Appendix B.4 on page 42 lists the candidate requirements which already have been established at the start of the research. This prototyping phase will take these requirements into account and will both validate these requirements and elaborate on them.

To validate the requirements, four different exploratory prototypes are developed in a short timespan to receive quick feedback from the stakeholders to determine which approach will best fit their needs. The chosen approaches are already introduced in Section 5.1.3 and make sure that a variety of prototypes is tested in order to explore the different approaches to tackle the problem of classification. The chosen approaches are using a Naive Bayes classification algorithm, clustering, a deep neural network, and the regular counting of features.

The exploratory prototyping is primarily used to get an idea of which kind of algorithm provides the best performance for the problem of ransomware classification. Because the prototyping has been performed early-on in the research, there was no verified data set available for testing purposes. To make sure the accuracy of the classification algorithms could still be tested, a dataset with similar a similar categorical data structure was manually crafted for testing purposes.

The metrics mentioned in Section C.3 on page 47 are used to test the performance of the prototype.

5.2.1 Naive Bayes

The first prototype is using the Naive Bayes classification algorithm. This type of algorithm is relatively easy to implement yet known to produce accurate results. Although it is an algorithm based on a relatively simple concept, it often has an exceedingly well performance compared to other more complex algorithms. Naive Bayes is a probabilistic algorithm that assigns each feature a probability based on its occurrence.

For data sets similar to the CTM Endpoint Module data set, the implemented Naive Bayes algorithm has an accuracy of 99.9%. Algorithm 1 contains the straightforward implementation of the Naive Bayes algorithm for a Bernoulli distribution [30].

```
import sklearn.naive_bayes.BernoulliNB
df = generate_dataset()

results = BernoulliNB().fit(df.train, df.train.classes).predict(df.test)
```

Algorithm 1: Naive Bayes Prototype

5.2.2 Deep Neural Network

The following exploratory prototype makes use of a Deep Neural Network (DNN), which is a more complex approach for classification. The DNN is implemented using the open-source software library TensorFlow, which is developed by Google [31].

The configuration for the neural network, the hidden layers which are built up by neurons, is configured in the same way as is done in the examples given by TensorFlow [32]. The approach shows promising results within a relatively short execution time. The prototype with the TensorFlow implementation reaches an accuracy of 99.6% up to 100%, depending on the configuration of the neural network. A better accuracy does cost more computational power and therefore requires a longer execution time.

Algorithm 2 contains a code snippet from the implementation of the DNN classification algorithm.

```
import tensorflow as tf
df = generate_dataset()

cls = tf.contrib.learn.DNNClassifier(
    feature_columns=df.columns,
    hidden_units=[10, 20, 10],
    n_classes=2)
result = cls.fit(input_fn=df.train, steps=500).evaluate(input_fn=df.test)
```

Algorithm 2: Deep Neural Network Prototype

5.2.3 K-Modes

The two previous exploratory prototypes mentioned in Sections 5.2.1 and 5.2.2 are supervised learning algorithms, which means that they require pre-classified datasets in order to train the model.

The k-modes algorithm is an unsupervised machine learning algorithm and does not require labeled data to train the classifier with. Based on the features of the data set the algorithm will create *n* clusters which new entries will be related to. Using the k-modes algorithm new data entries will be classified to the nearest cluster. As the dataset is difficult to classify manually because of the many different (unknown) processes, an unsupervised approach might be preferred over supervised classification.

However, in order to validate if the output of the classification algorithm is correct, the process data samples still need to be classified for testing. Another common problem of clustering is that not all data sets are suitable for clustering which might lead to wrong results.

The clustering algorithm reaches an accuracy of approximately 85% up to 90%. The code snippet of the prototype implementing the k-modes cluster is provided in Algorithm 3.

```
import kmodes
df = generate_dataset()

km = kmodes.KModes(n_clusters=2, n_init=10)
results = km.fit(df)
```

Algorithm 3: K-Modes Prototype

5.2.4 Counting

A different and more trivial approach which does not use machine learning would be to count the amount of features that are positive. Because the dataset contains binary data, where a '1' would indicate a feature is present and '0' means it does not, this approach is actually expected to perform overall pretty well. The threshold to classify a data sample as malicious is set to 3; which means half of the features in the dataset should be positive. This value turned out to provide the best results after testing.

The accuracy of this prototype provided similar results to the Naive Bayes classification algorithm provided in Section 5.2.1, although the accuracy of this exploratory prototype would drop drastically if the dataset contains a lot of noisy features.

The algorithm to count the features in the data set is provided in Algorithm 4.

```
df = generate_dataset()
results = [None] * len(df)

for index in range(len(df)):
        results[i] = 1 if sum(list(df[i]) > 3 else 0
```

Algorithm 4: Counting Prototype

Prototyping Results

All classification algorithms performed rather well. The k-modes clustering algorithm was the only algorithm that did not provide sufficient results to consider for further development based on its accuracy. Although the other algorithms (Naive Bayes, Deep Neural Network, and Counting) provided a good accuracy overall, the Naive Bayes machine learning algorithm is chosen for further development as it provides the best results overall. Table 5 shows the scores assigned to each metric.

The other metrics that have been taken into account, which are the noise, execution time, and complexity of the prototypes do not differ much and therefore are not decisive, except for the complexity of the Deep Neural Network implementation. This approach appears to be too complex for the prototype to be developed.

Tab. 5: Prototype features summary

Category	Naive Bayes	K-Modes Cluster	Deep Neural Network	Counting
Accuracy	+ +	−	+ +	+
Noise	−	−	+	− −
Convergence	+	−	+ +	− −
Execution time	+	+	−	+
Complexity	+	+	− −	++

Based on these results, the Naive Bayes machine learning algorithm is chosen for further implementation of the final prototype as it has the best results overall. The implementation of the Naive Bayes machine learning algorithm is described in detail in Chapter 6

During the prototyping phase a new requirement has come forward. Currently, the CTM Endpoint Module can either allow or block system events. The prototype can, due to the use of a machine learning algorithm and the behavioral model of the system process tree, assign probabilities to sequential events. Each branch of the process tree will be monitored individually. If a branch reaches a certain threshold it can be flagged to monitor by a security analyst, and if a next threshold is surpassed the process can be blocked completely. A branch in the process tree might be used to accumulate a total malware score. Additionally, there should be no time constraint in place to monitor sequential events, as malware might make use of a the internal Windows Task Scheduler *tasksche.exe* to purposely delay its execution [33].

5.3 Research Requirements

The exploratory prototyping phase described in Section 5.2 was used to validate the requirements for the final prototype to be developed.

In order to address the impact of this research an overview of all the actors who are directly affected by the development of the CTM Endpoint Module has been made. This global overview is shown in Table 12 on page 51. The development team of the CTM Endpoint Module, the actors who are most directly involved, are not taken into account as they are not impacted by this research. This requirements analysis is required in order to determine whether the final prototype performs as desired. The actors who are involved in the CTM Endpoint Module have been identified as the following:

The **end user** has a computer with the CTM Endpoint Module installed on it. This actor is directly affected by the performance of the software. The end user does not want to be infected with ransomware and therefore the CTM Endpoint Module should block a malicious event before it does irreversible harm to the system of the end user. The end user does not want to receive unnecessary notifications of possible insecure system behavior, which means that false positive alerts should be reduced to a minimum.

The alarms generated by the prototype have to be analyzed by a **security analyst** for further inspection. This actor is responsible for monitoring security incidents reported by the classification algorithm. Due to limited time to handle an incident, no more than a total of 100 incidents per day can be reported, including false positives. Ideally, the amount of incidents will go down to only 10 incidents per day, which is a realistic amount of true positive alarms as this is the average amount of serious incidents. This means that the classification algorithm requires a minimum precision, or Positive Predictive Value (PPV), of 10%.

The **product owner** of the CTM Endpoint Module is responsible for the development and performance of the software. To guarantee maintainability of the algorithm it should be written in Python 3.5, which is the common programming language for CTMp. A high true positive ratio is required to be able to maintain the high performance of the CTM Endpoint Module. This is preferably achieved using a new and innovative solution using a self learning anomaly detection algorithm. An implementation using open-source software libraries is preferred to avoid costs. The prototype should be able to classify potentially malicious system events from legitimate system events. The prototype is meant to explore the possibilities of classification of system processes and does not have to operate in real-time. The overall system performance of the system running the prototype does therefore not have to be taken into account.

The above descriptions of the actors are summarized as environment requirements and system requirements:

Environment Requirements

- The prototype uses a self-learning classification algorithm
- The prototype is written in Python 3.5
- The prototype must have a higher precision than the current implementation of the CTM Endpoint Module
- The performance of the system running the prototype does not have to be taken into account
- The prototype will be trained and tested with live malware samples in a virtualized environment

System Requirements

- The prototype is able to classify process creation events from a log file
- The prototype should have a precision (PPV) of at least 10%
- The prototype will not handle system events in real-time
- The prototype flags an event to monitor if the likelihood of being malicious exceeds a probability of at least 50%
- The prototype flags an event to block if the likelihood of being malicious exceeds a probability of at least 95%

5.4 System Architecture

Based on the prototyping process described in Paragraph 5.2 on page 16, the supervised learning approach was chosen for further development develop as it provided the most promising results. The complete and detailed overview of the architecture of the final prototype to be developed is provided Appendix D on page 56 and will be discussed globally in Chapter 6 on the following page.

6 Development

The system architecture which resulted from the definition phase and the exploratory prototyping is presented in this chapter, along with the actual development of the prototype. The development is split up into three different sprints with their own themes, or epics. Each sprints has its own code snippet included to provide a global overview of the actual software. This chapter concludes with the results obtained by the testing of the prototype.

6.1 System Architecture

The system architecture has been developed with the aim to keep the prototype simple, yet easy to extend if a more complex classification algorithm is required. Although the Naive Bayes classification has shown the most promising results for the exploratory prototype, it is not unimaginable that a new kind of machine learning algorithm is introduced in the future. Because of this probable scenario, the specialized instance of such a machine learning algorithm is designed to be derived from a base class Classifier which contains all default methods for classification. The modular design also makes to possible to run different models simultaneously and to compare their results. A global overview of the design of the class diagram is shown in Figure 2. Note that the K-Modes clustering algorithm and the Deep Neural Network have only been tested during the exploratory prototyping and will not be incorporated in the final prototype.

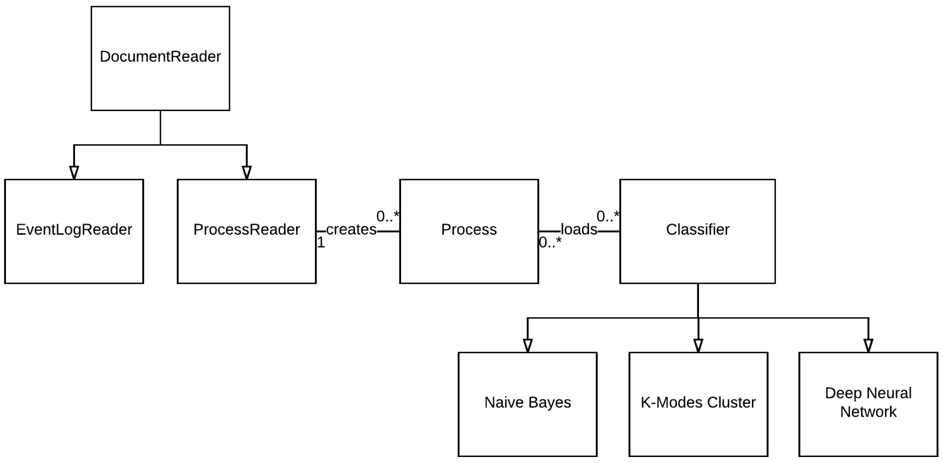


Fig. 2: Prototype system architecture

The class DocumentReader in the top-left corner of Figure 2 is a generic class responsible for loading and converting documents to a log file, which can be interpreted by the Process and Classifier class to convert to a 2-dimensional DataFrame object for classification. The subclasses EventReader and ProcessReader are responsible for loading the specific Event and Process logs, which are generated by the CTM Endpoint Server and ProcessMonitor respectively.

The Process class which is the main component of the prototype requires a ProcessMonitor log file to construct a process tree model. This model is used by a Classifier instance to make predictions of the process classes being either malicious or benign. A detailed class diagram is included in Appendix D.

The flow chart of the prototype is shown in Figure 3. The flow chart provides an overview of the internal operation of the prototype.

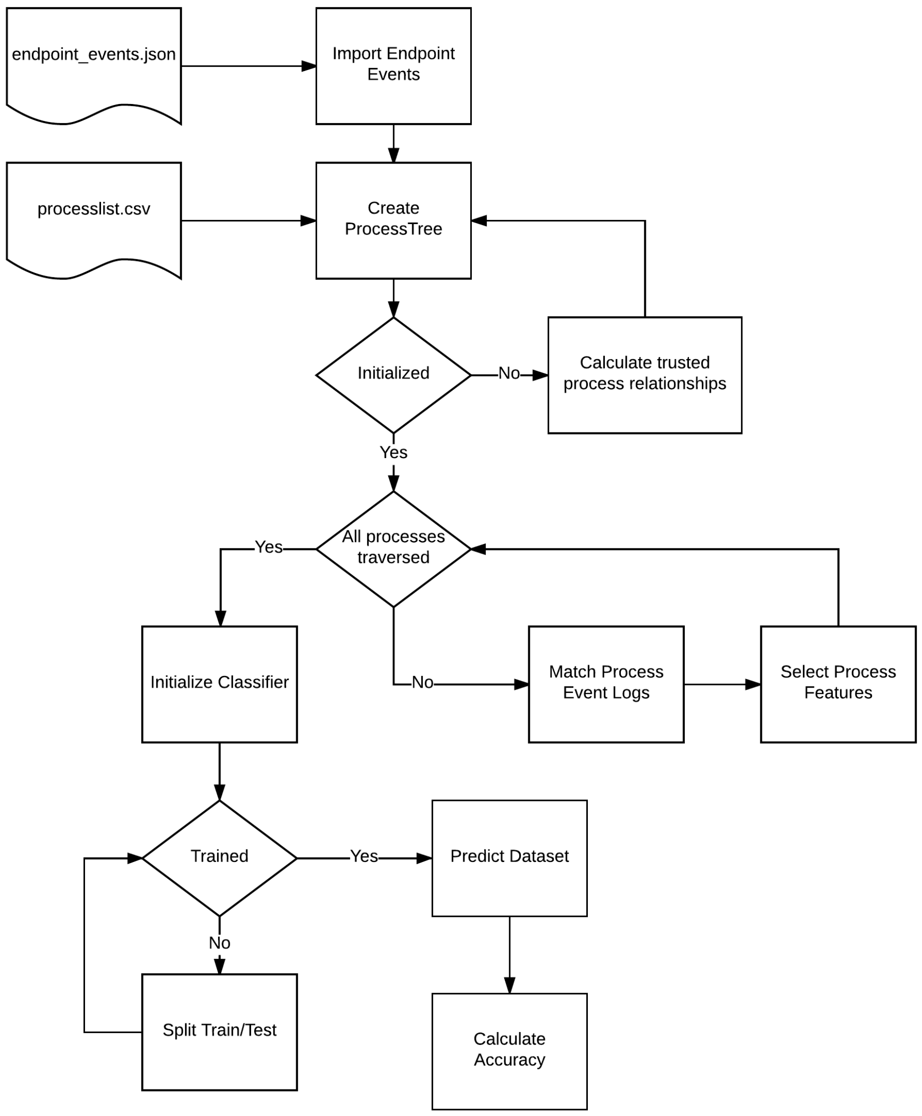


Fig. 3: Prototype flowchart

The development of the final prototype based on this system architecture is split in three sprints of three weeks each, which is described in Section 6.2 on the following page.

6.2 Sprints

An agile approach is used during the development of the prototype. This approach is chosen because it is likely that the prototype will change over time. Features such as a self-learning capability are added in a later sprint after the base model of the classifier has been implemented. The agile approach offers the possibility to often receive feedback based on the progress. Software testing has been performed simultaneous with the development of the prototype. The testing strategy is further elaborated upon in Section 6.3 on page 29. The software of the prototype is developed in Python, following the standard conventions from the PEP 8 style guide and the docstring conventions from PEP 257 [38, 39]. By creating code consistency it will be significantly easier to maintain and transfer the source code of the prototype after the end of this research.

The development phase of the research has been split up in three sprints of three weeks each. Every sprint has a main theme which has the focus on a specific group of user stories. The theme of a sprint is defined by its epic. The user stories related to the development of the final prototype are attached in Appendix **??**.

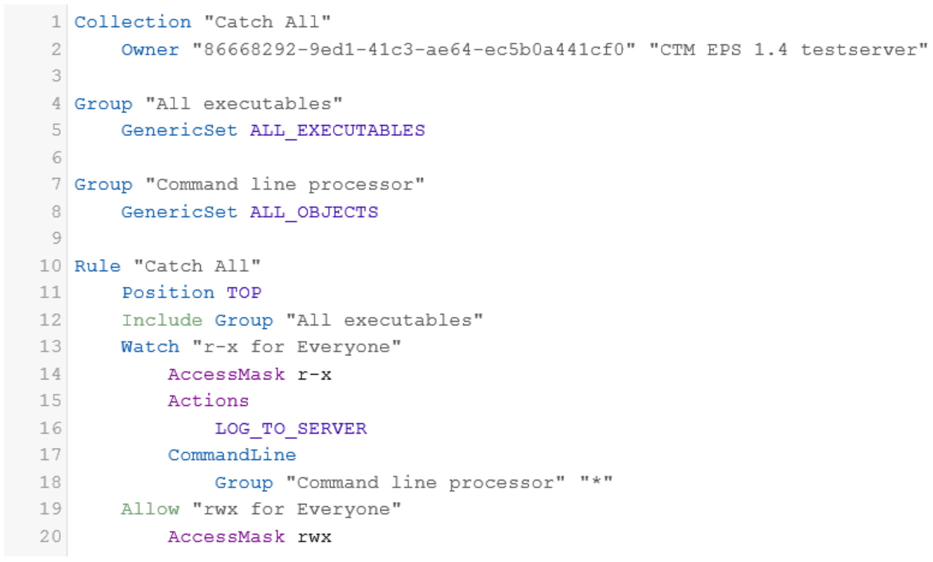
6.2.1 Data Acquisition

The Epic for this sprint is defined as:

Create a single data set with logs from multiple sources by extracting the most important features.

The first sprint is aimed at collecting and formatting the data required for the classification. In order to obtain all features required for classification, data has to be collected from multiple sources. Both the event collector from the CTM Endpoint Module and ProcessMonitor supply a dataset containing process events which are merged to create a new dataset with combined features. Feature selection is applied on this dataset to reduce the amount of features to enable faster classification and improved understanding of the data, as mentioned in Section 5.1.3 on page 12. Finally, the dataset is converted to a DataFrame object to make it suitable for classification by a machine learning algorithm.

The process of data collection and rule set configuration is described in more detail in Appendix G on page 77. Figure 4 on the facing page shows the custom rule that has been created in order to log only process create events to be used as input for the prototype. Appendix G.2 on page 78 provides an explanation of the operation of the CTM Endpoint Server rule set.



```
Collection "Catch All"
    Owner "86668292-9ed1-41c3-ae64-ec5b0a441cf0" "CTM EPS 1.4 testserver"

Group "All executables"
    GenericSet ALL_EXECUTABLES

Group "Command line processor"
    GenericSet ALL_OBJECTS

Rule "Catch All"
    Position TOP
    Include Group "All executables"
    Watch "r-x for Everyone"
        AccessMask r-x
        Actions
            LOG_TO_SERVER
        CommandLine
            Group "Command line processor" "*"
    Allow "rwx for Everyone"
        AccessMask rwx
```

Fig. 4: CTM Endpoint Server process monitoring rule

Unfortunately, multiple logging methods were required because they both provided different information about features. The CTM Endpoint Module of Fox-IT provided a lot of information about the single processes on a system, but because of the current implementation of the logging it does not contain process IDs (PIDs) of child processes. Without these PIDs it is not possible to create a process tree.

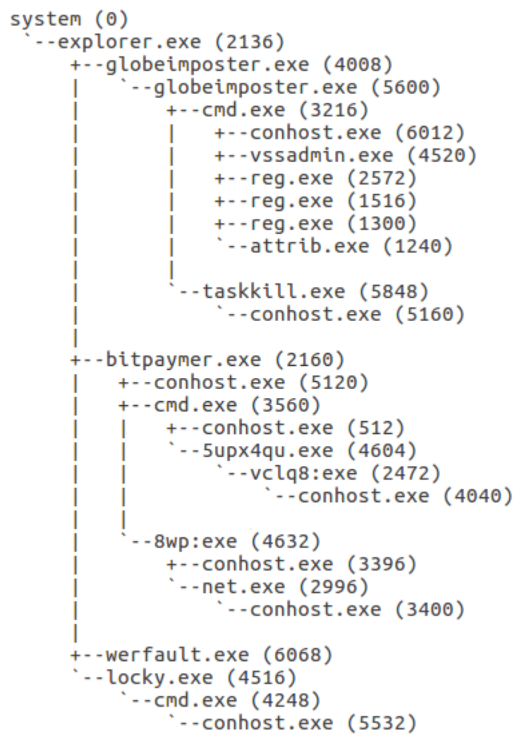
In order to obtain this missing information the ProcessMonitor logs are used, which do store the PIDs of the child processes. These logs can be matched based on their name, parent process ID and command line arguments. When a match is found the informative features are merged to establish a final process tree.

Algorithm 5 is a global example of the basic logic used for the modeling of the process tree.

```
# The process tree root is used for the linking child processes
root = Process(name='system', pid=0, children=[], root=True, parent=None)

for line in processfile.csv:
    features = line.split()
    Process(**features)

class Process:
    def __init__(self, **kwargs):
        self.features = **kwargs.split()
        if (self.root) or (root is None):
            return
        if self.children is None:
            self.parent.children.append(self)
        else:
            child = Process(parent=self, self.child_args)

        # If the parent process already exists because it already has created
        # another child process, add the new child to the existing parent.
        if child is not None:
            for process in root.walk():
                if (process.pid == self.pid) and (process.name == self.name):
                    if child is not None:
                        process.children.append(child)
                    return

        # Set references for top-level processes without a parent
        if self.parent is None:
            self.parent = root
            root.children.append(self)
```

Algorithm 5: Process tree creation pseudo code

This algorithm provides a model of the system process tree which looks like the following shown in Figure 5 on the facing page, when displayed in the command line interface of the prototype. Each line in this figure represents a process and its corresponding process ID. Other attributes, such as the process features are omitted in this figure. This model allows computational operations required in order to make predictions regarding the malware probability of a process, which is further developed in the next sprint.



```
system (0)
 `--explorer.exe (2136)
     +--globeimposter.exe (4008)
     |   `--globeimposter.exe (5600)
     |       +--cmd.exe (3216)
     |       |   +--conhost.exe (6012)
     |       |   +--vssadmin.exe (4520)
     |       |   +--reg.exe (2572)
     |       |   +--reg.exe (1516)
     |       |   +--reg.exe (1300)
     |       |   `--attrib.exe (1240)
     |       |
     |       `--taskkill.exe (5848)
     |           `--conhost.exe (5160)
     |
     +--bitpaymer.exe (2160)
     |   +--conhost.exe (5120)
     |   +--cmd.exe (3560)
     |   |   +--conhost.exe (512)
     |   |   `--5upx4qu.exe (4604)
     |   |       `--vclq8:exe (2472)
     |   |           `--conhost.exe (4040)
     |   |
     |   `--8wp:exe (4632)
     |       +--conhost.exe (3396)
     |       `--net.exe (2996)
     |           `--conhost.exe (3400)
     |
     +--werfault.exe (6068)
     `--locky.exe (4516)
         `--cmd.exe (4248)
             `--conhost.exe (5532)
```

Fig. 5: Process tree representation

6.2.2 Classification

This sprint's epic is:

Accurately classify system processes as malicious or benign.

The main focus of this sprint is the integration of the classification algorithm. According to the prototyping phase described in Section 5.2 on page 16, a Naive Bayes classification algorithm is most suitable for the current data set. This classification algorithm knows three different types of classification for different types of data sets, because a data set containing continuous variables has different characteristics compared to a data set containing discrete or binary variables [30].

- Gaussian: This type of Naive Bayes is used for data sets of which the features follow a normal distribution.
- Multinomial: Is used for discrete feature variables.
- Bernoulli: This is a binomial model which is used when the features of the data set are binary data.

Because the features of the processes to be classified are reduced to binary data the Bernoulli classification will be used.

The Python libraries *pandas* and *sklearn* provide a solid framework for data science and machine learning [28, 29] . These libraries will be used for the development of this research as they are commonly used for scientific research, data science and machine learning and have proven their effectiveness. Extensive documentation of the software libraries makes it a good choice for the implementation in the final prototype to be developed.

Algorithm 6 shows the basic structure of the Classifier base class, including a specialized instance of the NaiveBayesClassifier.

```
class Classifier:
    def __init__(self, df, target)
        self.df = df
        self.target = target
        self.train, self.test = self.split_train_test(self.df)

class NaiveBayesClassifier(Classifier):
    # Set the mode of the classifier. Bernoulli, Gaussian or Multinominal
    mode = sklearn.BernoulliNB()

    # Initialize the dataframe parameters from the base class
    def __init__(self, df, target):
        super().__init__(df, target)
        self.model = NaiveBayesClassifier.mode.fit(self.df, self.df.classes)

    # Predict the class of a process dataframe
    # Remodel the estimator model based on the prediction
    def predict(self, process):
        return self.model.predict(process)
```

Algorithm 6: Classification pseudo code

The following sprint focuses on the implementation of a self-learning classification algorithm for the prototype to further enhance its predictive capabilities.

6.2.3 Learning

The epic for the final sprint where the final improvements of the classification algorithm are implemented is:

Train the classification algorithm based on its input to improve future classifications.

The self-learning capabilities of the machine learning algorithm have been improved in this sprint. The optional features have been applied to the classification algorithm to ensure that the model can be altered after creation. This way any false positives it generated can be used as input for the classification algorithm to reduce the change of that false positive occurring again in the future.

Algorithm 6 from the previous paragraph is extended with self-learning features. The extension of this code is shown in Algorithm 7.

```
class Classifier:
    def __init__(self, df, target)
        self.df = df
        self.target = target
        self.train, self.test = self.split_train_test(self.df)

class NaiveBayesClassifier(Classifier):
    # Set the mode of the classifier. Bernoulli, Gaussian or Multinominal
    mode = BernoulliNB()

    # Initialize the dataframe parameters from the base class
    def __init__(self, df, target):
        super().__init__(df, target)
        self.model = NaiveBayesClassifier.mode.fit(self.df, self.df.classes)

    # Predict the class of a process dataframe
    # Remodel the estimator model based on the prediction
    def predict(self, process):
        cls = self.model.predict(process)
        self.remodel(process, cls)

    # Input a process dataframe including its class to remodel the estimator
    def learn(self, process, cls):
        self.remodel(process, cls)

    # Appends the process dataframe including its class to remodel the estimator
    def remodel(self, process, cls):
        self.df = self.df.append(process.append(cls))
        self.model = NaiveBayesClassifier.mode.fit(self.df)
```

Algorithm 7: Learning pseudo code

6.3 Testing

Software testing is performed using the built-in Python *unittest* library. Test cases have been developed according to the Python *unittest* standards [40].

Software testing occurred parallel to the development of the final prototype. The Use Case Test (UCT) strategy has been used in combination with Error Guessing (EG). The UCT strategy requires the use cases defined in Appendix E on page 68 as a basis for the test development. In addition to the UCT strategy the Error Guessing technique is applied in order to to test the cases which are prone to failures, based on experience gained during the development of the prototype.

The test cases included in Appendix E.1 are based on the use cases and Appendix E.2 includes the test cases based on the error guessing technique. The use-case based tests consist of a test-suite for each user story, which are built up of different test cases.

Especially the error-guessing test cases revealed various software bugs in the code for which would occur in uncommon situations, such as the same process being added to the process tree with a different reference, or the occurrence of a circular reference resulting in a infinite loop while traversing the process tree. Some failures which have been discovered by Error Guessing have not been fixed in the prototype as they have an insignificant impact on the prototype itself and its final results.

All use case based tests which are most important for the validation of the correct execution of the prototype are successful. These results verify that the prototype is working according to the expectations given normal circumstances. Table 6 provides a summary of the test results from the test cases from Appendix E.

Tab. 6: Test results

Test Case	**Result**
TST-EG-1	Test Successful
TST-EG-2	Test Failed
TST-UC-1	Test Successful
TST-UC-2	Test Successful
TST-UC-3	Test Successful
TST-UC-4	Test Successful
TST-UC-5	Test Successful
TST-UC-6	Test Successful
TST-UC-7	Test Successful
TST-UC-8	Test Successful

The test case that failed, TST-EG-2, did not impact the outcome of the results of the prototype. The bug in the software is documented so that it can be fixed in the event of the implementation of the prototype in the future.

7 Results

A prototype was developed to classify system events as either malicious or benign. The goal of the research has therefore been reached. The actual performance of the prototype is presented in the following section.

7.1 Classification performance

The prototype managed to detect all of the malware samples that were used as input, while only resulting in 4 false positives for the complete dataset.

The classification results of the prototype are shown in Figure 6. The orange bars represents all process which are actually malicious and the blue bars represent benign processes. At the first sight, it would appear that a lot of malicious processes would go unnoticed if they have a malware probability score of 0.5 or less. However, this is not the case. All processes that are malicious and score below 0.95 are actually child processes of the initial malicious executable. These processes also do not have to be malicious by nature, but are marked as malicious because they are a child of a malicious process. If the prototype would be implemented in real time and all processes that would score above the threshold of 0.95, its child processes that would create false negatives would never be created. This is why only the first alert is taken into account when the threshold is reached, and not the child processes that will be created afterwards.

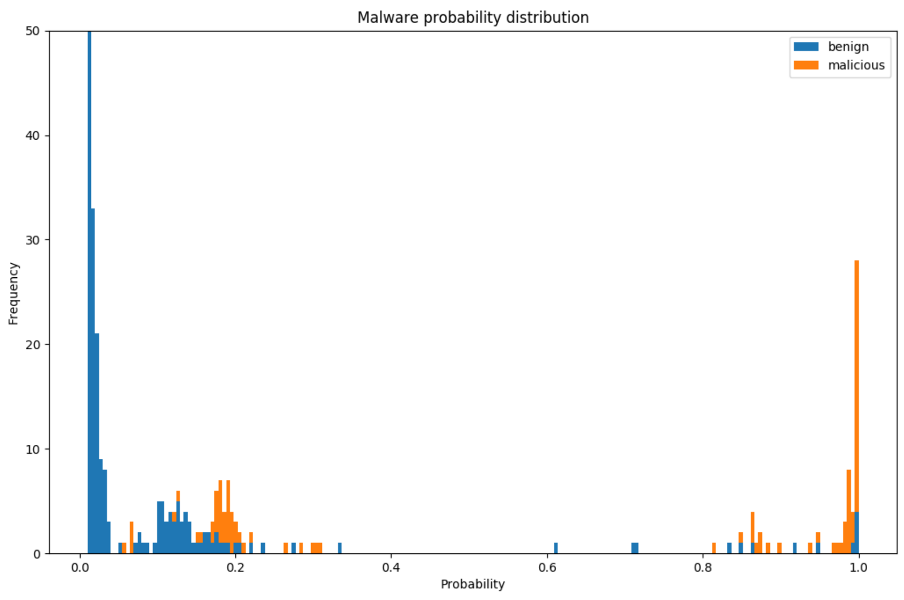


Fig. 6: Malware probability histogram

A total of 17 cases were created for all the malware samples, of which 4 were false positives. The predictive ability of the model using the cases to cluster alerts is represented by the confusion matrix in Table 7 on the next page.

A numeric representation of the output of the classification algorithm is shown in the confusion matrix in Table 7. This confusion matrix displays the results of the predictive capabilities of the algorithm to easily deduct which prediction are made correctly and incorrectly. It is important to mention that the results in the confusion matrix are only the predicted values with a probability of 95% or higher to be

Tab. 7: Classification confusion matrix

Predicted \ Actual	**True**	**False**
True	13	4
False	0	1110

The precision of the output is calculated using the following formula:

$$PPV = \frac{TP}{TP + FP} \tag{4}$$

With the data from the confusion matrix from table 7 the precision of the prototype is calculated.

$$PPV = \frac{13}{13 + 4} * 100\% = 76,5\%$$

The classification algorithm manages to reach an precision of 76.5%.

The overall accuracy of the classification algorithm is to be determined by the following formula:

$$ACC = \frac{(TP + TN)}{(TP + FP + FN + TN)} \tag{5}$$

The overall accuracy of the model is therefore:

$$ACC = \frac{13 + 1110}{13 + 4 + 0 + 1110} * 100\% = 99,65\%$$

The TPR and the FPR are:

$$TPR = \frac{13}{13 + 0} * 100\% = 100\%$$

$$FPR = \frac{4}{4 + 1110} * 100\% = 0,35\%$$

Although the False Positive Rate is very low, it should be noted that there is a clear explanation for false positives that occurred as they did indeed show all the required signs of a malicious process which should be worth further investigation, if they occurred in a real-world scenario. The occurrence of false positives and their origin is further discussed in Chapter 9 on page 35.

8 Conclusion

The prototype provided exceedingly well results, and managed to obtain a precision of 76.5%. The minimum required precision of 10% has therefore been greatly surpassed and the prototype provided results that were well beyond the initial expectations.

The classification algorithm was able to classify every single malicious process correctly whilst producing a relatively low amount of false positives, which makes it a very suitable approach for malware classification. This approach does not only reduce the amount of false positives by more than 96%, but it is also capable of alerting on malicious processes which can not be detected in the current situation.

The prototype was primarily developed for the classification of ransomware, however, other generic malware samples were also classified correctly by its algorithm. It can therefore be concluded that the chosen approach makes it possible to classify generic types of malware just as well as ransomware, although the prototype has initially been designed for the latter.

Based on these results from Chapter 7, the main research question can be answered:

Is it possible to accurately classify system processes on a Windows endpoint as ransomware based on its features?

Yes. Ransomware, and malware in general, show specific characteristics which make it very suitable for ransomware classification. By creating a model of the system process tree, underlying process relationships can be used to provide an accurate classification of malicious processes and its sub-processes.

The sub-questions, which were a part of the literature research and the prototyping, will be summarized again. These question did eventually lead to the conclusion stated above.

1. *How are the features of system processes on a Windows endpoint being logged?*

   The features of system processes are logged by the CTM Endpoint module and contain a total of 25 features, which are used to extract more specific features from. However, because the process ID's are not being logged, the third party utility ProcessMonitor is required in order to log all required data for the construction of a process tree and classification.

2. *What are the strongest features of ransomware noticeable on an endpoint?*

   Ransomware, and malware in general, contains specific features which can be distinguished from benign processes. Literature research in Section 5.1 on page 10 has pointed out the different characteristics of malware which would provide strong indicators, which have been tested and validated during the development of the prototype. The strongest features are a combination of 6 different characteristics which are commonly shared by malware. These are:

   - The directory the process resides in
   - If the process is signed
   - Deletion of a system shadowcopy
   - Process meta-data such as a description or icon

- If the process executes a scripting file
- If the process is a known child process of its parent

3. *Which type of algorithm is most suitable for classification?*

   The Naive Bayes classification algorithm proved to be most suitable for the current dataset. Its accuracy, simplicity and speed are factors that make this algorithm very suitable for the classification of the dataset. This has been confirmed by the phase of structural exploratory prototyping in Section 5.2 on page 16

4. *What is a reasonable accuracy for the classification of system processes?*

   The accuracy does not provide much information about the overall performance of an estimator model. Instead, the precision is a value that provides a lot more information about the actual performance of the algorithm. In accordance with the product owners and security analysts, a minimum precision of a classification algorithm was determined at 10%.

8.1 Recommendations

With the outstanding results that have been obtained by the prototype, it is recommended to continue the development of the prototype and to incorporate it in the driver of the CTM Endpoint Module to support the real-time monitoring of system processes. The implementation of the classification algorithm is expected to provide very little overhead for the end user of an endpoint, as there are relatively few process creation events on an endpoint when compared to all other system events.

If the updated CTM Endpoint Module is deployed on multiple endpoints, significantly more data can be collected to train the self-learning algorithm that it uses. When extensively trained, the algorithm is expected to provide even more reliable results. This research regarding the system based behavioral modeling of process events provides a solid basis for the first steps towards the transformation of the CTM Endpoint Module into a Next Generation Anti-Virus solution.

9 Evaluation

Originally, the goal of the research has been formulated to detect anomalous patterns in the Windows kernel in real-time to determine if an endpoint is being infected with ransomware. However, quickly after the start of the research the description has been altered in a way to provide more focus on researching the feasibility of anomaly detection itself. Therefore, both the real-time constraint and the direct interface to the Windows kernel are not applicable anymore in the newly formulated goal of the research as described in Chapter 4. Instead, event logs of endpoint devices were collected and used for static classification of system process events.

During the research there were three factors which influenced the progress that had not been foreseen. It was expected that malware logs were already available from earlier analyses. There were logs available, although the specific information required for this research was not present. Therefore a test environment had to be set up for malware analysis.

The Endpoint Module did not provide the required features for process tree creation, which is why the third party application ProcessMonitor was required in order to enhance the logs of the Endpoint Module. Although this logging required a significant additional effort for the development of the prototype, it did provide additional insights in system processes which benefited the research.

A third hiccup during the research was a ransomware sample which unexpectedly encrypted a remote virtual machine where the logs were written to. This specific action has been defined as a risk during the risk analysis in Section 3 on page 9. Although the most recent Windows updates and security measures were in place the ransomware still managed to bypass the Windows Firewall and Windows Defender to successfully encrypt the machine, which resulted in a loss of the log files. The impact of the risk was reduced to a minimum because a snapshot of the earlier known-good state could still be restored.

These three factors which influenced the research eventually resulted in a lack of time to test a more representative sample size of both malicious and benign processes. The data set which was used for testing contained a total of 1127 processes, which included 12 different malware samples and various different installers and other legitimate programs. Although a malware sample is made up of one or more different processes and can therefore contain more than one malicious process, the data set is relatively small. Ideally, a larger amount of malware and legitimate installers would have been used as input for the prototype to draw more substantiated conclusions.

However, the used malware samples were a random selection and obtained from real-world encounters, which makes it a representative sample set for this research and therefore provide sufficient basis to substantiate the drawn conclusions.

References

[1] Rhode, M. (2017). Early Stage Malware Prediction Using Recurrent Neural Networks.

[2] Russinovich, M. (2015). Malware Hunting with the Sysinternals Tools.

[3] Kornblum, J. (2010). Windows Memory Analysis.

[4] Mathews, L. (2017). NotPetya ransomware attack cost shipping giant Maersk over 200 million. https://www.forbes.com/sites/leemathews/2017/08/16/notpetya-ransomware-attack-cost-shipping-giant-maersk-over-200-million/. Accessed on: September 2017.

[5] Khandelwal, S. (2017). Petya Ransomware spreading rapidly worldwide, just like WannaCry. https://thehackernews.com/2017/06/petya-ransomware-attack.html. Accessed on: September 2017.

[6] Periman, K. (2017). Ransomware lessons for the financial industry. https://blogs.cisco.com/financialservices/ransomware-lessons-for-the-financial-services-industry. Accessed on: September 2017.

[7] Morgan, S. (2017). Ransomware damage report. https://cybersecurityventures.com/ransomware-damage-report-2017-5-billion/. Accessed on: September 2017.

[8] Johnson, B. (2016). What is next generation Antivirus? https://www.carbonblack.com/2016/11/10/next-generation-antivirus-ngav/. Accessed on: December 2017.

[9] Treit, R. (2017). Detonating a bad rabbit: Windows Defender Antivirus and layered machine learning defenses. https://blogs.technet.microsoft.com/mmpc/2017/12/11/detonating-a-bad-rabbit-windows-defender-antivirus-and-layered-machine-learning-defenses/. Accessed on: December 2017.

[10] Fox-IT. Security solutions to protect your business. https://www.fox-it.com/en/our-areas-of-expertise/cyber-threat-management/. Accessed on: December 2017.

[11] Fox-IT. For a more secure society. https://www.fox-it.com/en/about-fox-it/manifesto/, Accessed on: November 2017.

[12] Fox-IT. Our services & technology. https://www.fox-it.com/en/our-technology-services/. Accessed on: December 2017.

[13] Russinovich, M. Process Monitor v3.40. https://docs.microsoft.com/en-us/sysinternals/downloads/procmon. Accessed on: October 2017.

[14] Hulse, E. (2016). The general behavior of ransomware. https://blogs.cisco.com/security/the-general-behavior-of-ransomware. Accessed on: October 2017.

[15] Sophos News (2015). The current sate of ransomware: CryptoWall. https://news.sophos.com/en-us/2015/12/17/the-current-state-of-ransomware-cryptowall/,. Accessed on: October 2017.

[16] Van Dantzig, M. (2017). FAQ on the WanaCry ransomware outbreak. https://blog.fox-it.com/2017/05/13/faq-on-the-wanacry-ransomware-outbreak/. Accessed on: October 2017.

[17] Bacurio, F. (2017). Locky Unleashes Multiple Spam Waves with a New Variant "ykcol". https://blog.fortinet.com/2017/09/21/locky-unleashes-multiple-spam-waves-with-a-new-variant-ykcol. Accessed on: November 2017.

[18] Cimpanu, C. (2017). BitPaymer Ransomware Hits Scottish Hospitals. https://www.bleepingcomputer.com/news/security/bit-paymer-ransomware-hits-scottish-hospitals/. Accessed on: November 2017.

[19] Zhang, X. (2017). Analysis of New GlobeImposter Ransomware Variant. https://blog.fortinet.com/2017/08/05/analysis-of-new-globeimposter-ransomware-variant. Accessed on: November 2017.

[20] Fortiguard Labs (2017). Jaff Ransomware. https://fortiguard.com/encyclopedia/botnet/7630282. Accessed on: November 2017.

[21] Lakhani, A. (2017). Critical Update: WannaCry Ransomware. https://blog.fortinet.com/2017/08/05/analysis-of-new-globeimposter-ransomware-variant. Accessed on: November 2017.

[22] Fortiguard Labs (2017). Geodo. https://fortiguard.com/encyclopedia/botnet/7630048. Accessed on: November 2017.

[23] Zhang, X. (2016). Deep Analysis of the Online Banking Botnet Trickbot. https://blog.fortinet.com/2016/12/06/deep-analysis-of-the-online-banking-botnet-trickbot. Accessed on November 2017.

[24] Fortiguard Labs (2016). Kronos. https://fortiguard.com/encyclopedia/botnet/7629833. Accessed on: November 2017.

[25] Brownlee, J. (2016). Naive Bayes for Machine Learning. https://machinelearningmastery.com/naive-bayes-for-machine-learning/. Accessed on: October 2017.

[26] Huang, Z. (1997). A Fast Clustering Algorithm to Cluster Very Large Categorical Data Sets in Data Mining.

[27] pypi (2017). kmodes 0.7. https://pypi.python.org/pypi/kmodes/. Accessed on: October 2017.

[28] Pandas. https://pandas.pydata.org/. Accessed on: October 2017.

[29] Scikit-learn. http://scikit-learn.org/stable/. Accessed on: October 2017.

[30] Scikit-learn. Naive Bayes. http://scikit-learn.org/stable/modules/naive_bayes.html. Accessed on: October 2017.

[31] TensorFlow. An open-source software library for Machine Intelligence. https://www.tensorflow.org/. Accessed on: October 2017.

[32] TensorFlow (2017). tf.contrib.learn.DNNClassifier. https://www.tensorflow.org/api_docs/python/tf/contrib/learn/DNNClassifier. Accessed on October 2017.

[33] Symantec. (2017). Ransom.Wannacry. https://www.symantec.com/security_response/writeup.jsp?docid=2017-051310-3522-99&tabid=2. Accessed on: October 2017.

[34] Elastic.co, X-Pack for the Elastic Stack. https://www.elastic.co/guide/en/x-pack/current/xpack-introduction.html. Accessed on: September 2017.

[35] Rieck, K. (2008). Learning and Classification of Malware Behaviour.

[36] Ye, Y. (2010). Automatic Malware Categorization Using Cluster Ensemble.

[37] Hofmeyr, S.A. (1998). Intrusion Detection using Sequences of System Calls.

[38] Van Rossum, G., Warsaw, B., Coghlan, N. (2013). PEP 8 – Style Guide for Python Code. `https://www.python.org/dev/peps/pep-0008/`. Accessed on: November 2017.

[39] Goodger, D., Van Rossum, G. (2001). PEP 257 – Docstring Conventions. `https://www.python.org/dev/peps/pep-0257/`. Accessed on: November 2017.

[40] Python Unittest Documentation. `https://docs.python.org/3/library/unittest.html`. Accessed on: November 2017.

[41] Brownlee, J. (2014). Classification Accuracy is Not Enough: More Performance Measures You Can Use. `https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/`. Accessed on: December 2017.

[42] Knight, W. (2017). DARPA is funding project that will try to open up AI's black boxes. `https://www.technologyreview.com/s/603795/the-us-military-wants-its-autonomous-machines-to-explain-themselves/`. Accessed on: December 2017.

[43] VirtualBox. Virtual Networking. `https://www.virtualbox.org/manual/ch06.html`. Accesed on: October 2017.

[44] CTM Endpoint Module Dev.Team (2017). Rule User Manual for CTM Endpoint Module

A Afstudeerplan

Titel afstudeeropdracht:

Countering malware using anomaly detection of endpoint events in the Windows kernel

Opdrachtomschrijving

Bedrijf

Fox-IT is een Nederlands bedrijf dat is gespecialiseerd in cybersecurity. De grootste klanten van het bedrijf zijn overheden, financiële instellingen en bedrijven met een vitale infrastructuur.

De opdracht is ontstaan door de grote opkomst van ransomware in het afgelopen jaar. Door de grote toename in deze soort malware is het van belang dat er een werkend concept wordt ontwikkeld om dit soort besmettingen bij klanten van Fox-IT te kunnen voorkomen. Omdat het klantenbestand van Fox-IT bestaat uit organisaties met een vitale infrastructuur dient deze opdracht een significant maatschappelijk belang.

Voor de opdracht worden malware samples beschikbaar gesteld door Fox-IT. Door deze samples in een Elasticsearch database te laden kan anomaly detection worden toegepast of specifieke patronen te bepalen. Met behulp van machine learning kunnen deze patronen gebruikt om nog onbekende patronen te kunnen beoordelen op ransomware.

Probleemstelling

Er is niet voldoende kennis in de cybersecurity sector om een aanval met ransomware in real-time te kunnen detecteren op end point devices die op Windows draaien.

Doelstelling van de afstudeeropdracht

Het doel van de opdracht is om real-time afwijkende patronen te kunnen detecteren in de Windows kernel om zo te kunnen beoordelen of een end point device wordt getroffen door ransomware.

Resultaat

Om de doelstelling van de opdracht te kunnen bereiken wordt een prototype ontwikkeld dat gebruik maakt van machine learning dat in staat is om de afwijkende patronen van malware te kunnen detecteren in de Windows kernel.

Uit te voeren werkzaamheden, inclusief een globale fasering, mijlpalen en bijbehorende activiteiten

Initiatiefase (1 week)

- Verhelderen opdracht
- Project risico's analyseren

- Planning opstellen

Definitiefase (7 weken)

- Inlezen naar malware karakteristieken
- Inlezen naar machine learning voor beoordelen events
- Exploratory structural prototyping voor bepalen (on)mogelijkheiden
- Achterhalen behoeften belanghebbenden
- Opzetten architectuur

Ontwikkelfase (3x 3 weken)

- Schrijven van een rule set voor anomaly detection
- Ontwikkelen prototype
- Testen prototype

Op te leveren (tussen)producten

- Plan van Aanpak
- Definitierapport
- Ontwikkelrapport
- Testrapportage

Te demonstreren competenties en wijze waarop

G1: Praktische aspecten hanteren in (internationale) projecten
Dit wordt aangetoond aan de hand van de risicoanalyse in het plan van aanpak.

A1: Analyseren van het probleemdomein
Dit wordt aangetoond aan de hand van de beschrijving van de prototypes in het definitierapport.

C8: Ontwerpen van een technisch informatiesysteem
Dit wordt aangetoond aan de hand van het ontwerp van de rule set en de prototypes.

D16: Realiseren van een technisch informatiesysteem
Dit wordt aangetoond aan de hand van de gebruikte frameworks.

D17: Testen van software systemen
Dit wordt aangetoond aan de hand van de testrapportage in het ontwikkelrapport.

B Plan of Action

B.1 Introduction

The past year a major increase in ransomware infections has taken place. Due to the increase in this specific type of malware with disastrous results, it is crucial to develop a solution to detect and prevent a possible outbreak. Traditional event monitoring is not advanced enough to detect these types of advanced threats, which is why a new approach is required for system event monitoring. This research, commissioned by Fox-IT, is focused on the research and development of a new concept in cybersecurity event monitoring which is theoretically capable of detecting and preventing ransomware infections in real-time.

The research of this assignment will be focused on the feasibility of realizing an anomaly based event detection algorithm to evaluate if events on endpoint devices are potentially malicious. If this research proves to be positive, a prototype will be developed to improve the accuracy and to further reduce the alerts generated from system events collected by the Endpoint module of the Cyber Threat Management platform (CTMp) of Fox-IT.

B.2 Background

Traditional event monitoring is based upon static rulesets. These rulesets work by the principle of whitelisting and blacklisting specific actions, therefore creating a sandbox-like environment for endpoints. Configuration of these rules is a labor-intensive tasks for both the initial configuration and maintenance afterwards, while still not being able to reduce the amount of false positives to the desired minimum. To avoid the many downsides of the traditional rule set implementation, it is desired to develop an anomaly based detection algorithm which is a completely new technique in endpoint security monitoring. This research is focused on the various possible concepts of such algorithms and their implementation in the Endpoint module of CTMp.

B.3 Goal

The goal of the research is to propose a solution in the form of a working prototype that is able to classify system event data samples that are retrieved from Windows endpoints into potentially malicious and benign events.

B.4 Assignment

To reach the goal of the research as stated in Paragraph B.3, research will have to be performed regarding the feasibility of an anomaly based detection algorithm for data samples of system events. Based on the results on the research a prototype will be developed utilizing an anomaly based detection algorithm to detect and alert on potentially malicious events.

Candidate Requirements

The requirements of the prototype have been established in accordance with the company supervisor. The following candidate requirements have been formulated at the start of the research:

- The prototype must be written in the default programming language used by the Endpoint development team, which is Python 3.5
- The prototype should be able to detect and alert on potentially malicious events
- Input for the classification algorithm consists of malware samples and endpoint event data samples from Fox-IT
- The accuracy of the classification algorithm has to be higher than the currently implemented ruleset in the Endpoint module of CTMp
- The anomaly detection algorithm of the prototype should make use of a self-learning classification algorithm
- The prototype does not have to process system events in real-time

B.5 Planning

The research will consist of three separate phases, each with their own deliverables. Table 8 on the facing page shows each phase with its corresponding tasks and duration.

The initiation phase of the research is used for elaboration of the original assignment. The possibilities and impossibilities are discussed with the company supervisor to be able to create a realistic planning for the total duration of the research. The plan of action is delivered by the end of this phase.

During the definition phase the main focus is on the exploratory structural prototyping, where the feasibility of several different anomaly based concepts are tested. This approach is used as it allows to quickly receive feedback in a short time span to determine the stakeholders' needs and to refine the actual software specifications, if necessary. When the details of the final prototype to be developed are in place, the epics will be written to describe the global features of the software that will make up the most viable product. Based on the epics, a software architecture is developed to be used as a starting point for the actual development of the software. The definition report is completed by the end of this phase.

The last phase of the assignment, the development phase, consists of three sprints of three weeks each. This time span of three weeks creates enough time to be able to develop significant increments, while still having sufficient iterations to adjust if necessary. The epics written during the definition phase will be used as an initial starting point. At the end of the first two sprints new user-stories will be written for the following sprint, based on remaining system requirements. At the end of this phase the development report and the test report are completed.

Tab. 8: Project planning

Phase	Task
Initiation (1 week)	Assignment description
	Analyzing project risks
	Create planning
Definition (7 weeks)	Read about malware characteristics
	Read about machine learning possibilities
	Exploratory structural prototyping
	Determining the stakholders' needs
	Write epics
	Software architecture development
Development (3 x 3 weeks)	Software development
	Software testing

B.6 Project Boundaries

The research was initiated on September 4^{th}, 2017 and will end on December 21^{th}, 2018. This means there are effectively 17 weeks available to perform this research, which is in accordance with the planning as described in Table 8 in Paragraph B.5 on the preceding page.

Before implementing an algorithm, research has to be performed to determine whether it is actually possible to implement a classification algorithm based on the available data to train a classifier. Fox-IT has various data samples available of both malware and legitimate data which can be used for training and testing of the classification algorithm. If this data does not seem to be sufficient for the training of a classification algorithm another solution will be implemented, which is described in Table **??** in Paragraph B.8 on the following page.

The prototype is primarily developed for endpoint devices of 'Local User' accounts. Endpoints that are used by 'Administrator' accounts will not be taken into account for this research.

There is no budget required for software for the successful completion of the research. All required tools are available as open-source software.

B.7 Deliverables

Each phase of the research will yield its own deliverables, such as described in Paragraph B.5 on the preceding page. The following products will be delivered at the end of the internship:

- Plan of Action
- Definition Report
- Development Report
- Test Report

B.8 Risk Analysis

Insufficient access to internal resources
Due to the moderate screening level, there is no full access to internal resources. If specific restricted resources are required, this could become a problem. This risk is described in Table 9. It is also relevant for the access to malware samples, as there might be no authorization to run real malware in an environment set up for this research.

Tab. 9: Insufficient access to internal resources

Description	Insufficient access to internal resources
Probability	Medium
Impact	Research cannot be performed or validated without the required test data
Remediation	Discuss the need of the specific resources with the stakeholders to get access, or try to implement a workaround of the solution which does not require this specific resource.

Ransomware encrypts endpoint
Ransomware samples have to be run on a network an monitored for their activity in order to make predictions based on its behavior. If the ransomware manages to break out of its contained environment it will encrypt the (log) files on the system, resulting in a loss of work.

Tab. 10: Ransomware encrypts endpoint

Description	Ransomware encrypts endpoints (on the network)
Probability	Medium
Impact	Log files or other work might get encrypted and therefore lost
Risk reduction	The ransomware samples are executed in a virtualized internal network environment disconnected from an outside network. All machines except the targeted machine are provided with the latest updates and security patches
Impact reduction	Regular snapshots of known-good points in time are made to create a system restore. Only log files are stored on the machine
Contingency Plan	Restore the most recent snapshot

C Definition Report

C.1 Problem Analysis

Fox-IT does not have sufficient knowledge to detect and prevent an advanced ransomware attack on Windows endpoint devices.

In order to detect ransomware attacks, the detection mechanism of the anti-virus (AV) solution will have to be improved. A new approach is required which resulted in this research with the following primary research question:

Is it possible to accurately classify system processes on a Windows endpoint as ransomware based on its features?

In order to classify system processes, a selection of features has to be defined to serve as input for the classification algorithm. The selection of these features and the type of algorithm used for classification is determined by the following subquestions:

1. *How are the features of system processes on a Windows endpoint being logged?*
2. *What are the strongest features of ransomware noticeable on an endpoint?*
3. *Which type of algorithm is most suitable for classification?*
4. *What is a reasonable accuracy for the classification of system processes?*

C.1.1 Signature based detection

Traditional consumer anti-virus (AV) solutions which provide endpoint security are not capable of preventing advanced threats. However, neither do enterprise-grade intrusion detection systems. Currently, most malware detection approaches use common signature based detection algorithms. These algorithms scan the file system for hashes (signatures) of known malicious files, or scan files for known malicious byte patterns. This approach works pretty well, although this is only the case if the signatures are updated regularly. In the event that a malicious file is altered, its signature changes and the file is not detected anymore. Because the traditional AV approach can only detect known signatures, it is impossible to detect zero-day exploits which can be the initial attack vector of a ransomware attack.

While this technique of signature-based malware detection is used for most AV solutions for personal use, it is not advanced enough to provide proper protection for high-value targets with a vital infrastructure such as the customers of Fox-IT.

C.2 Ransomware analysis

The Threat Intelligence (TI) department of Fox-IT has provided several malware samples to be used for generating test data. Section C.2.1 on the next page discusses the malware characteristics found by literature research and Section C.2.2 on page 47 relates these features to the samples provided by Fox-IT.

C.2.1 Ransomware features

Ransomware, and malware in general, is known to show certain anomalies from regular system processes. For regular users, processes are created in a predictable manner [3]. Most processes are started by *explorer.exe* and not by *cmd.exe*. Furthermore certain operating system services are known not to create new processes, which makes it suspicious and anomalous behavior if they do. This is a major indicator which should be used when evaluating process creation events.

Russinovich, a renowned security researcher at Microsoft, states that ransomware has several characteristics that can be observed during execution [2]. Ransomware is typically not a type of malware that is designed to stay on a system undetected as it is meant to extort its victim. The best way to detect ransomware would be to monitor processes on the system with a tool such as Process Monitor [13]. Russinovich states that processes that are worth investigating are processes that contain features such as the lack of an icon, have no description, are unsigned, or live in the Windows directory or user profile. These specific features can be explained due to the fact that malware is developed with the purpose to extort its victim and not to be a high-quality piece of software. Another recommendation given by the security researcher from Microsoft is to be especially wary of items residing the the Windows directory or the AppData directory of the user profile.

According to Cisco, one of the most important characteristics of ransomware is the deletion of system shadow copies, which prevent a system backup restore [14]. In 2014, the CryptoWall ransomware was the first ransomware family which incorporated this technique after it had been disclosed that a system could be restored with a shadow copy [15]. The process to delete a shadow copy from the file system should therefore also be a major indicator of ransomware. Because administrative privileges are required to perform this action, an elevated process should be hijacked first.

The recent WannaCry ransomware from May 2017 did show traces of all features mentioned in this section which eventually led to the successful encryption of over 160,000 computers worldwide [16]. Using a publicly available exploit the process *lsass.exe*, which has administrative privileges, was hijacked to eventually start the process *tasksche.exe* which launches a command prompt to delete shadow copies and create persistence at boot to show the ransom note by adding a key to the registry. Under normal circumstances, the process *lsass.exe* should never start *tasksche.exe*.

To summarize, the features of system processes that will be used for classification are the following:

- Residing directory
- Unsigned executables
- Deletion of shadow copies
- Lack of process meta-data
- Unknown process relationship
- Scripting files

C.2.2 Ransomware samples

The following ransomware samples have been provided by Fox-IT. These samples have been harvested directly from infected machines. The malware has not been altered and should therefore be representative for this research.

The ransomware samples have been run on a network of virtualized endpoints. Both the CTM Endpoint module and the tool ProcessMonitor collected logs of system process events during the execution of the ransomware samples. Both monitoring tools were required to create a complete overview of the system, by merging the features of both logs. The process and the setup used for collecting process event logs is described in more detail in Appendix G.

- Locky: Ransomware family released in 2016. The ransomware became active again in the summer of 2017 when it was spread through multiple SPAM emails containing an invoice that required payment. The attached document was a Microsoft Word document that contained malicious macros [17].
- Bitpaymer: A ransomware variant targeted at hospitals. The ransomware unpacks several malicious unsigned executables into the user profile folder [18].
- GlobeImposter: This ransomware variant is obtained through a drive-by executed by a malicious Javascript file. This ransomware deletes shadowcopies from the system [19].
- Jaff: A ransomware variant that has been primarily spread through SPAM emails and spearphishing attacks [20].
- WannaCry: The infamous ransomware which gained a lot of publicity because it managed to disrupt infrastructure on a global level [21].

In addition to the ransomware samples mentioned above, two additional malware samples have been provided. Note that these samples are not ransomware and are only being tested in order to see if the classification algorithm is also able to detect other kinds of malware instead of only ransomware which it will initially be trained for.

- Geodo: Botnet used as a loader for more malware and SPAM [22].
- Trickbot: Banking malware which injects malicious code into the browser when specific websites are accessed [23].
- Kronos: Banking malware which injects malicious code into the browser when specific websites are accessed [24].

C.3 Machine Learning

A self-learning classification algorithm is a solution which is likely to solve the problems introduced by static signature-based detection as described in Section C.1. This type of classification algorithm is able to evaluate events and to classify them either as malicious or benign. The main strength of self-learning classification is that it is able to classify events which have not been hard-coded into the detection which is the case for signature-based detection. This means that new variants of ransomware families, or even completely new ransomware families, could still be detected dynamically based on their features instead of a static hash of the executable. With the self-learning capabilities of the classification algorithm it is even able to make even more accurate predictions after it has seen a specific sample multiple times. The learning curve, or the convergence rate, determines how well the classification algorithm is able to adjust itself. This way, false positives can be used as input for the algorithm to reduce the probability of occurrence in the future.

Four different approaches for a machine learning algorithm are chosen for the exploratory prototyping. These different algorithms will be tested and evaluated based on their performance. The algorithm which performs the best will be chosen for the implementation of the actual prototype. The following metrics are used:

- Accuracy - The accuracy of the classifier based on a manually crafted data set for training and testing
- Noise - How well the algorithm is able to handle irregularities in the dataset
- Convergence - How well the classifier is able to learn from new data samples
- Execution time - How fast the classifier is able to make a prediction
- Complexity - The complexity of the algorithm behind the classification

C.3.1 Classification algorithms

A classification algorithm will be implemented to test which type of algorithm is most suitable for the classification of system processes. A variety of different approaches for machine learning implementation has been chosen in order to test the various different characteristics of these algorithms. A brief description of the algorithms which will be incorporated in the prototyping phase is presented below:

- Counting

This is the most trivial approach for the classification problem. Given the set of features, the amount of features that indicate ransomware can be counted and if a certain threshold is reached the process is being classified as malicious. Although this solution is very easy to implement it is expected to generate a lot of false positives and false negatives. Another downside is that using this approach shows many similarities to a rule-based AV solution, which makes it a less feasible solution. This is the only approach not implementing a machine learning algorithm.

- Naive Bayes

The Naive Bayes algorithm is a supervised classification algorithm that assumes independence among data features. This algorithm is known to perform relatively well in general while being easy to implement. It is a probabilistic algorithm that is trained by using already classified data. To train the classifier, both benign and malicious data has to be collected for training [25].

- K-Modes Clustering

The algorithm will generate clusters based on the features of the input data. When using a unsupervised learning algorithm data does not have to be labeled to train the classification algorithm. However, to test the accuracy of the algorithm labeled test data is still required to determine if the predicted clusters of the classification algorithm are correct. This unsupervised learning algorithm is the only one suitable for discrete values which are used for the Endpoint data set. Because it is different from the other supervised learning algorithms it is interesting to try it during the exploratory prototyping phase. Because not all data sets are suitable for clustering it is not always an effective approach [27, 26].

- Deep Neural

Network A Deep Neural Network is another approach to supervised machine learning and approximates a 'black-box' approach where the models are determined empirically, instead of theoretically [42]. The approach of a neural network makes it an interesting option to try during the prototyping phase.

C.3.2 Dimensionality Reduction

Machine learning implementations often suffer from the *curse of dimensionality*, meaning the dataset contains too many features to apply proper predictions. Datasets with too many features are often noisy and contain features which are not relevant.

Enhancing the dataset by removing redundant or irrelevant features will significantly improve the performance of the algorithm in the following ways:

- Better understanding of the data
- Improving prediction accuracy
- Faster predictions

The most expressive features of the dataset have to be selected to optimize the classification. The features mentioned in Paragraph C.2.1 on page 46 will therefore be the features that will remain after applying dimensionality reduction to the dataset.

Feature Selection The CTM Endpoint module stores 25 different features of a process, with an average of 15 KB of data per entry. This is a significant amount of data to process, especially with the aim of real-time processing in the future. The feature selection process could still provide a great improvement in performance.

The initial amount of 25 features can be reduced to only 14 which are required to extract data from to construct the features mentioned in Section C.2.1. This new dataset has an average size of 1 KB for each entry which therefore results in a 15-fold increase in performance.

Feature Extraction The newly created feature set of 14 features is used to extract the six features mentioned in Section C.2.1. These features have been set by applying simple boolean logic to check if specific values are present that apply to the corresponding feature. This results in a 1-dimensional dataset consisting of only 6 binary features.

C.3.3 Classification accuracy

Brownlee, the author of various books regarding machine learning, provided a writeup on the pitfalls of calculating the accuracy of classification algorithms and how this result can often be misleading [41]. The results are misleading because of the class imbalance in the dataset which is used for classification, which is also referred to as the *Accuracy Paradox*. Additional measures are required to properly evaluate the classification algorithm.

A measure which is commonly used which provides a good insight in the performance of the classification algorithm is the precision. This value is the number of True Positives divided by the number of True positives and False Positives. This results in a score which represents the overall ability of the classification algorithm to correctly predict True Positives. For the ransomware classification problem where the amount of malicious processes is significantly less compared to benign processes, it is desired to know the ability to correctly predict True Positive results instead of True Negatives.

To provide a clean and unambiguous solution to present the predictions made by the classification algorithm, a confusion matrix is used. This table shows a clear overview of the correctly and incorrectly predicted results which will be used to calculate the metrics to determine classifier performance. An example of a confusion matrix is provided in Figure 7 on page 32.

Tab. 11: Confusion Matrix

Predicted \ Actual	**Positive**	**Negative**
Positive	True Positive (TP)	False Positive (FP)
Negative	False Negative (FN)	True Negative (TN)

The following can therefore be said about the last subquestion of this research regarding the accuracy of the classification algorithm:

"What is a reasonable accuracy for the implementation of the classification algorithm?"

The accuracy of the algorithm itself is not relevant as this does not say much about its predictive capabilities. Instead, the precision metric is the most important for the malware classification problem. With the formula to calculate the precision of the classification algorithm, its is also significantly easier to assign a specific value to this metric. The total amount of True and False positives have to be tracked down in order to set a minimum requirement for the classification algorithm to achieve. The exact interpretation of this requirement is provided in Section 5.3 on page 20 in collaboration with the actors involved in the CTM Endpoint Module. The formula for the precision, also referred to as the Positive Predictive Value (PPV) is as follows:

$$PPV = \frac{TP}{TP + FP} \tag{6}$$

Additional metrics that provide insight in the performance of the algorithm are the False Postive Ratio (FPR) and the True Positive Ratio (TPR), which provide information about the amount of incorrectly classified negatives and correctly classified positives. The following formulas are used to calculate these metrics:

$$FPR = \frac{FP}{FP + TN} \tag{7}$$

$$TPR = \frac{TP}{TP + FN} \tag{8}$$

C.4 Stakeholder Requirements

To determine whether the prototype performs as desired, the requirements of all actors who are involved are taken into account. The actors are defined as all the people within Fox-IT who are either directly or indirectly involved with the Endpoint module itself or the alarms generated by it. The actors who are involved in the CTM Endpoint module are shown in Table 12.

Tab. 12: Stakeholder overview

Actor	Role	Impact
Security Analyst	• An employee of Fox-IT responsible for monitoring security incidents	• The security analyst can't do his work if the system is flooded with indicators
End User	• The employee of a customer of Fox-IT • User of the Endpoint module	• Affected by system performance • Unable to work during ransomware infection
Product Owner	• Responsible for development of CTMp Endpoint module • Responsible for development of CTMp Endpoint module	• Better performance and reliability of Endpoint server • Requires new approach to continue development

The **end user** has a computer with the CTM Endpoint Module installed on it. This actor is directly affected by the performance of the Endpoint module. The end user does not want to be infected with ransomware and therefore the endpoint module should block a malicious event before it does irreversible harm to the system of the end user. The end user does not want to receive unnecessary notifications of possible insecure system behavior, which means that false positive alerts should be reduced to a minimum.

The alarms generated by the prototype have to be analyzed by a **security analyst** for further inspection. This actor is responsible for monitoring security incidents reported by the classification algorithm. Due to limited time to handle an incident, no more than 100 incidents per day can be reported, including false positives. Ideally, the amount of incidents will go down to only 10 incidents per day, which is a realistic amount of true positive alarms as this is the average amount of serious incidents. This means that the classification algorithm requires a minimum precision, or Positive Predictive Value (PPV), of 10%.

The **product owner** of the CTM Endpoint module is responsible for the development and performance of the software. To guarantee maintainability of the algorithm it should be written in Python 3.5, which is the common programming language for CTM. A high true positive ratio is required to be able to maintain the high performance of the Endpoint module. This is preferably achieved using a new and innovative solution using a self learning anomaly detection algorithm. An implementation using open-source software libraries are preferred to avoid costs. The prototype should be able to classify potentially malicious system events from legitimate system events. The prototype is meant to explore the possibilities of classification of system processes and does not have to operate in real-time. The overall system performance of the system running the prototype does therefore not have to be taken into account.

The above descriptions of the actors are summarized as environment requirements and system requirements:

Environment Requirements

- The prototype uses a self-learning classification algorithm
- The prototype is written in Python 3.5
- The prototype must have a higher precision than the current implementation of the Endpoint Module for CTMp.
- The performance of the system running the prototype does not have to be taken into account.
- The prototype will be trained and tested with live malware samples in a virtualized environment

System Requirements

- The prototype is able to classify process creation events from a log file
- The prototype should have a precision (PPV) of at least 10%
- The prototype will not handle system events in real-time
- The prototype flags an event to monitor if the likelihood of being malicious exceeds a probability of at least 50%
- The prototype flags an event to block if the likelihood of being malicious exceeds a probability of at least 95%

C.5 Prototyping

The needs of the stakeholders that have been described in Paragraph C.4 on the previous page will be validated by using the technique of exploratory structural prototyping. Four different prototypes are developed in a short timespan to receive quick feedback from the stakeholders to determine which approach will best fit their needs. The chosen approaches for the exploratory prototyping have already been introduced in Section C.3.1.

The exploratory prototyping is primarily used to get an idea of which kind of algorithm provides the best performance for the problem described in Paragraph C.1. Because the prototyping has been performed early-on in the research, there was no verified data set available for testing purposes. To make sure the accuracy of the classification algorithms could still be tested, a dataset with similar a similar categorical data structure was manually crafted for testing purposes.

The metrics mentioned in Section C.3 are used to test the performance of the prototype.

- Accuracy
- Noise
- Convergence
- Execution time
- Complexity

C.5.1 Naive Bayes

The first prototype is using the Naive Bayes classification algorithm. This type of algorithm is relatively easy to implement yet known to produce accurate results. Although it is an algorithm based on a relatively simple concept, it often has an exceedingly well performance compared to other more complex algorithms. Naive Bayes is a probabilistic algorithm that assigns each feature a probability based on its occurrence.

For data sets similar to the Endpoint data set the implemented Naive Bayes algorithm has an accuracy of 99.9%. Algorithm 1 contains the straightforward implementation of the Naive Bayes algorithm for a Bernoulli distribution [30].

```
import sklearn.naive_bayes.BernoulliNB
df = generate_dataset()

results = BernoulliNB().fit(df.train, df.train.classes).predict(df.test)
```

Algorithm 8: Naive Bayes Prototype

C.5.2 Deep Neural Network

The following exploratory prototype makes use of a Deep Neural Network (DNN), which is a more complex approach for classification. The DNN is implemented using the open-source software library TensorFlow, which is developed by Google [31].

The configuration for the neural network, the hidden layers which are built up by neurons, is configured in the same way as is done in the examples given by TensorFlow [32]. The approach shows promising results within a relatively short execution time. The prototype with the TensorFlow implementation reaches an accuracy of 99.6% up to 100%, depending on the configuration of the neural network. A better accuracy does cost more computational power and therefore requires a longer execution time.

Algorithm 2 contains a code snippet from the implementation of the DNN classification algorithm.

```
import tensorflow as tf
df = generate_dataset()

cls = tf.contrib.learn.DNNClassifier(
    feature_columns=df.columns,
    hidden_units=[10, 20, 10],
    n_classes=2)
model = cls.fit(input_fn=df.train, steps=500)
results = model.evaluate(input_fn=df.test)
```

Algorithm 9: Deep Neural Network Prototype

C.5.3 K-Modes

The two previous exploratory prototypes mentioned in Sections 5.2.1 and 5.2.2 are supervised learning algorithms, which means that they require pre-classified datasets in order to train the model.

The k-modes algorithm is an unsupervised machine learning algorithm and does not require labeled data to train the classifier with. Based on the features of the data set the algorithm will create *n* clusters which new entries will be related to. Using the k-modes algorithm new data entries will be classified to the nearest cluster. As the dataset is difficult to classify manually because of the many different (unknown) processes, an unsupervised approach might be preferred over supervised classification.

However, in order to validate if the output of the classification algorithm is correct, the process data samples still need to be classified for testing. Another common problem of clustering is that not all data sets are suitable for clustering which might lead to wrong results.

The clustering algorithm reaches an accuracy of approximately 85% up to 90%. The code snippet of the prototype implementing the k-modes cluster is provided in Algorithm 3.

```
import kmodes
df = generate_dataset()

km = kmodes.KModes(n_clusters=2, n_init=10)
results = km.fit(df)
```

Algorithm 10: K-Modes Prototype

C.5.4 Counting

A different and more trivial approach which does not use machine learning would be to count the amount of features that are positive. Because the dataset contains binary data, where a '1' would indicate a feature is present and '0' means it does not, this approach is actually expected to perform overall pretty well. The threshold to classify a data sample as malicious is set to 3; which means half of the features in the dataset should be positive.

The accuracy of this prototype provided similar results to the Naive Bayes classification algorithm provided in Section 5.2.1, although the accuracy of this exploratory prototype would drop drastically if the dataset contains a lot of noisy features.

The algorithm to count the features in the data set is provided in Algorithm 4.

```
df = generate_dataset()
results = [None] * len(df)

for index in range(len(df)):
        results[i] = 1 if sum(list(df[i]) > 3 else 0
```

Algorithm 11: Feature Counting Prototype

C.5.5 Conclusion

All classification algorithms performed rather well. The k-modes clustering algorithm was the only algorithm that did not provide sufficient results to consider for further development. Although the other algorithms (Naive Bayes, Deep Neural Network, and Counting) provided a good accuracy overall, the Naive Bayes machine learning algorithm is chosen for further development as it provides the best results overall. Table 13 shows the scores assigned to each metric.

Tab. 13: Exploratory prototyping metrics

Category	**Naive Bayes**	**K-Modes Cluster**	**Deep Neural Network**	**Counting**
Accuracy	+ +	−	+ +	+
Noise	−	−	+	− −
Convergence	+	−	+ +	− −
Execution time	+	+	−	+
Complexity	+	+	− −	++

During the prototyping phase a new requirement has come forward. Currently, the CTM Endpoint module can either allow or block system events. The prototype can, due to the use of a machine learning algorithm and the behavioral model of the system process tree, assign probabilities to sequential events. Each branch of the process tree will be monitored individually. If a branch reaches a certain threshold it can be flagged to monitor by a security analyst, and if a next threshold is surpassed the process can be blocked completely. A branch in the process tree might be used to accumulate a total malware score. There should be no time constraint in place to monitor sequential events, as malware might make use of a the internal Windows Task Scheduler *tasksche.exe* to purposely delay it's execution [33].

C.6 System Architecture

Based on the prototyping process described in Paragraph C.5 on page 52, the supervised learning approach has been chosen to develop further after the first promising results. The detailed design of the final prototype is included in Appendix D.

D Development Report

This report elaborates on the development of the machine learning algorithm used to evaluate potentially malicious system events.

D.1 Prototype Design

The system architecture has been developed with the aim to keep the prototype simple, yet easy to extend if a more complex classification algorithm is required. Although the Naive Bayes classification has shown the most promising results for the exploratory prototype, it is not unimaginable that a new kind of machine learning algorithm is introduced in the future. Because of this probable scenario, the specialized instance of such a machine learning algorithm is designed to be derived from a base class Classifier which contains all default methods for classification. The modular design also makes to possible to run different models simultaneously and to compare their results. A global overview of the design of the class diagram is shown in Figure 2. Note that the K-Modes clustering algorithm and the Deep Neural Network have only been tested during the exploratory prototyping and will not be incorporated in the final prototype.

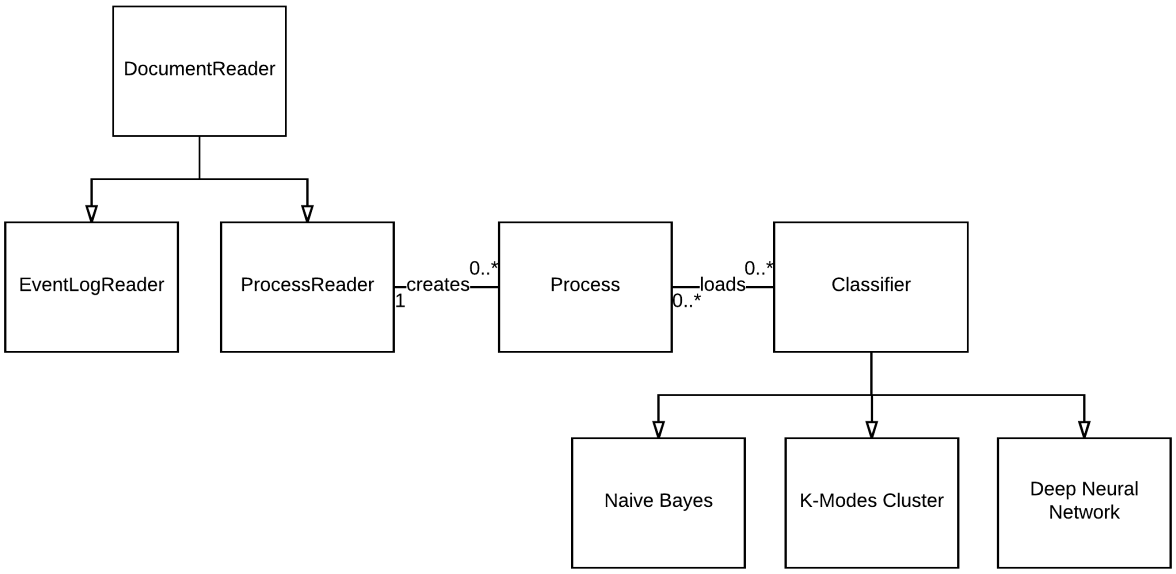


Fig. 7: Global prototype class diagram

The class DocumentReader in the top-left corner of Figure 2 is a generic class responsible for loading and converting documents to a log file, which can be interpreted by the Process and Classifier class to convert to a 2-dimensional DataFrame object for classification. The subclasses EventReader and ProcessReader are responsible for loading the specific Event and Process logs, which are generated by the endpoint server and ProcessMonitor respectively.

The *Process* class is the base entity for a system process. The attributes of this class include the features which will be used for classification by the classifier. A new instance of a process is created for each process found by the *ProcessReader* class. The process class creates a new child Process instance if the process is a parent process. It will then link itself to its parent and child class by adding a reference to these instances. The class method *treeview()* will start at the root process and traverse the complete tree structure recursively structure based on references linked to child processes.

Classifier is the base class for classifier models. Different types of classification algorithms inherit the data structure from their parent class including methods for creating a dataset for training and testing, and to learn or predict new samples entered into the classifier. Each subclass of the *Classifier* base class contain methods specific to the classification algorithm. For the final prototype, only the *NaiveBayesClassifier* class has been implemented.

Figure 8 shows the detailed class diagram including functions and attributes off the DocumentReader component of the prototype in relation to the Process class.

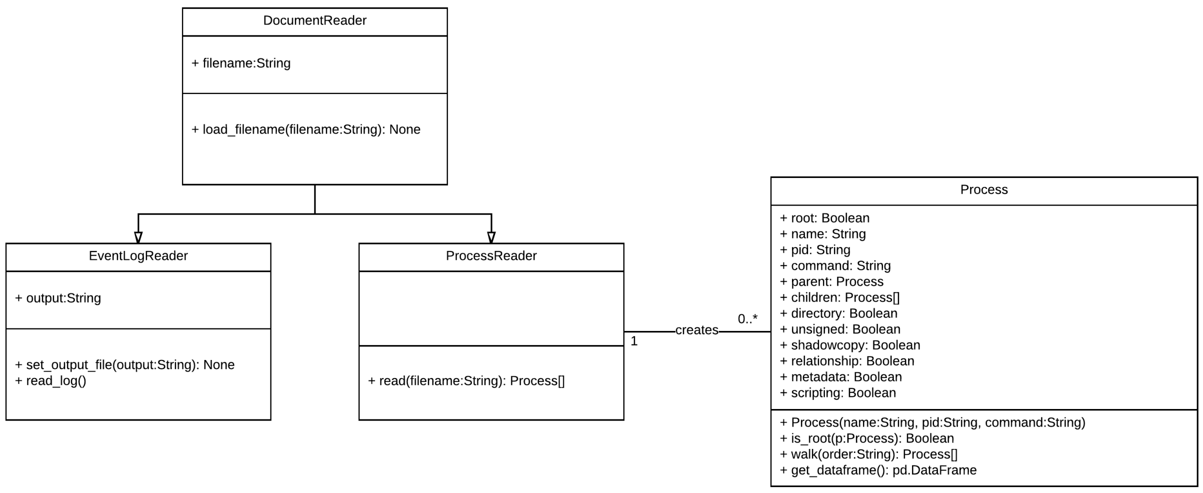


Fig. 8: DocumentReader class diagram

Figure 9 on the following page shows the detailed relationship between the Process class and the Classifier.

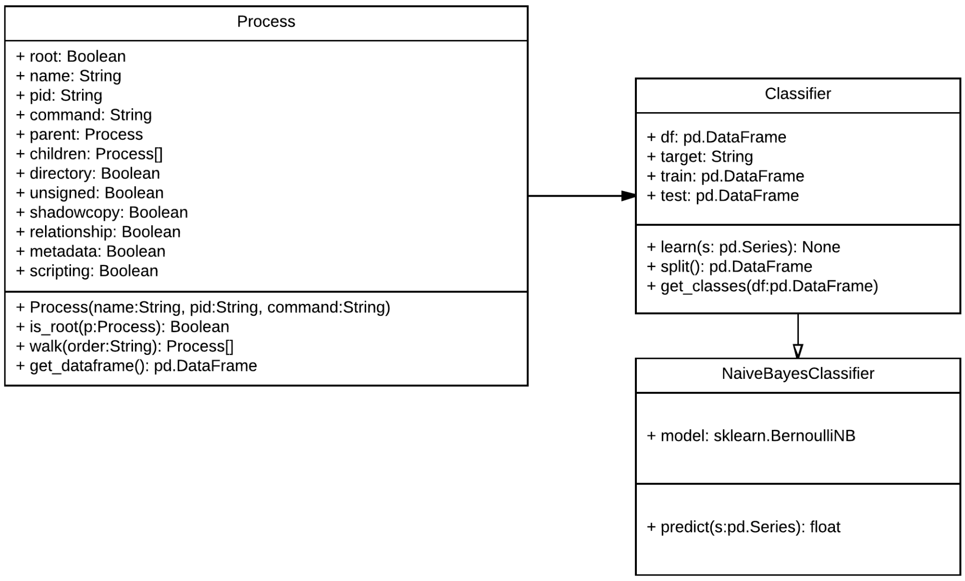


Fig. 9: Classifier class diagram

The flow chart of the program is shown in Figure 10 on the next page. The flow chart provides an overview of the internal operation of the prototype.

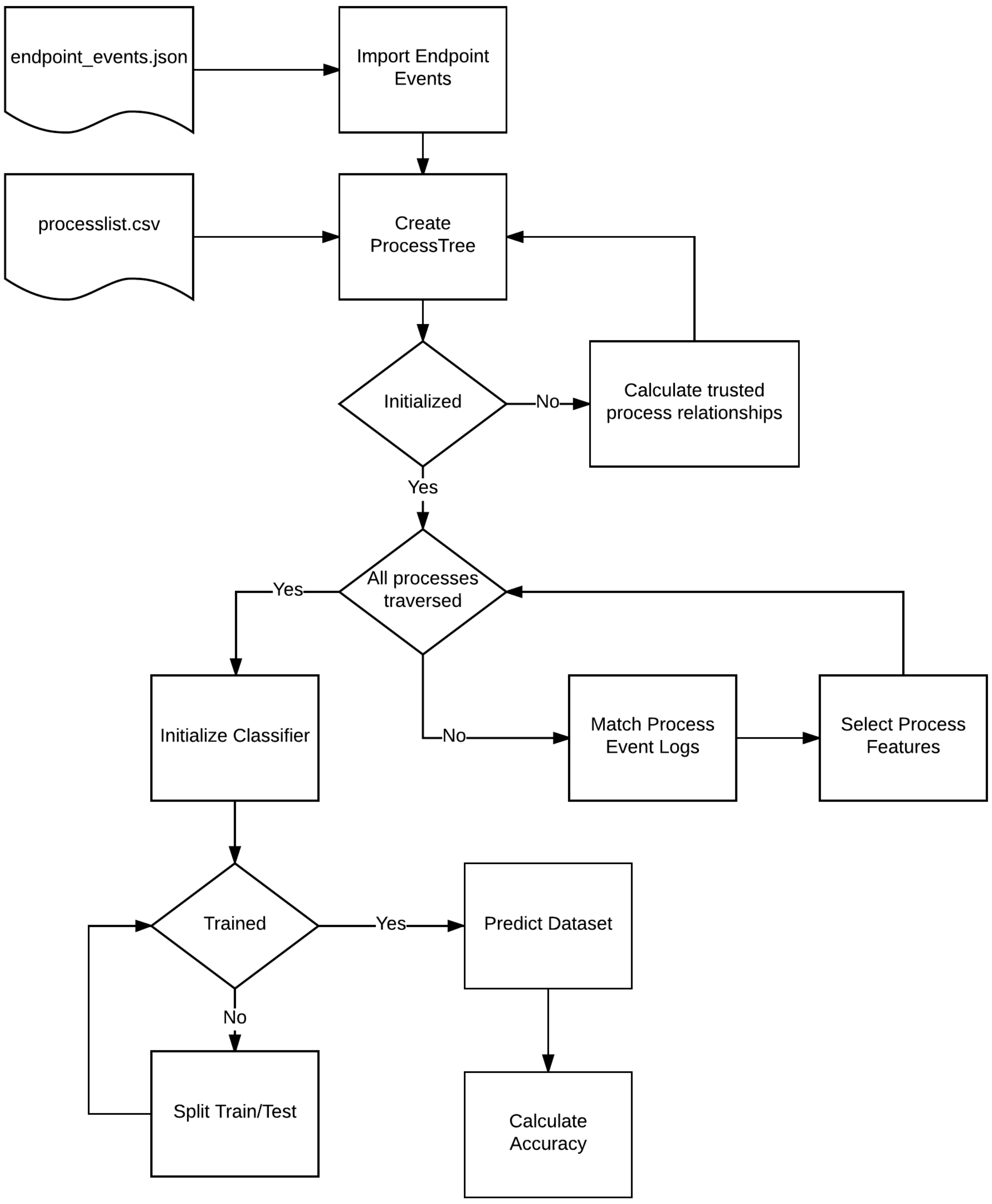


Fig. 10: Prototype flowchart

D.2 Prototype Backlog

The prototype development is divided into the themes *data acquisition, classification, and accuracy*. Those themes describe the three main components of the prototype and are translated to an epic.

The theme *classification* contains *epics* for the type of classification algorithm being implemented. The size of a user story is determined by the points it has, where a single point is the equivalent of four hours, and therefore 2 points make up a single day.

Based on the requirements that have been determined in Appendix C.4 on page 51 the following user stories are written. The tests for the user stories are attached in Appendix E on page 68.

D.2.1 Data Acquisition

This sprint Epic is defined as the following:

Create a single data set with logs from multiple sources by extracting the most important features.

A detailed description of epic and its user stories is provided below:

Tab. 14: EP-1 Data Acquisition

Type	Epic
Name	EP-1 Data Acquisition
Description	Create a single data set with logs from multiple sources by extracting the most important features.
Acceptance Criteria	• The feature set has been reduced • The feature set contains the most important features regarding ransomware • Irrelevant features regarding ransomware are discarded • Features are expressed as binary data
Size	30

Tab. 15: US-1 Data Acquisition

Type	User Story
Name	US-1 Data Acquisition
Description	As a product owner, I want the prototype to use only the most expressive features, so that classification will be faster and more accurate.
Acceptance Criteria	• Irrelevant features regarding ransomware are discarded • The feature set has been reduced
Size	10

Tab. 16: US-2 Data Acquisition

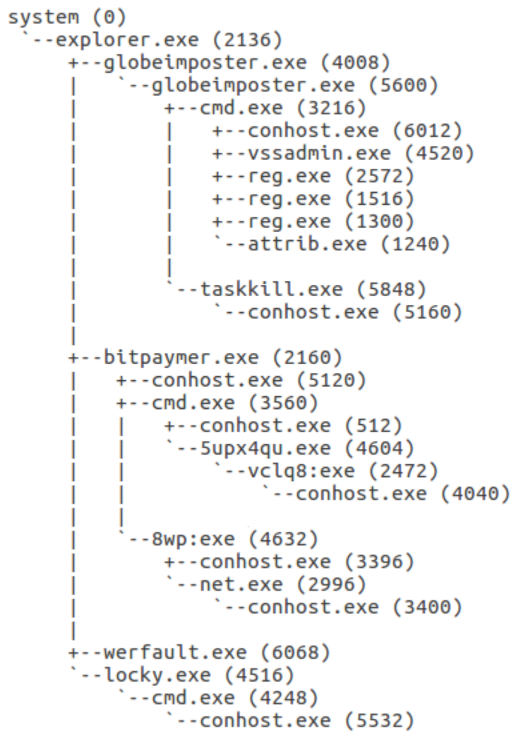
Type	User Story
Name	US-2 Data Acquisition
Description	As a product owner, I want the prototype to create a model from the system process tree, so that process relationships can be mapped
Acceptance Criteria	• Every process has a parent/child relationship • A process tree is created from the input log files
Size	20

Algorithm 12 shows a simplified overview of a part of the code which resulted from the first sprint.

```
# The process tree root is used for the linking child processes
root = Process(name='system', pid=0, children=[], root=True, parent=None)

for line in processfile.csv:
    features = line.split()
    Process(**features)

class Process:
    def __init__(self, **kwargs):
        self.features = **kwargs.split()
        if (self.root) or (root is None):
            return
        if self.children is None:
            self.parent.children.append(self)
        else:
            child = Process(parent=self, self.child_args)

        # If the parent process already exists because it already has created
        # another child process, add the new child to the existing parent.
        if child is not None:
            for process in root.walk():
                if (process.pid == self.pid) and (process.name == self.name):
                    if child is not None:
                        process.children.append(child)
                    return

        # Set references for top-level processes without a parent
        if self.parent is None:
            self.parent = root
            root.children.append(self)
```

Algorithm 12: Process tree creation pseudo code

This algorithm provides a model of the system process tree which looks like the following shown in Figure 11 on the next page when displayed in the command line interface of the prototype. Each line in this figure represents a process and its corresponding process ID. Other attributes, such as the process features are omitted in this figure.



```
system (0)
 `--explorer.exe (2136)
     +--globeimposter.exe (4008)
     |   `--globeimposter.exe (5600)
     |       +--cmd.exe (3216)
     |       |   +--conhost.exe (6012)
     |       |   +--vssadmin.exe (4520)
     |       |   +--reg.exe (2572)
     |       |   +--reg.exe (1516)
     |       |   +--reg.exe (1300)
     |       |   `--attrib.exe (1240)
     |       |
     |       `--taskkill.exe (5848)
     |           `--conhost.exe (5160)
     |
     +--bitpaymer.exe (2160)
     |   +--conhost.exe (5120)
     |   +--cmd.exe (3560)
     |   |   +--conhost.exe (512)
     |   |   `--5upx4qu.exe (4604)
     |   |       `--vclq8:exe (2472)
     |   |           `--conhost.exe (4040)
     |   |
     |   `--8wp:exe (4632)
     |       +--conhost.exe (3396)
     |       `--net.exe (2996)
     |           `--conhost.exe (3400)
     |
     +--werfault.exe (6068)
     `--locky.exe (4516)
         `--cmd.exe (4248)
             `--conhost.exe (5532)
```

Fig. 11: Process tree representation

D.2.2 Classification

The Epic for this sprint is defined as:

Accurately classify system processes as malicious or benign.

A detailed description of epic and its user stories is provided below:

Tab. 17: EP-2 Classification

Type	Epic
Name	EP-2 Classification
Description	Accurately classify system processes as malicious or benign.
Acceptance Criteria	• The precision of the classification algorithm is at least 10% • Processes are being classified with a probability between 0% and 100% • The classifier is trained using a different data set than the testing data.
Size	30

Tab. 18: US-3 Classification

Type	User Story
Name	US-3 Classification
Description	As a product owner, I want the classification algorithm to use a training and a testing dataset, so that the validity of process is ensured.
Acceptance Criteria	• The classification algorithm is able to train using a training dataset • The accuracy of the classification can be tested using a testing dataset
Size	10

Tab. 19: US-4 Classification

Type	User Story
Name	US-4 Classification
Description	As a product owner, I want the classification algorithm to predict the likelihood of a process being malicious, so that malicious processes can be blocked
Acceptance Criteria	• Processes are being classified with a probability between 0% and 100% • The classification algorithm uses a Bernoulli Naive Bayes implementation to make predictions
Size	8

Tab. 20: US-5 Classification

Type	User Story
Name	US-5 Classification
Description	As a security analyst, I want to see less false positives, so that I will work more efficiently
Acceptance Criteria	• An overall precision of the classification algorithm of at least 10%
Size	4

Tab. 21: US-6 Classification

Type	User Story
Name	US-6 Classification
Description	As a security analyst, I want the prototype to correlate multiple similar alerts into a single case, so that I have more contextual information
Acceptance Criteria	• Alerts are grouped by branch • The root process of a branch is the parent process of the first encountered malicious process in the branch. • The case is named after the root process of its branch
Size	4

Tab. 22: US-7 Classification

Type	User Story
Name	US-7 Classification
Description	As an endpoint user, I want the classification algorithm to block all ransomware processes, so that my work does not get lost
Acceptance Criteria	• The classification algorithm does not produce false negative alerts • The probability threshold for blocking a process is set at the optimal level
Size	4

A part of the simplified code that resulted from the second sprint is provided in Algorithm 13 on the facing page:

```
class Classifier:
    def __init__(self, df, target)
        self.df = df
        self.target = target
        self.train, self.test = self.split_train_test(self.df)

class NaiveBayesClassifier(Classifier):
    # Set the mode of the classifier. Bernoulli, Gaussian or Multinominal
    mode = sklearn.BernoulliNB()

    # Initialize the dataframe parameters from the base class
    def __init__(self, df, target):
        super().__init__(df, target)
        self.model = NaiveBayesClassifier.mode.fit(self.df, self.df.classes)

    # Predict the class of a process dataframe
    # Remodel the estimator model based on the prediction
    def predict(self, process):
        return self.model.predict(process)
```

Algorithm 13: Classification pseudo code

D.2.3 Learning

The Epic for this sprint is defined as:

Train the classification algorithm based on its input to improve future classifications.

A detailed description of epic and its user stories is provided below:

Tab. 23: EP-3 Learning

Type	Epic
Name	EP-3 Learning
Description	Train the classification algorithm based on its input to improve future classifications.
Acceptance Criteria	• A security analyst is able to provide input for the classification algorithm to learn • The classification algorithm updates its model after a classification
Size	30

Tab. 24: US-8 Learning

Type	User Story
Name	US-8 Learning
Description	As a product owner, I want the prototype to implement a self-learning algorithm, encourage innovation
Acceptance Criteria	• The classification algorithm has a self-learning capability
Size	10

Tab. 25: US-9 Learning

Type	User Story
Name	US-9 Learning
Description	As a security analyst, I want to enter incorrect predictions into the classifier, so that the classifier will be trained for the future
Acceptance Criteria	• The classification algorithm is using a dynamic estimator model • The classification algotihm allows input to add to its model
Size	10

Tab. 26: US-10 Learning

Type	User Story
Name	US-10 Learning
Description	As a security analyst, I want the prototype to automatically learn relationships between trusted processes, so that the probability of false positives occurring is reduced
Acceptance Criteria	• Process relationships are updated automatically after classification
Size	10

A part of the resulting source code from this last sprint is show (simplified) in Algorithm 14 on the next page

```
class Classifier:
    def __init__(self, df, target)
        self.df = df
        self.target = target
        self.train, self.test = self.split_train_test(self.df)

class NaiveBayesClassifier(Classifier):
    # Set the mode of the classifier. Bernoulli, Gaussian or Multinominal
    mode = BernoulliNB()

    # Initialize the dataframe parameters from the base class
    def __init__(self, df, target):
        super().__init__(df, target)
        self.model = NaiveBayesClassifier.mode.fit(self.df, self.df.classes)

    # Predict the class of a process dataframe
    # Remodel the estimator model based on the prediction
    def predict(self, process):
        cls = self.model.predict(process)
        self.remodel(process, cls)

    # Input a process dataframe including its class to remodel the estimator
    def learn(self, process, cls):
        self.remodel(process, cls)

    # Appends the process dataframe including its class to remodel the estimator
    def remodel(self, process, cls):
        self.df = self.df.append(process.append(cls))
        self.model = NaiveBayesClassifier.mode.fit(self.df)
```

Algorithm 14: Sprint 3 code snippet

E Test Report

To verify the correct execution of the classification algorithm test cases have been written for the critical components of the prototype. Each class has its own test suite which is described below:

Software testing occurred parallel to the development of the final prototype. The Use Case Test (UCT) strategy has been used in combination with Error Guessing (EG). The UCT strategy requires the use cases defined in Section # as a basis for the test development. In addition to the UCT strategy the Error Guessing technique is applied in order to to test the cases which are prone to failures, based on experience gained during the development of the prototype.

The test cases included in Appendix E.1 are based on the use cases and Appendix E.2 includes the test cases based on error guessing.

Especially the error-guessing test cases revealed various software bugs in the code for which would occur in uncommon situations, such as the same process being added to the process tree with a different reference, or the occurrence of a circular reference resulting in a infinite loop while traversing the process tree. Some failures which have been discovered by Error Guessing have not been fixed in the prototype as they have an insignificant impact on the prototype itself and its final results.

All use case based tests which are most important for the validation of the correct execution of the prototype are successful. These results verify that the prototype is working according to the expectations given normal circumstances.

Tests are developed using the built-in Python unittest library. Test cases have been written according to the Python unittest standards [40].

E.1 Error Guessing

Tab. 27: TST-EG-1 Read log

Name	TST-EG-1 Read log
Description	The correct amount of logs are imported by the ProcessReader
Precondition	The ProcessMonitor log file is imported as 'proclog.csv', and includes at least 1 process.
Expected output	The process tree model has the same size as the unique amount of processes in the logs, excluding the root process.
Test steps	1. Get the unique amount of logs from 'proclog.csv' + 1 2. The length of the process tree model should be equal to the amount from step 1; `len(Node.root.walk()) == step 1`
Result	The test is successful. Both lengths are equal, all process are included in the tree model.

Tab. 28: TST-EG-2 Tree model

Name	TST-EG-2 Tree model
Description	Identical processes are combined as a single process.
Precondition	The ProcessMonitor log file is imported as 'proclog.csv', and includes at least 1 process. The system root process has been initialized.
Expected output	The process tree model merges identical processes as a single process
Test steps	1. Initialize an instance of the ProcessReader class; `pcr = ProcessReader()` 2. Read the file 'proclog.csv'; `pcr.read('proclog.csv')` 3. Get the amount of processes in the current tree model; `len1 = len(Process.root.walk())` 4. Read the file 'proclog.csv' again; `pcr.read('proclog.csv')` 5. Get the amount of processes in the new tree model; `len2 = len(Process.root.walk())` 6. Check if the length from the first model is equal to the length of the second model; `len1 == len2`
Result	The test failed. Processes are not merged when read directly from the log file.

E.2 DocumentReader

Tab. 29: TST-UC-1 Tree model

Name	TST-UC-1 Tree model
Description	The log files from the enpoint server and ProcessMonitor are combined to construct a model of the process tree
Precondition	Log files from the endpoint server and Processmonitor are imported as 'epslog.json' and 'proclog.csv'. A system root process has been initialized as the root for the process tree model as Process.root.
Expected output	A tree model is constructed using the file 'proclog.csv' by the ProcessReader() class. The EventLogReader() class reads the file 'epslog.csv' and appends the features to the tree model.
Test steps	1. Read the file 'proclog.csv'; `ProcessReader.read_log('proclog.csv')` 2. Read the file 'epslog.json'; `output = EventLogReader.read_log('epslog.json')` 3. Merge the log files; `Node.root.merge_logs(output)` 4. Verify all processes have been matched; `count([True if p.matched for p in Process.root.walk()]) == len(Process.root.children)`
Result	The test is successful. All processes have been matches, execpt the child processes of the system root; as these can not be logged by design as they are already started at the time the logging is initiated.

Tab. 30: TST-UC-2 Feature reduction

Name	TST-UC-2 Feature reduction
Description	The feature set of a single process entry should be reduced to a maximum of 6 featues which should be used for classification.
Precondition	A model of the processtree has been created and feature extraction has been applied.
Expected output	Every process entry has exactly 6 features. This includes the root system process, child processes and processes that do not have a reference to the endpoint log file.
Test steps	1. Iterate over the process tree model in pre-order mode; `for process in Process.root.walk(order='pre-oder')` 2. Create a dataframe for a single node; `df = process.get_dataframe()` 3. Check the length of the dataframe; `len(list(df.columns)) == 6`
Result	The test is successful. Every dataframe is of length 6; which is equal to the amount of features used for classification

Tab. 31: TST-UC-3 Feature selection

Name	TST-UC-3 Feature selection
Description	Feature selection drops the unusable features from the original log file to reduce the total amount of features.
Precondition	A model of the processtree has been created.
Expected output	The original feature set of processes has been reduced from 25 to 14 features.
Test steps	1. Iterate over the process tree model in pre-order mode; `for process in Process.root.walk(order='pre-oder')` 2. Create a dataframe for a single node; `df = process.get_dataframe()` 3. Verify the amount of features in the dataframe is equal to 14; `len(list(df.columns)) == 14`
Result	The test is successful. Every dataframe is of length 6; which is equal to the amount of features used for classification

E.3 Process

Tab. 32: TST-UC-4 Split dataset

Name	TST-UC-4 Split dataset
Description	The dataframe containing all processes is split into a training and testing dataset to determine the accuracy of the model
Precondition	A model of the processtree has been created.
Expected output	Every process entry has exactly 6 features. This includes the root system process, child processes and processes that do not have a reference to the endpoint log file.
Test steps	1. Iterate over the process tree model in pre-order mode; `for process in Process.root.walk(order='pre-oder')` 2. Create a dataframe for a single node; `df = process.get_dataframe()` 3. Verify the amount of features in the dataframe is equal to 6; `len(list(df.columns)) == 6`
Result	The test is successful. Every dataframe is of length 6; which is equal to the amount of features used for classification

Tab. 33: TST-UC-5 Predict malware probability

Name	TST-UC-5 Predict malware probability
Description	The classification algorithm predicts the likelihood of a process being malware.
Precondition	A process tree model has been initialized. The Classifier (cls) has been initialized with a dataframe which has been split up into training and testing data. The classification algorithm has been trained.
Expected output	The classification algorithm assigns each process a score between 0 and 1, where 0 would be not malicious and 1 would be very malicious.
Test steps	1. Make a single prediction for each process in the process tree; `cls.predict(p.to_dataframe()) for p in Process.root.walk(order='pre-order')`
Result	The test is successful. All processes receive a malware probability score. The processes which have been marked as malware beforehand received a score of 0.95 or higher.

Tab. 34: TST-UC-6 Cluster alerts

Name	TST-UC-6 Cluster alerts
Description	Processes being marked as malicious which are in the same branch are correlated into a single case.
Precondition	A model of the process tree has been created. The model of has been classified by the classification algorithm.
Expected output	The amount of cases created is equal to the amount of malware samp
Test steps	1. Walk the process tree in pre-order mode; `for process in Process.root.walk(order='pre-order')` 2. If the node is the first malicious node in the branch, mark the underlying branch (including the current process parent) and return; `if process.probability > threshold:` `process.parent.create_case()`
Result	The test is successful. All detected malware samples are assigned a case which includes all processes started by the malware.

Tab. 35: TST-UC-7 Classifier precision

Name	TST-UC-7 Classifier precision
Description	Calculate the precision of the classification algorithm.
Precondition	A model of the process tree has been classified by the classification algorithm.
Expected output	The precision of the classification algorithm is at least 10%
Test steps	1. Get the total amount of true positives (TP) 2. Get the total amount of false positives (FP) 3. Calculate the precision (PPV) using; `PPV = TP / (TP + FP)`
Result	The test is successful. The precision is above 10%

E.4 Learning

Tab. 36: TST-UC-8 Self-learning algorithm

Name	TST-UC-8 Self-learning algorithm
Description	The algorithm is self-learning and adjusts its model correctly after receiving feedback
Precondition	The classifier (cls) has been initialized without a model or training
Expected output	The first classification scores 50% and is therefore marked as '1'. After learning, this value is adjusted to the learning input.
Test steps	1. Get the probability of a (random) entry; `p = cls.predict(pd.DataSeries([0, 1, 0, 1, 0, 1]))` 2. Verify the probability *p* is 1, as the classifier is not trained; `p == 1` 3. Learn the classifier that the entry from step 1 is not malicious; `cls.learn(pd.DataSeries([0, 1, 0, 1, 0, 1, 0])` 4. Get the probability of the entry in step 1; `p = cls.predict(pd.DataSeries([0, 1, 0, 1, 0, 1]))` 5. Verify the probability *p* is 0, as the classifier has learned this entry is not malicious and therefore assigns a lower score; `p == 0`
Result	The test is successful. The self-learning classification model successfully learned from the user input.

F Results

This document provides the results obtained by the final prototype that was developed for this research.

F.1 Process Classification

A total of 1127 processes have been used as input for the classification algorithm. Figure 12 shows the output results of the classification algorithm. The orange bars represents all process which are actually malicious and the blue bars represent benign processes. The processes which scored lower than 0.1% are discarded from the figure, as this would create a spike of hundreds of legitimate, and only legitimate, processes.

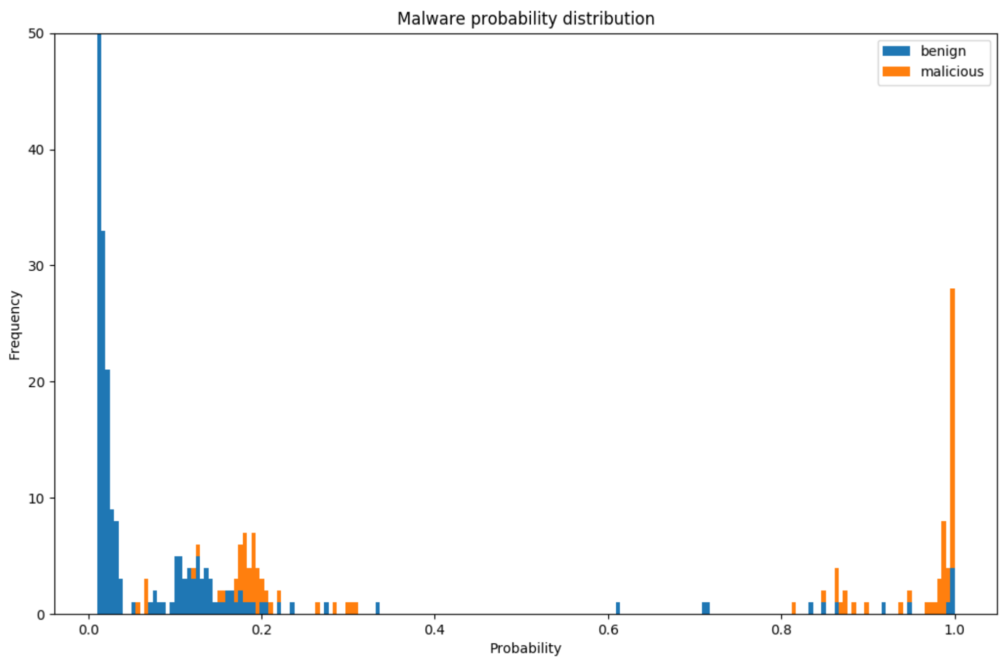


Fig. 12: Malware probability histogram

The confusion matrix in Table 37 shows the distribution of the predicted classes and their actual class.

Tab. 37: Classification confusion matrix

Predicted \ Actual	**True**	**False**
True	13	4
False	0	1110

With the data from the confusion matrix from table 7 on page 32 the precision of the prototype is calculated.

$$PPV = \frac{13}{13+4} * 100\% = 76,5\%$$

The classification algorithm manages to reach an precision of 76.5%.

The overall accuracy of the model is calculated using the following formula:

$$ACC = \frac{13+1110}{13+4+0+1110} * 100\% = 99,65\%$$

The accuracy is able to predict if a process is malicious or not with an accuracy of 99,65%.

The TPR and the FPR are:

$$TPR = \frac{13}{13+0} * 100\% = 100\%$$

$$FPR = \frac{4}{4+1110} * 100\% = 0,35\%$$

G Collector Setup

This document describes the approach to create an environment for the collection of data sets required as input for the prototype developed during this research.

G.1 Networking Setup

Figure 13 shows an overview of the network setup to collect the log files from the ransomware samples.

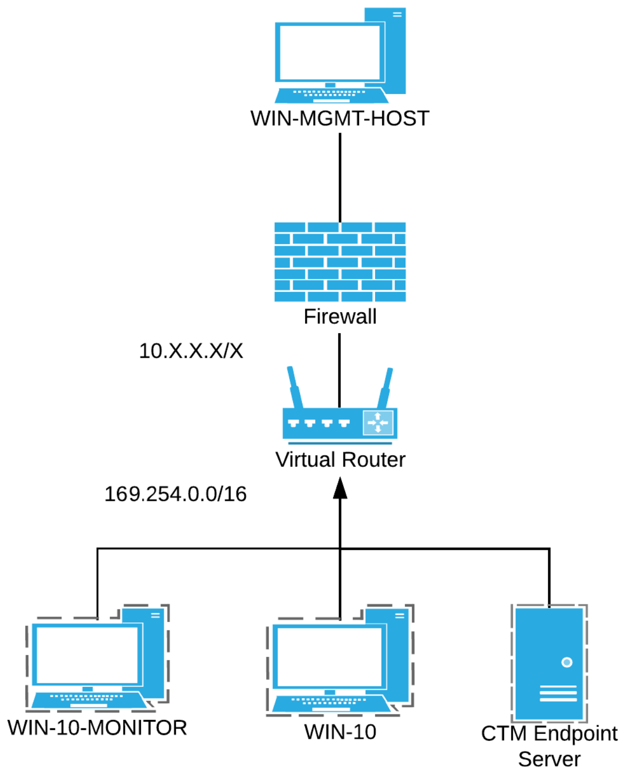


Fig. 13: Virtual network setup

The host (MGMT-HOST-PC) is the computer which runs VirtualBox with three virtual machines in an internal network [43]. The machines on the internal network are not connected to the host in any way, which reduces the risk of a ransomware outbreak over the network to a minimum.

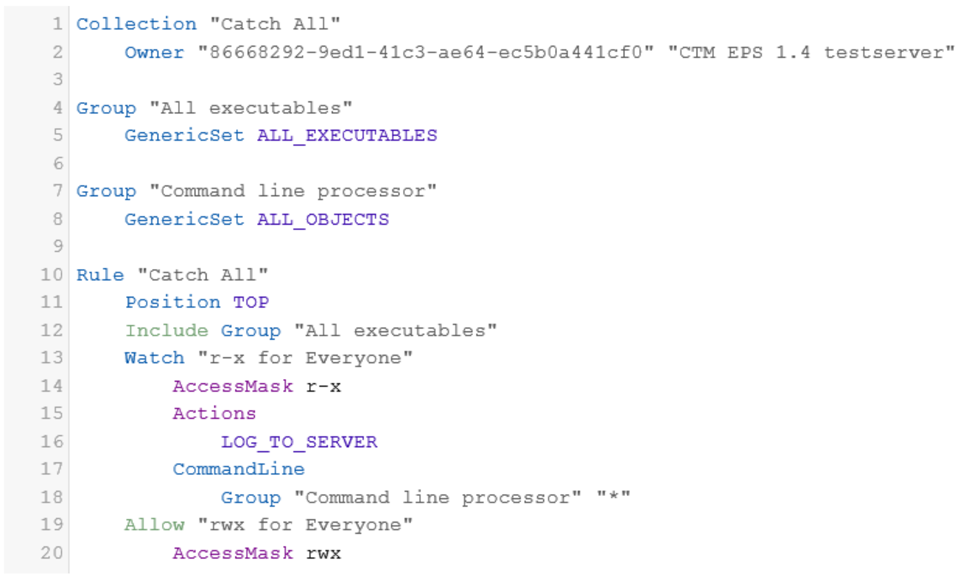
Two virtual machines are configured as endpoints running Windows 10. One of these endpoints is used for monitoring (WIN-10-MONITOR) to write the log files to, and the other virtual machine (WIN-10) will run the ransomware samples to get infected. This endpoint is not patched, has its firewall turned off and Windows Defender disabled in order to make sure the ransomware could be executed without being blocked by Windows.

The monitoring VM is configured with the latest security updates in order to reduce the risk of an infection over the network.

Both Windows 10 endpoints are connected to the third virtual machine, which is the CTM Endpoint Server. This server controls the rule set deployed on its clients and stores the alerts generated by triggered rule events. This server is running on Debian Jessie and is configured with a firewall.

Both Windows endpoints are connected to the server with the CTM Endpoint Module Agent. The monitoring VM is configured with the default rule set to block any known malicious source while the other Windows 10 endpoint, which runs the ransomware, is configured with a rule set which will only monitor system processes.

In order to only log system processes a custom rule has been written for the CTM Endpoint Module, which is shown in Figure 14.



```
Collection "Catch All"
    Owner "86668292-9ed1-41c3-ae64-ec5b0a441cf0" "CTM EPS 1.4 testserver"

Group "All executables"
    GenericSet ALL_EXECUTABLES

Group "Command line processor"
    GenericSet ALL_OBJECTS

Rule "Catch All"
    Position TOP
    Include Group "All executables"
    Watch "r-x for Everyone"
        AccessMask r-x
        Actions
            LOG_TO_SERVER
        CommandLine
            Group "Command line processor" "*"
    Allow "rwx for Everyone"
        AccessMask rwx
```

Fig. 14: CTM Endpoint Server monitoring rule

G.2 Endpoint Server

The CTM Endpoint Server logs system events when an event matches a rule specified in the configuration of the CTM Endpoint Server. The global design of a rule collection is defined by the following structure as shown in Figure 15.

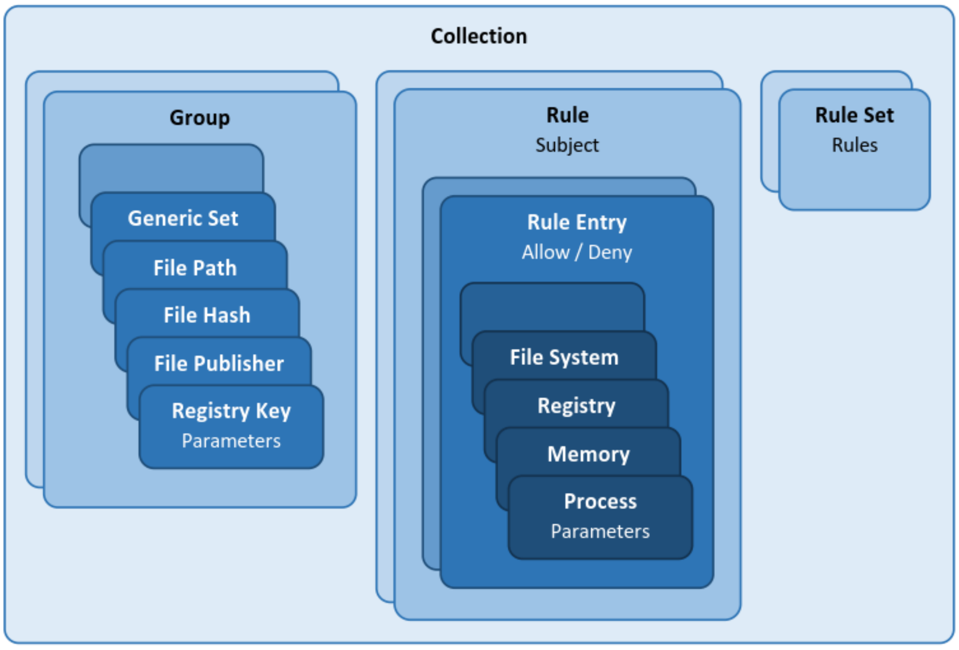


Fig. 15: CTM Endpoint Server rule set

A custom collection is written to filter all Process Create events on the connected endpoints. As the only requirement is to filter out these type of events, the rule syntax is relatively simple. A *GenericSet* Group used to define a group of executables and a group for any object. The *Rule Entry* is set to *Watch*, a feature that is not mentioned in Figure 15 as it is primarily used for testing purposes. The behavior of the *Watch* node is documented in the Rule Manual [44].

The rule that has been written using the rule syntax for the CTM Endpoint Module is shown in figure 3. This rule filters all process creation events and writes them to the database.

The rule entry filters events by checking the *CommandLine* arguments of a process event for any input in combination with the *AccessMask* value set to read, execute. This results in all process creation events to trigger the rule. The action *LOG_TO_SERVER* defines that an event matching the rule will be logged to the server.

G.3 Combining logs

Unfortunately, multiple logging methods were required because they both provided different information about features. The CTM Endpoint Module of Fox-IT provided a lot of information about the single processes on a system, but because of the current implementation of the logging it does not contain process IDs (PIDs) of child processes. Without these PIDs it is not possible to create a process tree.

In order to obtain this missing information the ProcessMonitor logs are used, which do store the PIDs of the child processes. These logs can be matched based on their name, parent process ID and command line arguments. When a match is found the informative features are merged to establish a final process tree.