RP2 research paper

Malicious behavior detection based on CyberArk PAS logs through string matching and genetic neural networks

July 5, 2020

Supervisors:
Deloitte: Roel Bierens and Bartosz Czaszynski

Ivar Slotboom
MSc Security and Network Engineering
University of Amsterdam
ivar.slotboom@os3.nl

Mike Slotboom
MSc Security and Network Engineering
University of Amsterdam
mike.slotboom@os3.nl

***Abstract*—CyberArk Privileged Access Security (PAS) is a Privileged Access Management solution, which makes sure the right access is being granted for each user, while monitoring and logging user behavior. The Privileged Threat Analytics (PTA) component of CyberArk generates security alerts based on manually defined security configurations. This component processes logs, but is limited in functionality as it takes samples of the logs generated during user sessions. The result of this is that PTA disregards additional logs generated in the PAS system.**

To extend the malicious behavior detection capabilities, this research defined 18 use cases based on a captured data set. A total of 26 attack techniques were performed in a test environment which were based on the MITRE ATT&CK Enterprise Matrix. These use cases are defined as a filter in Splunk by using a string matching approach to reliably filter out logs resulting from malicious events. Because this approach is based on known and manually defined models, it is prone to human error and does not detect incidents outside of its scope.

In order to solve this issue of reliably detecting malicious events, machine learning was applied based on genetically training a neural network using the Genann library. The bag of words approach was used to convert a single log entry into a series of frequencies based on each field entry in the log. Two applications of machine learning have been applied in a Proof of Concept (PoC): 1: Detector, which separates malicious logs from normal behavior logs; and 2: Classifier, which matches malicious logs with an attack type (i.e. use case) that was previously defined.

Experiments were conducted to define optimal parameters used in the neural network. These parameters are for both the Detector and Classifier: Four hidden layers, 20 nodes per hidden layer, a classification threshold of 0.5. In the Detector roughly 99% of the distinguishable malicious logs could be successfully identified as malicious, resulting in a True Positive Ratio (TPR) of 98.95% and a False Positive Ratio (FPR) of 8.01%. The classifier was setup differently by matching a single malicious log to a model, one match (True Positive) resulted in 16 negative results for the other models (True Negative). This classifier scored with 34.19% TPR and 1.38% FPR, which is still a significant result.

Keywords*—CyberArk, malicious behavior, string matching, machine learning, genetic neural network, bag of words

I. Introduction

Privileged Access Management is an important topic and endless discussion in information security of modern IT environments. Several solutions are on the market to make sure the right access is being granted for each user, while monitoring and logging their behavior. CyberArk Privileged Access Security (PAS) is one of the proprietary solutions. CyberArk PAS uses a vault to control privileged sessions and credentials on host systems. A user can use PAS to find passwords and connect to the correlating systems in the environment via an intermediate session. This session is fully recorded in the PAS system for both automatic and manual review. Privileged Threat Analysis (PTA) is a component that monitors the use of privileged access in the sessions based on logs from the vault and generates threat analysis logs and alerts out of this information[4]. However, this PTA has limited capabilities of detecting all malicious behavior in the CyberArk PAS system.

This research applies selected attack techniques on CyberArk PAS in the test environment and investigates the generated logs to detect malicious behavior. These logs are used to determine what type of attack has been performed (i.e. use cases).

II. Background

A. CyberArk Architecture

In this research, CyberArk PAS solution version 11.4.0000 was used. This solution is built up from six main components[4]. CyberArk provides a controlled environment, where (recorded) privileged sessions can be started using the Password Vault Web Access (PVWA).

1) Password Vault: The Password Vault is the center of the solution. In this component, the data is stored in separate software safes (e.g. Linux Accounts, Windows Accounts, SSH Keys) and user actions are logged.

2) Password Vault Web Access: The PVWA is a web portal to use the contents of the Password Vault using a browser. Based on the assigned roles per user, this portal can be used to log in to start and stop privileged sessions in order to watch security events and editing security configurations.

3) Central Policy Manager: Op top of the Password Vault, password management is done via the Central Policy Manager (CPM) component. In the CPM, policies can be defined and enforced to generate, change and verify passwords. This CPM connects to the managed devices to make automatic changes.

4) Privileged Session Manager: The privileged sessions that a user starts are maintained by the Privileged Session Manager (PSM). This component can enforce privileged access by connecting to the device via an intermediate device. The user therefore logs in into the session of the PSM and the PSM starts a session on the device. Within CyberArk PAS, two PSM variants are available: a default Windows PSM for graphical Windows sessions and a SSH proxy for connecting to hosts with SSH.

5) Privileged Threat Analytics: The Privileged Threat Analytics (PTA) component of CyberArk generates security alerts depending on the security configuration. PTA makes it possible to generate an alert based on detected behavior in privileged sessions and irregularities based on user behavior. For example, it is possible to detect Golden Ticket or Pass The Hash attacks when the additional sensors are in place.

6) PrivateArk Client: In order to administer the Password Vault, the PrivateArk Client can be used. This is an application that connects to the Vault and makes it possible for a privileged user to make changes to the users and vaults as well as viewing the contents of the vaults inside.

B. MITRE ATT&CK Matrix

The MITRE Corporation has published the MITRE ATT&CK Enterprise Matrix, which shows techniques and tactics that are used to attack systems[10]. The attacks are divided into categories (e.g. Execution, Persistence and Privilege Escalation) and the framework functions as a reference for defense against attacks. This framework will be used to define the use cases, where the attacks that could be reproduced will be performed.

C. Splunk

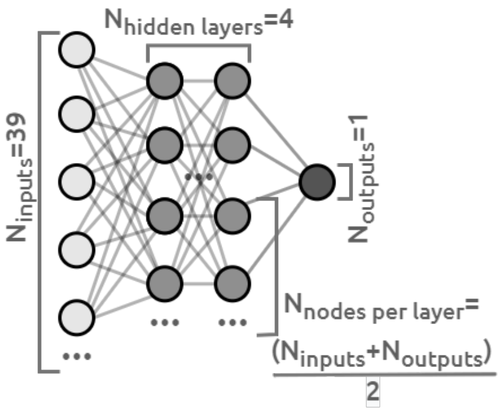
Splunk is an analytics solution to collect end-point data and summarize them through visualization and recognizing behavioral patterns[14]. This can be used with CyberArk PAS and/or PTA to correlate logs and pool together urgent threats, potential risks and other forms of threats. This overview can provide administrators or auditors the right start how to respond, given that notifications can be setup in Splunk[14].

In this research, use cases were defined from CyberArk PAS logs to learn from historical events in order to mitigate future incidents. A use case is defined as a filter that captures the logs resulting from a suspicious event. Examples of use cases are queries in Splunk to determine patterns from a chosen attack, which are listed in Appendix A.

D. Genetic Neural Networks

A neural network works through several layers in order to produce an output. Its layers and their respective properties are depicted in Figure 1 and are the following:

- Input values (i.e. N_{inputs}) that are inserted in the neural network, in our case every field entry of a log.
- Hidden layers (i.e. $N_{\text{hidden layers}}$) that allows abstraction and capturing relationships between the inputs and the hidden layers, which consist out of nodes.
- Hidden layer nodes (i.e. $N_{\text{nodes per layer}}$) which all have a value (that are sampled by an activation function), process the input and mutate its value to get the best output.
- Output value (i.e. N_{outputs}) is the result of the neural network.



$N_{\text{hidden layers}}=4$
$N_{\text{inputs}}=39$
$N_{\text{outputs}}=1$
$N_{\text{nodes per layer}}=\frac{(N_{\text{inputs}}+N_{\text{outputs}})}{2}$

Fig. 1. Genetic Neural Network Default Setup

Training neural networks using genetic algorithm for model optimization is a concept promoting both a brute-force approach through mutation of the previous network[12]. The process shown in Figure 2 starts by creating a large pool of networks and starts assigning random weights to the nodes during the initialization phase. Every network in this pool is a slight mutation of one of the best scoring networks from the previous generation if this generation is available. A mutation is the change of a weight in the neural network and this can be done randomly or controlled. A change in the weight indicates a reset of the weight value to a random number between -1.0 and +1.0. The result of this can be minimal or significant, depending on the impact of the weight on the output. In genetic neural networks, these mutations are done randomly. After the initialization process is complete, every network in the pool gets the same classification challenge and classifies the log entries according to their newly constructed network model.

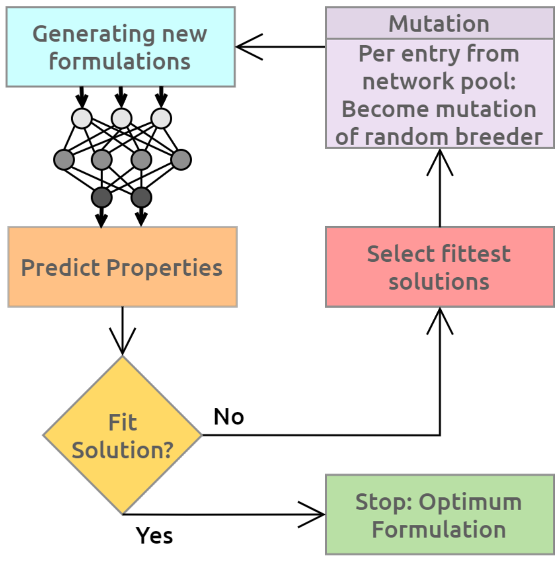

Generating new formulations
Mutation
Per entry from network pool: Become mutation of random breeder
Predict Properties
Select fittest solutions
Fit Solution?
No
Yes
Stop: Optimum Formulation

Fig. 2. Genetic Neural Network Process (based on [12])

As soon as every network has solved their challenge, a score is calculated based on how well each network performed (i.e. ”fitness”). From here on, only a small number of the top scoring networks (e.g. 16 out of the 512 pool) will be used for the breeding process of the next generation (which are now called ”breeders”). This 3% of the pool allows the best scoring networks to be chosen, but also allow a variety of classification models to be considered in the next round of the mutation[15]. This fills up the pool with new mutations based on these top scoring networks and the training process can repeat itself. The process stops when the solution fits, resulting in the optimum results.

III. Problem Description

While CyberArk PTA has the ability to identify and detect malicious behaviour in user sessions, PTA has less capabilities to identify misuse within the other PAS components such as the PVWA, which could indicate people circumventing the CyberArk PAS solution. On top of that, in order to manage the load of the incoming syslog messages, the PTA samples the data. This can result in attackers misusing systems without being able to trace the activities via the CyberArk PAS system.

PTA uses a configuration that has to be assigned manually, which means that malicious behavior can go unnoticed if the security engineer does not take such behavior into account. Furthermore, The output of events generated by CyberArk PTA do not carry useful information which can be used to make actionable decisions on incident response. This does not help in monitoring the malicious events and in correlating the alerts. As a result of the lack of proper analytical capabilities and the amount of generated log data, the actions taken can be insufficient, or be executed with a significant delay.

This research is about detecting future malicious incidents by filtering out this behavior from the CyberArk PAS logs. User interaction will be logged to detect malicious behavior, regardless of intent, based on use cases and to take appropriate actions. Automation techniques (e.g. machine learning) are utilized to detect malicious behavior based on previous experiences.

IV. Research Questions

This research answers a main research question, which is stated as follows:

How can one recognize malicious behavior based on the logs from CyberArk PAS in both the present and future?

A. *Sub research questions*

There are two sub research questions to assist the main research question. These questions are:

1) *Which use cases can be defined for Privileged Access Management to distinguish malicious behavior?*
2) *How can future incidents be detected by using previously researched behavior from the CyberArk PAS logs?*

V. Related Work

The related work is mostly based on applicable research in log data and correlation analysis.

Another research about log correlation has been carried out by Abad et. al[1] in 2003. While this research is dated, it determines the approach of anomaly detection in Intrusion Detection Systems (IDSs) and the actions that need to be taken to control a possible attack. Two types of approaches are proposed: top-down from attacks to logs or bottom-up from logs to attacks. Although this research was not focused on Privileged Access Management (PAM), the top-down approach is still relevant to apply in this research project, since this makes it possible to link logs to a specific predefined attack. Combining both approaches will give a more complete view of the actual events happening, but is more time-intensive and requires additional research because of their unique data sets.

Meera and Geethakumari[9] performed a research in 2016 about event correlation in cloud environments for forensic purposes. This research was focused on analyzing cloud API logs, but proposed a method about how to setup a correlation algorithm to create the right events out of the available API log information. This research proposed a correlation algorithm to define correlation criteria to match and filter out events based on predefined atomic conditions. This approach is useful for this research project, as the logs have to be parsed and correlated to identify (a series of) malicious events.

Huizinga[7] investigated in 2019 the use of machine learning to analyze the network traffic produced during a penetration test. This study used a machine learning model and trained it based on known inputs and outputs using supervised learning. This technique of knowing the inputs and outputs could be used to train a neural network to recognize malicious user behavior in order to predict similar reoccurring behavior in the future. The inputs can be gathered from the CyberArk PAS and PTA logs and the outputs can be the level of severity of the user's actions caused by their behavior.

In January 2020, Landauer et. al[8] published a survey about several types of log clustering approaches (e.g. Code analysis and neural networks) and how they are used to analyze log data manually or dynamically. This survey is useful when examining the techniques used to generate insights out of the CyberArk data. For instance, it is important that logs are parsable in a machine-readable format, since human-readable formats could contain repeated messages in different wording, as well as artifacts such as line numbers. Furthermore, log files are suited for dynamic clustering which allows for the allocation of sequences (based on log lines) that could be identified as patterns. Assuming patterns can be identified as repeated behavior, one could identify suspicious behavior if an action does not follow a certain pattern.

VI. Method

A. Test setup

The test setup was primarily aimed at identifying malicious behavior, regardless of intent. In Figure 3 the components of the test setup are depicted. A CyberArk Proof of Value (PoV) environment was used to execute attacks on. This environment was a working cluster with all of the main components of CyberArk, as mentioned in Paragraph II-A (i.e. Password Vault, PVWA, CPM, PSM, PTA and PrivateArk Client). Next to the PoV environment, two servers were used to capture syslog data from the Vault in the PoV environment using Splunk and to perform log analytics on this captured data.

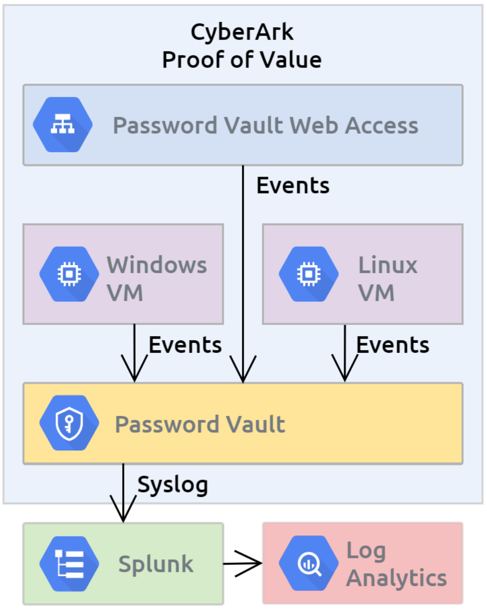


Fig. 3. Research Project Test Setup

B. Approach

The approach of this research project is divided into steps, which are depicted in Figure 4.

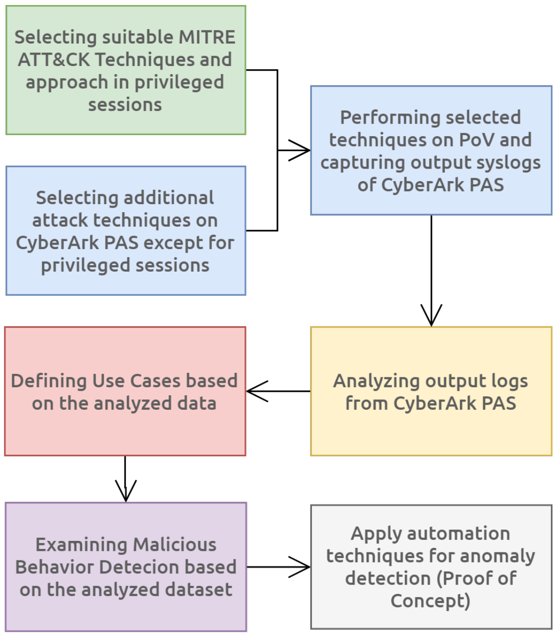

Selecting suitable MITRE ATT&CK Techniques and approach in privileged sessions
Selecting additional attack techniques on CyberArk PAS except for privileged sessions
Performing selected techniques on PoV and capturing output syslogs of CyberArk PAS
Analyzing output logs from CyberArk PAS
Defining Use Cases based on the analyzed data
Examining Malicious Behavior Detecion based on the analyzed dataset
Apply automation techniques for anomaly detection (Proof of Concept)

Fig. 4. Research Project Approach

1) Attack selection: The attacks in the MITRE ATT&CK Enterprise Matrix were selected in Appendix A to be performed in the test setup. Additionally, attacks based on the MITRE ATT&CK Enterprise Matrix were defined and specifically applied to the CyberArk PAS system (e.g. attack scenarios based on CyberArk PVWA and Password Vault).

2) Generating logs and analysis: To investigate how malicious behavior is visible in the generated syslog messages, a top-down approach was used to generate log data in the test setup, where predefined attack techniques were selected and performed. In the test setup, the attacks were executed three times on both Windows (i.e. Windows Server 2016) and Linux (i.e. Red Hat Enterprise Linux 7) if applicable and on the CyberArk PAS system (i.e. additional attack techniques). The syslog messages were formatted and captured in Splunk during the attacks to make sure that per attack technique the logs can be stored. The attacks were done multiple times and applying these techniques resulted in logs that formed footprints of specific attacks. The logs follow a predefined format using a key-value pair, which is visible in Table I.

TABLE I
CyberArk PAS Log Format

```
[month, day, time, ip_address, unknown, timestamp,
    host_name, format, platform, application,
    application_version, event_id, event_message,
    event_level, act, source_user, system_name,
    device, source_host, destination_host,
    destination_user, session_id, app, reason, 15
    _additional_attributes_used_based_on_event]
```

Log data is composed of a timestamp and IP Address as well as which activity has been logged. The activities are grouped in events and the ”reason” or the message (i.e. ”msg”) are used to indicate what the event is about. The captured data was sanitized to homogenize the data and to ease analysis as some data was improperly formatted. A total unbalanced data set of 5300 logs was used in this research, which was split in the three mentioned groups:

- 2272 Normal behavior logs (**“N”**): Normal behavior was captured, where commands of Linux and Windows System Administrators were simulated and recorded to compare this with the attack captures. This simulation consisted of commands and GUI actions, but were not exhaustive as this is limited by the capabilities of the test setup. Additionally, when the system is running idle without performing any attacks, the CyberArk PAS solution generates logs (e.g. verifying and updating policies). These events were captured for five hours in total.
- 2648 Suspicious logs (**“S”**): The ”idle normal behavior” was used in an analysis script to filter out normal behavior from the log captures of the attack techniques, where suspicious logs including malicious logs remain. By filtering out the malicious logs, the suspicious logs formed a data set that consisted of logs around when the attack happened. These logs show similar content as the malicious logs (e.g. ”date”, ”hostname”, ”source_user”), which makes it hard to differentiate.
- 380 Pure malicious logs (**“M”**): Logs from the attack techniques that show definitive malicious behavior.

As can be seen from this data set, the amount of pure malicious logs is significantly less (i.e. 14.3% compared to the normal and suspicious logs). This shows the problem that needs to be solved: finding a malicious log in a bulk of normal logs, hence the name ”unbalanced” data set.

3) Defining use cases: The malicious logs were analyzed further to identify which information serves as the best descriptor of an attack. The results of this process have been written down in Appendix B. When the logs were not showing the characteristics of an attack or it is not distinguishable from normal behavior (i.e. open shell), it was not possible to use the descriptors for this particular attack. After this filtering was done, search queries were build based on the descriptors which can be observed in Appendix C.

a) String Matching Malicious Behavior Detection Architecture: Based on the use cases, a system can be assembled that matches the use cases and triggers alerts based on the malicious events. In this research, we introduce a framework for a model matcher based on string matching. This framework is illustrated in Figure 5 to match the logs with predefined models.

The sanitized logs are matched with known strings in the model matcher, which is being fed with models from the model manager. The models are a set of predefined use cases based on queries (e.g. the defined use cases in Appendix C). For each attack technique, one model is being defined (e.g. a search query in Splunk configured as a Splunk Alert).

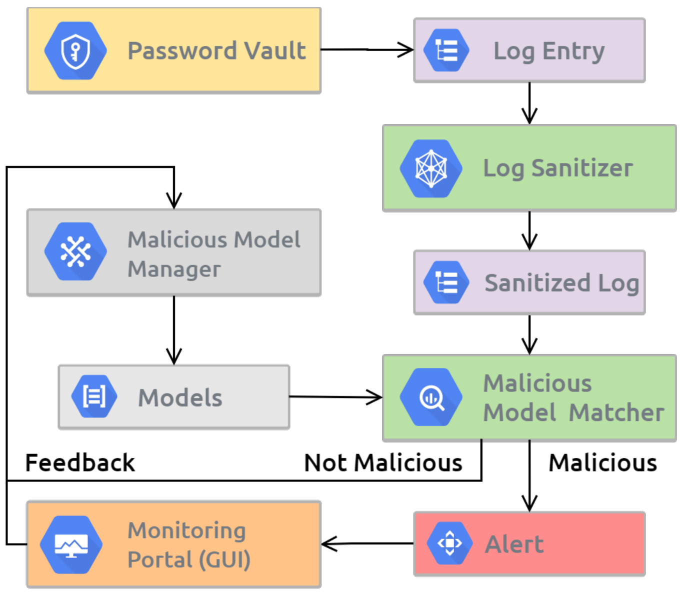


Fig. 5. String Matching Malicious Behavior Detector Framework

When a log matches with a model, the string matcher generates an ”Alert” event in the graphical user interface (i.e. monitoring portal), which is used by the IT Security Team members. Based on this event, actions can be taken, e.g.: restrict access, investigate further or relate to earlier events. This process could also be automated to react immediately upon an event. By storing previous events, this information can be added to the alert to enrich the data and to detect patterns based on attributes (e.g. user, application and host).

The event is being forwarded as feedback to the model manager. The feedback indicates if the model has to be adjusted and could be logged for events in the future. It is also possible to define new models in the model manager. The model matcher as well as the GUI are universal concepts, where in this research Splunk is used. This means that the model matcher as well as the GUI can be replaced with any kind of application or framework, given that they serve the same purpose.

Note that this approach is static as it only uses defined models and that it requires manual interaction to function properly. This means that there should be a security team available full-time to act on the security event, which could introduce delays, scalability issues and human error. An automatic approach is needed to adapt to the rate of the logs and events coming in.

4) Automatic Behavior Detection: Automation can be used to identify malicious behavior naturally by comparing anomalous behavior with normal behavior. In order to automate the process of detecting malicious behavior as well as making it adapt to future malicious incidents, another approach is needed. In this research, the application of machine learning to solve this problem is investigated. Genetic neural networks are trained to alert upon detecting malicious activity.

a) Automatic Behavior Detection Architecture: In Figure 6, we introduce a architectural framework to apply machine learning. First, the neural network needs to be trained (i.e. Malicious Activity Detection Trainer) with a data set based on malicious and non-malicious events, which should be updated periodically. As soon as the trained has finished training, its best neural network processes incoming log messages directly (i.e. Live Scanner). This approach of splitting the trainer from the scanner is chosen, because of scalability and the possibility to train from future incidents. When the trainer and scanner are the same, it gets harder for the system to cope with classifying loads of incoming log messages as well as training from this.

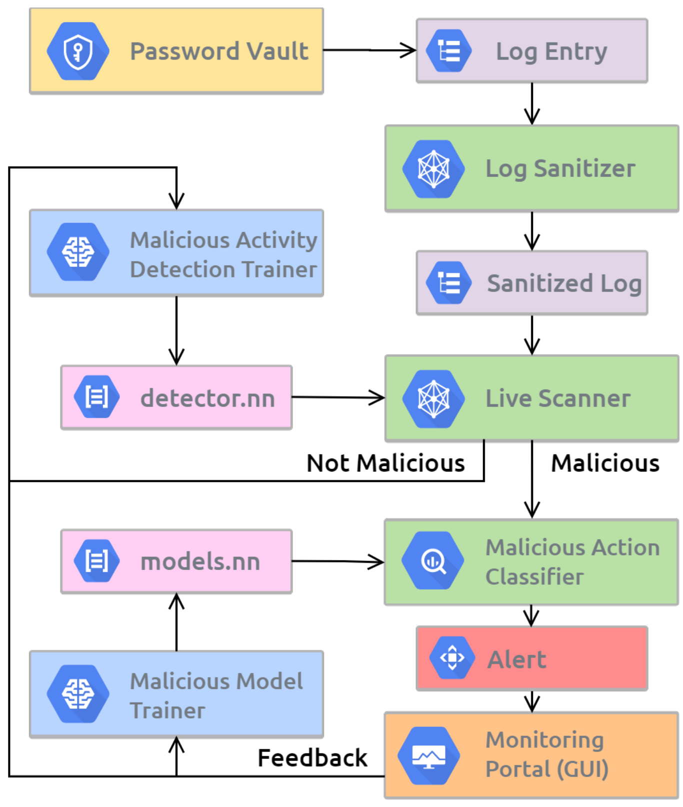


Fig. 6. Dynamic Malicious Behavior Detector Framework

When the sanitized log is found to be malicious, it is forwarded to the Malicious Action Classifier, which tries to classify which attack technique is being used from the predefined MITRE attacks techniques listed in Appendix III. These techniques are based on the defined use cases, which are trained in a separate neural network (i.e. Malicious Model Trainer). The classifier is also split from the model trainer for the same reasons as the Live Scanner: scalability and the possibility to train from future incidents.

5) Application of genetic neural networks:

a) The Genann library: For this research, the Genan libraryn[16] has been used to create and train the neural networks. The benefits of using Genann are as follows:

1) The library is minimal; only two files need to be added into the project and the code complexity is low, since it is focused on the core functionality of training a neural network.
2) The library supports parallelization, meaning that multiple networks could be trained at the same time in order to decrease the amount of time required to train a pool of networks.
3) The library is open-source and easily modifiable, meaning that additional functionality could be added in if this research requires it.

The default activation function from the Genann library is the Sigmoid function. This activation function requires a vast amount of clock cycles which causes a long training time. In order to optimize the calculation process, the Rectified Linear Units (ReLU) activation function has been implemented. According to our test that was run on the same CPU as that was used for the training, the ReLU activation function performed roughly 7.5 times as fast as the general Sigmoid function. The test can be observed in Appendix D.

The main benefits from the ReLU activation function over using a Sigmoid are that gradients are less likely to vanish when values are > 0, which is good for constant learning. A loss in gradients is an issue in machine learning where the network model will lose precision during its classification process. This can make it harder for a model to be trained where results require a lot of accuracy. Secondly, ReLU is more sparse when the node has a value of ≤ 0, which is good for dense representations, providing better classification results.

b) Neural network inputs: The training of the neural network was based on the data set that was produced to define the use cases, including normal behavior and the attack techniques. Since the amount of fields in a CyberArk log is the same (i.e. 39), every field can be used as an input for the network. The field entries can be references back to Table I.

The benefit of the constant amount of input fields is that one can create a dictionary of known field values. This technique is also commonly called feature extraction using a ”bag of words”[6][11]. By building this dictionary (the bag of words), one can identify the Term Frequency (TF) and the Inverse Document Frequency (IDF) of how often certain field values appear in the data set. The Term Frequency entails how often a word occurs in a document, whereas the Inverse Document Frequency entails the inverse function of the amount of documents in which the word occurs. By combining these, one can specify how common or rare a specific word is. If a word is common, the TF-IDF (calculated as $TF \cdot IDF$) will be close to 0. If the word is rare or does not exist, the frequency score will be (close to) 1. This allows the network model to find anomalies in conjunction with the other network inputs.

This model had to be adapted for our application in three ways, which were:

1) The previously mentioned documents are now considered the incoming log entries;
2) Every field value will now have their own dictionary; and

3) A bag of words will now be a bag of phrases, since fields like "timestamp" and "reason" can contain values that are not words.

This means that when a log entry comes in, every log entry will be split up into the values of the keys and each values will calculate their TF-IDF. The values of these TF-IDFs will then be provided as inputs for the neural network.

c) Neural network training process: To further improve upon the use of machine learning onto the detection of either malicious behavior or attack techniques, k-fold cross validation[2] was implemented. This validation technique is used as a statistical method to estimate the fitness of a neural network. This has been implemented by first randomizing the sanitized logs and then separating it into two sets: 75% training set and 25% validation set. This approach uses a sliding window to randomize both the training and validation sets by making sure each entry is one time available in the validation set [12]. By doing so, the training technique of a network does not become biased by validating itself on separate log data.

By applying this 4-fold cross validation approach, the neural network will be modeling itself in varying conditions, where there could be different ratios between malicious and normal behavior. The neural network will start modeling based on the training set and will measure its performance on the validation sets. This decreases how biased the training of the model (i.e. the model performs excellent on the training set, but performs poorly on new and unseen data) is [3].

d) Neural network outputs - general overview: The outputs determine the classification of the input sample: either the log entry is malicious or it is not. The result is a number between 0 (i.e. normal behavior) and 1 (malicious behavior), which indicates the network's confidence in its conclusion. Based on the environment, the threshold from where the log will be marked as malicious can vary. To evenly distribute this value, the results < 0.5 are marked as normal behavior and ≥ 0.5 are marked as malicious by default in this research. The conclusion can then be used to determine how well the network is performing, which is where the fitness scoring system comes in.

During the validation of the neural network, four indicators that are based on the predicted and actual outcomes were defined as follows:

- **True Positive (TP)**: Malicious entry is successfully classified as malicious
- **True Negative (TN)**: Normal behavior entry is successfully classified as not malicious
- **False Positive (FP)**: Normal behavior entry is misclassified as malicious behavior
- **False Negative (FN)**: Malicious entry is misclassified as normal behavior

During the training of the network, the networks have their own indicator counters to determine its performance. The direct outputs of the training are the four mentioned indicators and these values are used to calculate how well it is performing regarding the training set and validation set. Malicious behavior detection is based on finding some abnormalities in a bulk of data (i.e. unbalanced data set). For a classifier is both easy and naive to flag all data as normal behavior, since that will results in an accuracy of over 90%. Therefore, it is crucial that a scoring system is used to prevent the discrimination against this unbalanced data set.

Therefore, additional calculation needs to be done to measure the correctly estimate performance of the model. The four indicators are used to calculate the F1 Score[13]. The F1 Score relies on the Precision and Recall scoring mechanisms that also rely on the four indicators.

The Precision is measuring how many instances correctly classified by the model were relevant instances (TPs). [13]. It is calculated by the ratio between the TP (i.e. how many results were actual TP) and the total of how many results are marked as positive (i.e. TP and FP):

$$Precision = \frac{TP}{TP + FP}$$

The Recall is another ratio about ratio between correctly classified instances and all instances which should belong to malicious class (i.e. TP and FN):

$$Recall = \frac{TP}{TP + FN}$$

By combining the Precision and Recall, the F1 score can be calculated. This score is a balance between what is predicted to be TP and what actually is TP [13]. This prevents discrimination on the quantity of the total malicious and non-malicious entries, which helps to properly classify them as either malicious or normal behavior while working with an unbalanced training set. The F1 score is defined as follows:

$$F1\ score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

The neural network needs indicators to train on. The F1 score is used to measure model performance, but this is not enough to improve on previous results. As the results of an entry is between 0 and 1, this can also be used to visualize how accurate the answer was that was given by the neural network. That is why we introduced an error margin score, which we called: "Delta score". This score indicates how well the neural network classifies a malicious entry as malicious and does the same for non-malicious data. To calculate this score, an algorithm was being followed as described in Algorithm 1.

The Delta score can be summarized as:

$$Deltascore = 1 - \frac{TotalErrorDelta}{TotalErrorDeltaSamples}$$

When the F1 score is combined with the Delta score, the performance of the neural network training could be measured. As the Delta score is being introduced in this research, another score is introduced to find a balance between the two scores. This balance score we called the "Training score' and is calculated as follows:

$$Trainingscore = \frac{F1score + Deltascore}{2}$$

Algorithm 1: Delta Score Calculation

```
Result: Delta Score
errorTotal = 0.0;
for trainingEntry in trainingSet do
    // trainingEntry.desiredResult is
    // either 1.0 or 0.0
    if trainingEntry.desiredResult == 0.0 then
        errorTotal += output;
    else
        errorTotal += 1.0 - output;
    end
end
deltaScore = 1.0 - (errorTotal / trainingSet.size());
```

By combining the F1 score with the Delta score, the Training score is motivated by the gains in both regards. The Delta score ensures that the confidence in the output is more accurate, whereas the F1 score ensures that the classification is correct.

6) Performance Experiments: In order to train the genetic neural network with maximized performance, the parameters of the network should be chosen accordingly. To start testing, different setups were used, where we defined one setup as reference setup. These test were run on a Dell PowerEdge R240 Ubuntu 18.04.4 LTS server with an Intel(R) Xeon(R) E-2124 CPU @ 3.30GHz, 16GB of memory (i.e. 2x Samsung M391A1K43BB2-CTD) and Samsung SSD 860 hard disk. This setup gained a correctness value of 90% in 5 minutes.

In Figure 1, this reference setup is shown, where the number of hidden layers (i.e. $N_{\text{hidden layers}}$) is 4 and the number of nodes per layer (i.e. $N_{\text{nodes per layer}}$) is 20. The input values (i.e. N_{inputs}) are based on the rarity of the log message field, which consists of 39 attributes. The N_{outputs} is 1, as this is the value given back by the neural network if the log entry is malicious or not.

We have defined five experiments in Table II to get to the optimum parameters and to verify the performance of these experiments on the detector component as well as the classifier component from Figure 6. Experiment A was only run on the detector neural network, as the classifier had one data set.

There are more variables that can be adjusted in the setup, but in order to test, we set fixed variables based on the default setup. We used 16 iterations for the detector, where one iteration is a run where every log entry has been once in the validation set. The number of iterations for the classifier is set to 1024 as the data set is smaller. In this setup, only the malicious logs and the total logs per technique are used, which is a data set of 2272 entries. Per iteration, a network pool of 512 is initialized, where the top 16 networks mutate with a maximum of 10 weights. This restricts randomness of the mutation, but provided a solid learning curve in the default setup.

In order to make these results visible, a Receiver Operating Characteristic (ROC) curve has been produced out of these results [5]. This curve uses the True Positive Ratio (TPR) (i.e. the Y-axis) and the False Positive Ratio (FPR) (i.e. the X-axis) to indicate how well the machine learning performs using the different parameters from the experiments. The most ideal scenario has a TPR value of 1 and a FPR value of 0 (top-left corner in the graph), meaning that none of the TPs are missed when classifying the malicious logs and that all normal behavior is being classified as such. When calculating the TPR and FPR per parameter set in the experiments and plotting it as ROC curve, the used parameters resulting in the most top-left point in the graph are considered the optimal parameters.

TABLE II
EXPERIMENTS FOR GENETIC NEURAL NETWORK TRAINING IN TEST SETUP

ID	Experiment Title	Description
A	Using different training sets (only detector)	During the capturing of the data, the malicious events (ie. ”M”) were exfiltrated from suspicious logs. Using the remaining part of the suspicious logs (i.e. ”S”) and classifying this as normal behavior increases the normal behavior data set (i.e. ”N”). This experiment was built to compare this extended data set (i.e. ”N+S” and ”M”) with the original data set (i.e. ”N” and ”M”).
B	Using a different number of hidden layers	The neural network can be made more complex by adjusting the amount of hidden layers. In this experiment, the optimal number of hidden layers was searched for. The values we used here are: 1, 2, 4, 8, 12 (detector only) and 16 (classifier only).
C	Using a different number of nodes per hidden layer	The amount of nodes influences the amount of data can be stored per hidden layer. This parameter is related to the amount of hidden layers and is tested in this experiment. The values we used here are: 10, 20 and 40
D	Using different classification thresholds	In the default setup, the classification threshold is set to 0.5. In this experiment, different thresholds are used: 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 and 1.0.
E	Using optimal parameters from previous experiments to test performance	To verify the optimal parameters, this experiment combines the best results of the four experiments.

In the graph, a reference line runs linearly from TPR 0 and FPR 0 to TPR 1 and FPR 1. If the classifier is around this line, the classifier is not useful as it is characterized by the same probability of classifying correctly, as classifying incorrectly. The more the classifier is away from this line, the lower the chance is that the classification is incorrectly.

7) Source code: The source code of this research has been published on this `GitHub page`.

VII. Results

A. Use cases to detect malicious behavior

The attack techniques based on the MITRE ATT&CK Enterprise Matrix, were performed in the test setup and the syslog messages were captured in Splunk. The syslog messages were analyzed and use cases were defined based on the descriptors of the attack. In Appendix C, the use cases are summed up.

The use cases were validated using the Splunk queries of a live CyberArk system at a foreign bank.

B. Malicious behavior detection using genetic neural networks

In Appendix E, the results of the experiments from the detector are shown. To visualize this data, the ROC curve in Figure 7 is produced. Based on this curve, the default setup (i.e. dataset "N and M", 4 hidden layers, 20 hidden layer nodes and 0.5 classification threshold) turned out to be the optimal parameters to use in this neural network (i.e. Experiment E).

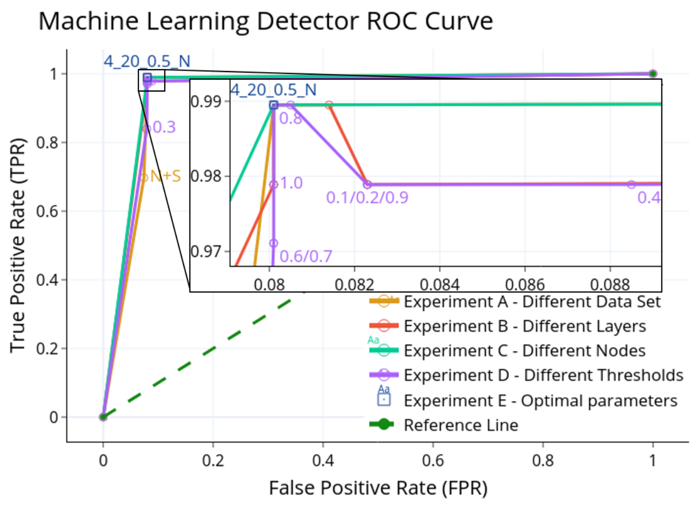


Fig. 7. ROC Curve - Machine Learning Detector

C. Malicious behavior classification using genetic neural networks

The result of the experiments on the classifier are shown in the ROC curve in Figure 8. Also in this curve, the default parameters (i.e. 4 hidden layers, 20 hidden layer nodes and 0.5 classification threshold) were the optimal parameters to use. The full results of the experiments are written down in Appendix F.

D. Comparison of the F1 score to the F1 score with the Delta score

In order to compare the benefit of the use of the Delta score, which was introduced to improve the confidence of the result an additional test was run using the optimal parameters. This test as, shown in Figure 9, validated the underlying effect of the combination of the F1 score separately from the Delta score, as well as the F1 score in collaboration with the Delta score.

VIII. Discussion

A. Finding attack techniques in privileged sessions

Twelve use cases could be identified out of the seventeen selected techniques as shown in Appendix B. From the nine

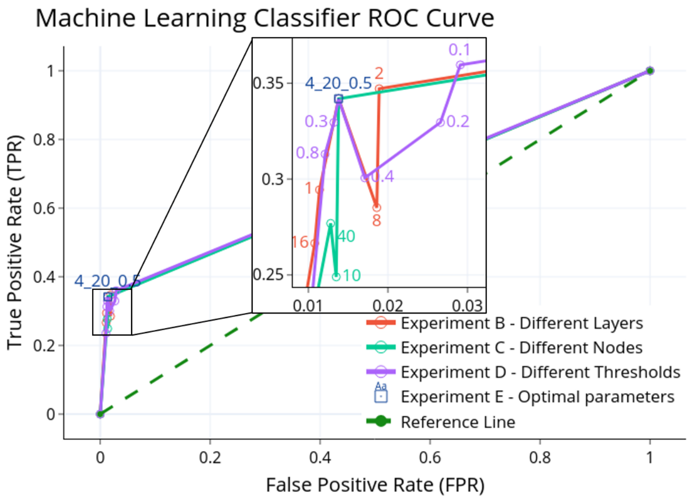


Fig. 8. ROC Curve - Machine Learning Classifier

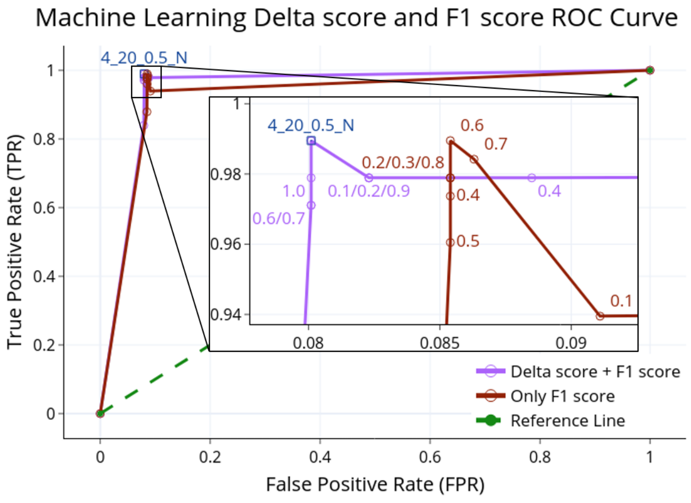


Fig. 9. ROC Curve - Machine Learning with Delta score and F1 score

additional, six techniques were shown in the logs and were distinguishable from normal behavior.

The remaining five attack techniques did either not generate syslog information or were not identifiable as malicious. We found that some syslog messages were not distinguishable enough in Windows due to the fact that it resembles normal behavior (i.e. T1) or in a session only the window title is captured without detailed information regarding the process name (i.e. T1, T15, T16 and T17). When an executable is run, this is visible in the logs as a security warning or an application that starts. In case of T15 (Input Capture), an application was started, that opened a way to capture all key strokes in the session, without showing this capture in the logs as it was not part of the window title.

Using the window title of an application is valuable for starting applications for e.g. user creation (i.e. T4) or running commands in Command Prompt (i.e. cmd), where the command is shown in the window title after executing a command (i.e. T2). However, when opening Windows Powershell, the command is not shown in the window title, which leads to a blind spot.

A similar event happens when a script (i.e. startup.bat in T3) runs. The Command Prompt will start the script, but does not show the commands in the script on the window title. This is also the case when running scripts using the Command Prompt as Administrator in Windows or the command line in Linux. In the logs was found that a script ran, but that was all that the CyberArk PAS system triggered.

B. *Findings additional attack techniques in CyberArk PAS*

In the additional attack techniques, three attack techniques were not visible in the log. When a client uses the PVWA, web session cookies could be captured (i.e. A3), without generating logs. This result was expected as this is done on the client outside of the CyberArk PAS system.

Disabling the security rules in the PVWA (i.e. A4), no logs were generated in the system. These rules are used to take actions in a session and to generate alerts, resulting in decreasing capabilities of the system as a whole. In order to manage the changes on the security configuration, the rights to change this should only be given to a restricted group administrators and the changes should be visible in the logs to be able to detect it.

1) Circumventing the CyberArk PAS system: The CyberArk PAS system only shows in the syslog what is happening inside the system. When the PSM is being circumvented (i.e. in A2 when a Remote Desktop connection is started directly, instead of going to the PSM first), the PAS did not send syslog messages, which leads to a gap in alerts.

The impact of this depends on how well CyberArk is implemented in the environment. The ideal situation would be connecting to RDP through the PSM component of the PAS system, but this could be altered by an administrator (e.g. in attack technique A2), allowing a user to connect directly via RDP. It should be possible to connect to a system directly, in case e.g. the PSM does not function properly, but this should be restricted to a single local administrator account. Monitoring these user accounts are crucial to minimize the risk of this attack.

Another way of minimizing the risk for attack technique A2 is by expanding the number of different alerts that can be generated by CyberArk PAS, which lead to a better understanding of the environment. Due to the current reporting limitations, these events can still go unnoticed. Within the privileged session only window titles or keystrokes are possible to record, which limits the capabilities of in-depth monitoring.

It is still possible to use this limited information, as can be seen in the use cases of Appendix C, but having more detailed alerts of the actual event, without relying on what is going on on the screen, could be a welcome addition. One way of extending the capabilities of CyberArk is running a service on the target host, which communicates to the Vault when e.g. a user is connecting to the server without using the PSM. This event should then raise some alerts that should be visible in the syslog messages.

C. *Verification of use cases in live environment*

During the verification on a live system, there were no results found from the Splunk queries. This could have different causes:

- There were no malicious events in the system that match with the attack technique.
- The Splunk queries were too strictly defined. In order to mitigate this, we loosened the queries and results were shown, but these results were not found to be malicious.
- The Splunk queries were based on the test setup in English and the language on the live system could differ, resulting in mismatches in events (i.e. Command Prompt is translated).

When verifying again we found that the log entries were formatted differently, because an older version was used on the production environment (i.e. version 11.2.0000 instead of 11.4.0000). Instead of `|411|Window Title|5|act="Window Title"`, the logs were formatted as `|411|Window[Title|5|act="Window]` including text brackets and therefore a query based on "act" was not working. When we altered the queries to "Event ID" (e.g. 411), the queries were generating results. However, the query could also be triggered if later in the log same ID is used in e.g. a command, which leads to a false positive in the query results. Because the "Event ID" is enclosed between pipes (i.e. "`|`") in the syslog message, these pipes are also included in the search query (e.g. "`|411|`"), which minimizes the chance of false positive results on the "Event ID" itself.

The other queries all succeeded in generating valuable output to investigate further, except for A9 (i.e. Shutting down Vault). This query ran, but did not show any results. This was due to the fact that a service (i.e. NotificationEngine) was not running on the Vault in this environment and the query only looks at the logs that are generated by this service. In the test environment, this service was available, so we can assume that this query works in other environments.

Another attack technique (i.e. T8/Archive Collected Data) did generate data, but this was not exclusively malicious. We removed a part of the Windows query (i.e. `" % Complete"`) that shows a dialog when extracting or compressing information, but in this case compressing was only necessary. When we changed this, the results of the query were adequate.

We found during the verification of the search queries that they depended on specific parameters (e.g. language, software version and services) used in the test environment. These parameters could be different in a real environment (e.g. the client environment during verification) and therefore should be adjusted based on what is needed to function properly. However, the test environment included the standard CyberArk setup running with all default services on the latest version possible (i.e. 11.4.0000) and these use cases should therefore be a well indicator of how it should work when updating properly.

D. Experiment results detector

The TPR and FPR in Figure 7 and Appendix F were clustered in the top-left corner of the ROC curve, away from the reference line. This means that this detector is performing well when classifying malicious logs successfully as well as minimizing the amount of FPs.

The curve and Table XI also show that when using other parameters for this neural network, the improvements in the results were minimal. The biggest difference is using another data set (i.e. "N+S") instead of only using "N" as normal behavior. We found that this was caused by the genetic neural network setup used in this machine learning application. In a couple of iterations, the classification of the detector had significantly improved due to the selection of the top 3%, that it stopped increasing with the same pace. It looked like it approached the global optimum, where it was difficult to make drastic improvements and therefore classifying the logs mostly in a similar way. An example of this behavior is visible when other amounts of nodes per hidden layer was tested (i.e. Experiment C). Here, the results were the same, where only the Delta score varied a bit. This shows that using a genetic training approach performs well in solving the problem of handling with an unbalanced data set, without fully depending on the optimal parameters.

When we looked at using other classification threshold values (i.e. Experiment D), using a value of 0.8 resulted in similar results as using 0.5 as threshold. The only difference was that in 0.8 one TN was classified as malicious (FP). The other values resulted in worse results. Therefore, a classification threshold of 0.5 is found to be optimal in this situation.

The detector was able to filter out 376 (i.e. 98.95% TPR) of the 380 malicious logs when using the N data set without suspicious logs and 4 hidden layers, 20 nodes per hidden layer and 0.5 classification threshold. This was the reference setup, which turned out to be working best for this type of classification.

The four remaining malicious logs were from the "NotificationEngine" (i.e. Shutting down the Vault in A9) that were hard to determine to be malicious for the neural network. We found that the logs looked like logs from normal behavior, where only the time and user are different. Similar events can go unnoticed where the differences are minimal, resulting in the fact that the results of the bag of words approach does not show a significant anomaly. However, what we saw in other attacks is that there were more differences than only a user (e.g. "host", "reason" and "external_id"), which made the work easier for the detector.

The other 376 malicious logs were classified successfully and from the 2272 normal behavior logs, 182 logs (i.e. 8.01% FPR) were falsely marked as malicious (i.e. FP). This was due to the fact that the bag of words approach was applied and these logs looked like malicious behavior (e.g. editing a file with a notepad++.exe instance in Windows), while these logs were part of the normal behavior simulation of a system administrator. These FPs would appear as alerts between malicious logs in a portal as discussed in the framework and cause noise. The amount of FPs are lower than the amount of actual malicious logs, resulting in a system that is still able to show malicious behavior without losing focus. Another advantage of this detector, is that this noise (i.e. amount of FPs) is still less than the FPs when the use cases were used in the string matching approach. The use of this detector works therefore well in this circumstance.

The time needed to train the neural network is an important factor of the applicability of this solution in a live environment. When the training takes time, it takes time to be be fully operational. Therefore, this training time needs to be as short as possible, while having still a reliable classification. During the experiments, the training time of the detector was captured. It took around 25 minutes in the reference setup to reach a correctness score above 90%, which was still increasing after this time. This indicates that the time to make this detector operational is limited. When we took a small setup (i.e. 1 hidden layer, 10 hidden layer nodes), the correctness was above 90% after slightly more than 9 minutes, where a complex setup (i.e. 12 hidden layers, 40 nodes per hidden layer) took around 3.5 hours. This shows that when the complexity of the neural network is increasing, the training times also increase substantially, but that in a couple of minutes decent results could be reached. Another observation of this is that when the complexity of a neural network increases, the performance does not necessarily increase.

E. Experiment results classifier

In comparison to the detector model, it is more difficult for the classifier to reach a True Positive Rate close to 1 as can be seen in Figure 8. This is mainly caused by the fact that every single log entry is not a single yes or no question, but rather 17 of them at the same time. One TP should result in 16 TNs, which is a different application than the detect or. Due to this high amount of TNs, the scores where lower than when the question is boolean. The classifier scored with 331 TPs (i.e. 34.19% TPR) and 637 FNs (i.e. 1.38% FPR) still significant. Since the sample ratio of the pure malicious logs per technique is significantly lower than the ratio of normal logs versus pure malicious logs, it is to be expected that this result is different than the detector.

The results in Table XII in Appendix D show that the use of other parameters influence the outcome of the model more than that was the case in the detector experiments. Using the parameters of the reference setup gave the best results overall, where in some experiments the TPR was better than the reference, but had a worse FPR or vice versa. For example, in Experiment B, the use of two hidden layers instead of four gave a higher TPR (i.e. 0.3471 over 0.3419), but resulted in a worse FPR (i.e. 0.0189 over 0.0138). In the same experiment, using 16 hidden layers resulted in a better FPR, but gave a worse TPR than the reference setup. This indicates that when other parameters are used, the workings of the neural network change.

Using different classification thresholds results in varying values for the classifier. Using a threshold of 0.1 resulted in a higher TP value, but due to a higher FPR, this threshold value is not optimal. A value of 0.7 gave the lowest FPR, but lacked a good TPR value. In this situation, the 0.5 classification value that was used in the reference setup gave the optimal results.

When we looked into the logs that were misclassified in the optimal setup, it turned out that some logs could be linked to multiple models. If the logs look similar and the classifier chooses one of the other models to match, this behavior is inevitable. In order to solve this misclassification, the models that are used should contain logs that are exclusively linked to this model or the classifier should be adjusted to show the multiple models that a log can be linked to with a confidence score. This way, when an alert is being triggered in the Portal, it should be clear what is happening or what could happen based on this log.

The use of the portal is also part of the training phase of this classifier, as the data set should be periodically updated with new logs and the portal being used as a way to optimize the training by editing how the system operates when using a portal.

F. Influence of using Delta score next to F1 score

When comparing the differences of the results in Figure 9, it is clear that the Delta score has a positive underlying effect on the F1 score. Using different classification thresholds results in a smaller FPR, except for the 0.4 threshold, indicating that log entries are classified more accurately. The reasoning for this is that the Delta score can improve in smaller fractions, indicating the building of momentum of the F1 score. In other words, as the Delta score improves, the chance of an improvement of the F1 score is higher, making this Delta score part of the success of this machine learning application.

IX. Conclusion

In this research, the following research question is answered: *How can one recognize malicious behavior based on the logs from CyberArk PAS in both present and future?* Two sub questions are formulated to answer this, regarding defining use cases based on the CyberArk PAS logs and providing a method to detect future incidents based on previously researched behavior.

In order to define use cases, attack techniques from the MITRE ATT&CK Enterprise Matrix were used create attacks on the CyberArk PAS Proof of Value (PoV) environment. Seventeen attack techniques were defined for simulating malicious behavior in privileged sessions via the CyberArk Privileged Session Manager (PSM) and nine additional techniques were created to attack the other components of CyberArk PAS.

The attacks were performed and use cases were defined as search queries based on the output logs of the CyberArk Vault. Next to these attack techniques, normal behavior was simulated to create system administrator behavior logs without malicious events. The logs were sanitized and filtered to become data sets of normal behavior, malicious behavior and suspicious behavior (i.e. the captured logs from the attacks with malicious behavior filtered out).

An architectural framework has been introduced, using models based on the use cases to perform string matching on the incoming logs and classifying them as malicious if the logs match. An alert could be raised and interaction can take place to act on this by the IT security team of an organization. This approach works on pre-defined use cases and is therefore not agile enough to adjust to unknown future incidents.

To be able to adapt to changes in behavior and to prepare for future unknown incidents, an architectural framework for malicious log classification was proposed. In this framework, genetic neural networks are applied, which is a machine learning technique. The same data sets were used to train the neural network for normal behavior and malicious behavior with a bag-of-words approach. To test the performance as well as the optimal parameters of this application of genetic neural networks, experiments were defined and performed.

The optimal parameters are the following for both Detector and Classifier: Four hidden layers, 20 nodes per hidden layers, a classification threshold of 0.5. In the Detector around 99% of the distinguishable malicious logs could be successfully identified as malicious, resulting in a True Positive Ratio (TPR) of 98.95 % and a False Positive Ration (FPR) of 8.01%. The classifier was setup differently, where by matching the malicious log to a model, one match (i.e. TP) resulted in 16 negative results for the other models. This classifier scored with 34.19% TPR and 1.38% FPR still significantly, where logs could sometimes be matched to multiple models.

X. Future work

Other machine learning techniques can be applied and compared to improve upon the capabilities of machine learning upon detecting malicious logs. The use cases are not exhaustive and additional use cases can be defined in future research.

Another topic that remained untouched is how the CyberArk PAS system can be extended. Due to the limited test setup, it was not possible to edit the alerting system or define new alerts. This research depends on the data that was sent and if this data will be expanded, the application of the solution will also be more valuable.

Furthermore, additional research could be done regarding the feed forward system from Figures 6 and 5. Having an additional loop where the machine learning model could learn from its previous classification could mean that no more user interaction is required to train the model to identify normal behavior. By reducing the requirement of human input, the training process will be less prone to human error.

Another future point of focus to attempt pattern recognition based on multiple logs instead of a single log. While our networks have been trained on behavior (which includes patterns) of normal behavior, identifying both short and long-term patterns could provide additional information on how a malicious action was performed. This could provide security engineers more detailed information on how to resolve a security issue whenever a malicious log has been detected.

References

[1] Cristina Abad, Jed Taylor, Cigdem Sengul, William Yurcik, Yuanyuan Zhou, and Ken Rowe. Log correlation for intrusion detection: A proof of concept. In *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, pages 255–264. IEEE, 2003.

[2] Jason Brownlee. A gentle introduction to k-fold cross-validation. https://machinelearningmastery.com/k-fold-cross-validation/. Accessed on June 23rd 2020.

[3] Jason Brownlee. Overfitting and underfitting with machine learning algorithms. https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/. Accessed on June 23rd 2020.

[4] CyberArk. Privileged access security solution architecture. https://docs.cyberark.com/Product-Doc/OnlineHelp/PAS/Latest/en/Content/PASIMP/Privileged-Account-Security-Solution-Architecture.htm. Accessed on May 27th 2020.

[5] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[6] Thiago S Guzella and Walmir M Caminhas. A review of machine learning approaches to spam filtering. *Expert Systems with Applications*, 36(7):10206–10222, 2009.

[7] Tiko Huizinga. Using machine learning in network traffic analysis for penetration testing auditability. https://rp.delaat.net/2018-2019/p39/report.pdf. Accessed on June 2nd 2020.

[8] Max Landauer, Florian Skopik, Markus Wurzenberger, and Andreas Rauber. System log clustering approaches for cyber security applications: A survey. *Computers & Security*, 92:101739, 2020.

[9] G Meera and G Geethakumari. Event correlation for log analysis in the cloud. In *2016 IEEE 6th International Conference on Advanced Computing (IACC)*, pages 158–162. IEEE, 2016.

[10] MITRE. Matrix - enterprise — mitre att&ck. https://attack.mitre.org/beta/matrices/enterprise/. Accessed on June 3rd 2020.

[11] Javaid Nabi. Machine learning — text processing. https://towardsdatascience.com/machine-learning-text-processing-1d5a2d638958. Accessed on June 2nd 2020.

[12] RC Rowe and Elizabeth A Colbourn. Neural computing in product formulation. *Chem Educator*, 8:1–8, 2003.

[13] Koo Ping Shung. Accuracy, precision, recall or f1? https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9. Accessed on June 22nd 2020.

[14] Splunk. Security, siem and fraud — security solutions — splunk. https://www.splunk.com/en_us/cyber-security.html. Accessed on May 27th 2020.

[15] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *CoRR*, abs/1712.06567, 2017.

[16] Lewis Van Winkle. Github - codeplea/genann: simple neural network library in ansi c. https://github.com/codeplea/genann. Accessed on June 22nd 2020.

APPENDIX

A. Selected Attack Techniques

In Table III, the attack techniques are selected based on the MITRE ATT&CK Enterprise Matrix[10]. The techniques have gotten an identifier (i.e. ID), where the IDs prefixed with 'T' were selected to be performed. The attacks were executed on Linux and Windows where applicable.

TABLE III
ATTACK TECHNIQUES FROM MITRE ATT&CK ENTERPRISE MATRIX

Relevant MITRE Categories: Initial Access, Execution, Defense Evasion, Collection, Persistence, Privilege Escalation, Credential Access and Impact.

ID	MITRE Technique Title	MITRE Category	Approach in Proof of Concept
T1	Phishing	Initial Access	Creating a phishing link and make a user click on it. The target is a website requiring user credentials.
T2	Command-Line Interface	Execution	Fill in elevated commands and create a script to perform administrative actions on a system.
T3	User Execution	Execution	Create a malicious binary to perform actions when the user clicks on it.
T4	Create Account	Persistence	Create a local user account, local administrator account and domain account via CLI.
T5	File and Directory Permissions Modification	Defense Evasion	Change permissions with an administrator on a file and document to be writable for unprivileged users and change the contents of the file by an unprivileged user, which has gained access now.
T6	Indicator Removal on Host	Defense Evasion	Have the user or a script alter log files and remove them altogether.
T7	Modify Registry	Defense Evasion	Have the user edit a registry key manually or by using a script.
T8	Archive Collected Data	Collection	Compress (e.g. ZIP or tarball) file(s) to make it ready for copying or uploading.
T9	Data from Local System	Collection	Use a script and use search function on the system to scan the entire local machine for "confidential" files.
T10	Data from Network Shared Drive	Collection	Use a script and use search function on the system to scan a network attached drive for "confidential" files.
T11	Data from Removable Media	Collection	Use a script and use search function on the system to scan removable media such as USB sticks for "confidential" files.
T12	Boot or Logon Initialization Scripts	Persistence, Privilege Escalation	Have a service or application run when the machine boots.
T13	Abuse Elevation Control Mechanism	Privilege Escalation, Defense Evasion	Edit sudoers file on a Linux system to prevent asking for a password when running a sudo command.
T14	Impair Defenses	Defense Evasion	Have the user change firewall rules on a system.
T15	Input Capture	Credential Access, Collection	Using a script to capture keystrokes and mouse movement.
T16	Steal Web Session Cookie	Credential Access	Using a script to fetch cookies from the browser's saving location.
T17	Data Encrypted for Impact	Impact	Writing a script to encrypt files using a shared key, similarly to what ransomware does.

TABLE IV
ATTACK TECHNIQUES IN CYBERARK PAS AND PVWA

ID	Title	Approach in Proof of Concept
A1	Suspicious password harvesting in PVWA	Open devices in PVWA and show passwords of multiple devices in a short time.
A2	Circumventing PSM	Open a direct RDP session with Domain Admin credentials, without using the PVWA to circumvent being recorded and logged.
A3	Capture client session cookies	Gather session cookies from Chrome to hijack PVWA session. Based on Steal Web Session Cookie.
A4	Deactivating security configuration rules	Login into PVWA and disable security configuration rules to remain under the radar when executing malicious events.
A5	Tempering with stored data in Vault	Login into PrivateArk Vault, open files, delete recording and edit file. This is based on Indicator Removal on Host
A6	Suspicious password harvesting in Vault	Open Vault with PrivateArk Client and then show and copy passwords of multiple devices and multiple vaults in a short time.
A7	Adding user manually to CyberArk PVWA	If a PVWA user has rights to add a user and assign privileges, this user can be used to perform privileged actions through PVWA. This is based on Create Account.
A8	Change user manually in CyberArk PVWA	If a PVWA user has rights to change and delete a user, this should be visible to detect malicious activities.
A9	Shutting down Vault	When the Vault has been shutdown, this can indicate a service outage or a change in the configuration. Both activities could be marked as potentially malicious.

B. *Recognized malicious behavior from attack techniques*

The result of the syslog analysis are included in Table V, Table VI, and Table VII. In these tables the malicious behavior per technique is written down as well as the descriptor of how the attack is distinguishable in the log. If the attack could not be properly categorized, the descriptor is marked as ”N/A”.

TABLE V
MALICIOUS BEHAVIOR FROM ATTACK TECHNIQUES

ID	MITRE Technique Title	Malicious behavior identified	Identified descriptor for attack in test setup
T1	Phishing	Not visible, as the logs only show the web browser being started.	N/A

T2	Command-Line Interface	Logs on Linux visible for editing, changing execution rights and running executable. On Windows only running commands the command line ”cmd.exe” was visible, which could indicate to be a weak descriptor. Using Powershell did not show any commands in the logs.	`Linux: ”event_id” is [”361”] and ”reason” contains [”.sh”] or [”chmod”] or [”chown”] or [”.py”] or [”python”] or [”./” and ”.sh”] or [”wget”] or [”curl”]`
			`Windows: ”event_id” is [”411”] and ”reason” contains [”cmd.exe” and ”Command Prompt –”] or [”cmd.exe” and ”C:\\Windows\\system32\\cmd.exe”] or [”Administrator: Command Prompt –”] or [”Administrator: Window Powershell”] or [”.bat”] OR [”.ps1”] or [”.py”] or [”python”]`

T3	User Execution	Only executed on Windows as command line execution is covered in Linux in T2 and file execution was visible	`Windows: ”event_id” is [”411”] and ”reason” contains [”.exe”] or [”Malware”] or [”Security Warning”]`

T4	Create Account	Account creation visible on command line (Linux and Windows) and GUI on Windows	`Linux: ”event_id” is [”361”] and ”reason” contains [”useradd”] or [”passwd”]`
			`Windows: ”event_id” is [”411”] and ”reason” contains [”net user” and ”/add”] or [”net localgroup” and ”/add”] or [”mmc.exe” and ”Properties”] or [”mmc.exe” and ”New Object”] or [”mmc.exe” and ”Select ”]`

T5	File and Directory Permissions Modification	Attack visible in logs on command line and GUI	`Linux: ”event_id” is [”361”] and ”reason” contains [”chmod”] or [”nano”] or [”vi”] or [”vim”]`
			`Windows: ”event_id” is [”411] and ”reason” contains [”dllhost.exe” and ”Properties”] or [”dllhost.exe” and ”Select User”] or [”dllhost.exe” and ”Permissions”] or [”dllhost.exe” and ”Advanced Security Settings”]`

T6	Indicator Removal on Host	Attack visible in logs on Linux, not distinguishable on Windows as only the title of cmd is shown without command (i.e. del) and only a pop up is shown when permanently deleting.	`Linux: ”event_id” is [”361”] and ”reason” contains [”rm ”] or [”sed ”]`
			`Windows: ”event_id” is [”411”] and ”reason” contains [”Delete”]`

T7	Modify Registry	Only possible in Windows and attack is visible.	`Windows: ”event_id” is [”411”] and ”reason” contains [”regedit.exe”] or [”Registry Editor”] or [”cmd.exe” and ”reg”]`

TABLE VI
MALICIOUS BEHAVIOR FROM ATTACK TECHNIQUES (CONTINUED)

ID	MITRE Technique Title	Malicious behavior identified	Identified descriptor for attack in test setup

T8	Archive Collected Data	Visible in logs on both systems, but only using the native archiving wizard in Windows or applications (e.g. 7Zip).	`Linux: "event_id" is ["361"] and "reason" contains ["zip"]`
			`Windows: "event_id" is ["411"] and "reason" contains ["7zG.exe"] or ["zip"] or ["Compress"]`

T9	Data from Local System	Only visible in logs that a script was being run and a command was executed. In Windows, the location was not visible, which results in an inaccurate descriptor.	`Linux: "event_id" is ["361"] and "reason" contains ["find" and " / "]`
			`Windows: "event_id" is ["411"] and "reason" contains ["explorer" and "Search Results"]`

T10	Data from Network Shared Drive	Only visible in Windows log that a script was being run, but not distinguishable from T9	N/A
T11	Data from Removable Media	Only visible in Linux log that a script was being run and a find command was executed	`Linux: "event_id" is ["361"] and "reason" contains ["find" and [[" /mnt"] or [" /mount"]]`

T12	Boot or Logon Initialization Scripts	Visible in logs of both systems. However, it is not visible what is being done exactly (i.e. Only file and folders mentioned in logs that could indicate malicious behavior).	`Linux: "event_id" is ["361"] and "reason" contains ["/etc/rc.d/rc.local"] or ["rc.d"] or ["rc.local"]`
			`Windows: "event_id" is ["411"] and "reason" contains ["explorer.exe" and "Startup"] or ["explorer" and "Search Results"]`

T13	Abuse Elevation Control Mechanism	Visible in logs on Linux (i.e. Windows out of scope)	`Linux: "event_id" is ["361"] and "reason" contains ["visudo"]`

T14	Impair Defenses	Visible in logs on both systems, but limited to native firewalls	`Linux: "event_id" is ["361"] and "reason" contains ["iptables"] or ["disable firewalld"] or ["enable firewalld"]`
			`Windows: "event_id" is ["411"] and "reason" contains ["mmc.exe" and "Windows Firewall"] or ["mmc.exe" and "Rule Wizard"] or ["mmc.exe" and "Firewall" and "Properties"]`

T15	Input Capture	Visible in logs on Windows (i.e. Linux out of scope), but only Security Warning (i.e. covered in T3) when running the executable. However, not distinguishable as attack because of an inaccurate descriptor.	N/A
T16	Steal Web Session Cookie	Visible in logs on Windows (i.e. Linux out of scope), but only Security Warning (i.e. covered in T3) when running the executable. However, not distinguishable as attack because of an inaccurate descriptor.	N/A
T17	Data Encrypted for Impact	Partly visible in logs on both systems: only unzipping and Security Warning (i.e. covered in T3) when running the executable. However, not distinguishable as attack because of an inaccurate descriptor.	N/A

TABLE VII
RECOGNIZED ATTACK TECHNIQUES IN CYBERARK PAS AND PVWA

ID	Title	Malicious behavior identified	Identified descriptor for attack in test setup
A1	Suspicious password harvesting in PVWA	Visible in logs, but counter needed to measure how many times this alert has been triggered in a certain time frame.	`"event_id" is ["295"] and "msg" contains ["Show Password"] or ["Copy Password"]`
A2	Circumventing PSM	Not visible in log	N/A
A3	Capture client session cookies	Not visible in log	N/A
A4	Deactivating security configuration rules	Not visible in log	N/A
A5	Tempering with stored data in Vault	Visible in logs, but hard to identify as this corresponds with normal behavior with retrieving and storing files. "suser" and "shost" should be considered to make it distinguishable.	`["event_id" is ["50"] or ["51"] or ["52"] and "cs2" contains [" PSMRecordings"]] or [["event_id " is ["0"] or ["1"] or ["73"] or ["142"] or ["145"] or ["148"] or ["149"] or ["154"] or ["155"] or ["170"] or ["183"] or ["188"] or ["189"] or ["198"] or ["272"]] and " shost" is not "<PSM IP address >"`
A6	Suspicious password harvesting in Vault	Visible in logs, but hard to identify as this corresponds with normal behavior with retrieving and storing files. "suser" and "shost" should be considered to make it distinguishable. A counter is needed about how many times this alert has been triggered in a certain time frame and the IP address of the PSM should be excluded to prevent normal behavior from being marked as malicious. The field "msg" should be empty to prevent messages from A1.	`"event_id" is ["295"] and "msg" is [""] and "shost" != "<PSM IP address>"`
A7	Adding user manually to CyberArk PVWA	Visible in logs, but "suser" and "shost" should be considered to make it distinguishable.	`"event_id" is ["180"] or ["265"]`
A8	Change user manually in CyberArk PVWA	Visible in logs, but "suser" and "shost" should be considered to make it distinguishable.	`"event_id" is ["184"]`
A9	Shutting down Vault	Visible in logs, but only as a LogOff action from user "NotificationEngine".	`"event_id" is ["8"] and "suser" is ["NotificationEngine"]`

C. *Splunk queries based on recognized malicious behavior*

In Table VIII and Table IX the identified descriptors are converted to queries. These queries could be used in Splunk to filter the events related to the attack techniques. If the descriptor was "N/A" in Appendix B, the attack technique is not defined in the tables. In Table IX users are mentioned (i.e. PasswordManager) in two queries (i.e. A5 and A6), which are standard users of the PSM in the test setup. If this user deviates in another environment, this query should be adjusted.

TABLE VIII
SPLUNK QUERIES FROM ATTACK TECHNIQUES

ID	MITRE Technique Title	Splunk query
T2	Command-Line Interface	((act="Keystroke logging" OR "|361|") AND (".sh" OR "chmod" OR "chown" OR ".py" OR "python" OR ("./" AND ".sh") OR ("wget" OR "curl")) OR ((act="Window Title" OR "|411|") AND (("cmd" AND "Command Prompt -") OR ("cmd" AND "C:\\Windows\\system32\\cmd.exe") OR "Administrator: Command Prompt -" OR "Administrator: Window Powershell" OR ".bat" OR ".ps1" OR ".py" OR "python")) NOT VaultMonitor
T3	User Execution	(act="Window Title" OR "|411|") AND (".exe" OR "Malware" OR "Security Warning") NOT VaultMonitor
T4	Create Account	((act="Keystroke logging" OR "|361|") AND ("useradd" OR "passwd")) OR ((act="Window Title" OR "|411|") AND ("net user" AND "/add" OR "net localgroup" AND "/add" OR "mmc.exe" AND "Properties" OR "mmc.exe" AND "New Object" OR "mmc.exe" AND "Select")) NOT VaultMonitor
T5	File and Directory Permissions Modification	((act="Keystroke logging" OR "|361|") AND ("chmod" OR "nano" OR "vi" OR "vim")) OR ((act="Window Title" OR "|411|") AND (("dllhost.exe" AND "Properties") OR ("dllhost.exe" AND "Select User") OR ("dllhost.exe" AND "Permissions") OR ("dllhost.exe" AND "Advanced Security Settings"))) NOT VaultMonitor
T6	Indicator Removal on Host	((act="Keystroke logging" OR "|361|") AND (("rm ") OR ("sed "))) OR ((act="Window Title" OR "|411|") AND ("Delete")) NOT VaultMonitor
T7	Modify Registry	(act="Window Title" OR "|411|") AND (("regedit.exe") OR ("Registry Editor") OR ("cmd.exe" AND "reg")) NOT VaultMonitor
T8	Archive Collected Data	((act="Keystroke logging" OR "|361|") AND ("zip")) OR ((act="Window Title" OR "|411|") AND ("7zG.exe") OR ("zip") OR ("Compress")) NOT VaultMonitor
T9	Data from Local System	((act="Keystroke logging" OR "|361|") AND (("find" AND " /"))) OR ((act="Window Title" OR "|411|") AND ("explorer" AND "Search Results")) NOT VaultMonitor
T11	Data from Removable Media	(act="Keystroke logging" OR "|361|") AND (("find" AND (" /mnt") OR (" /mount")))
T12	Boot or Logon Initialization Scripts	((act="Keystroke logging" OR "|361|") AND (("/etc/rc.d/rc.local") OR ("rc.d") OR ("rc.local"))) OR ((act="Window Title" OR "|411|") AND (("explorer.exe" AND "Startup"))) NOT VaultMonitor
T13	Abuse Elevation Control Mechanism	(act="Keystroke logging" OR "|361|") AND (("visudo")) NOT VaultMonitor
T14	Impair Defenses	((act="Keystroke logging" OR "|361|") AND (("iptables") OR ("disable firewalld") OR ("enable firewalld"))) OR ((act="Window Title" OR "|411|") AND (("mmc.exe" AND "Windows Firewall") OR ("mmc.exe" AND "Rule Wizard") OR ("mmc.exe" AND "Firewall" AND "Properties"))) NOT VaultMonitor

TABLE IX
SPLUNK QUERIES FROM ADDITIONAL ATTACK TECHNIQUES

ID	MITRE Technique Title	Splunk query
A1	Suspicious password harvesting in PVWA	(act="Retrieve password" OR "|295|") AND msg=*Password* NOT VaultMonitor| bucket _time span=30s | stats count by suser,shost | search count>2
A5	Tempering with stored data in Vault	((act IN ("Store File", "Retrieve File","Delete File") OR "|50|" OR "|51|" OR "|52|") AND cs2="*PSMRecordings*") OR ((act IN ("Delete File", "Delete Folder", "Delete Safe", "Delete Location")) OR "|0|" OR "|1|" OR "|73|" OR "|142|" OR "|145|" OR "|148|" OR "|149|" OR "|154|" OR "|155|" OR "|170|" OR "|183|" OR "|188|" OR "|189|" OR "|198|" OR "|272|") NOT VaultMonitor AND suser!=PSMApp_COMP01 AND suser!=PVWAAppUser
A6	Suspicious password harvesting in Vault	(act="Retrieve password" OR "|295|") NOT msg="*" NOT VaultMonitor AND suser!=PSMApp_COMP01 AND suser!=PVWAAppUser | bucket _time span=30s | stats count by suser,shost | search count>2
A7	Adding user manually to CyberArk PVWA	(act IN ("Add User", "Add Group Member") OR "|180|" OR "|265|") NOT VaultMonitor
A8	Change user manually in CyberArk PVWA	(act="Delete User" OR "|184|") NOT VaultMonitor
A9	Shutting down Vault	(act="LogOff" OR "|8|") AND suser="NotificationEngine"

D. Activation function performance test

In Table X a performance comparison between two activation functions (i.e. Sigmoid and ReLU) have been performed. The two functions were given 10.000 samples to process. The Sigmoid function took around 16 minutes and 10 seconds, whereas the ReLU function needed less time (i.e. 2 minutes and 9 seconds) to finish.

TABLE X
ACTIVATION FUNCTION PERFORMANCE TEST

```cpp
// Requires C++11 or later
// Necessary includes
#include <iostream>
#include <chrono>
#include <cstdint>

// Activation functions
double sigmoid(double a)
{
  return 1.0 / (1 + exp(-a));
}

double relu(double a)
{
  if (a < 0) return a * 0.01;
  return a;
}

// Main
int main(int32_t aArgC, char* apArgV[])
{
  static constexpr int32_t sampleSize = 10000; // Total samples
  int32_t startSample = -(sampleSize / 2); // Start in the negative, upwards to positive

  // Test the Sigmoid function
  auto sigmoidStart = std::chrono::high_resolution_clock::now();
  for (int32_t i = 0; i < sampleSize; ++i)
  {
    sigmoid(startSample + i);
  }
  auto sigmoidEnd = std::chrono::high_resolution_clock::now();

    // Test the RelU function
  auto reluStart = std::chrono::high_resolution_clock::now();
  for (int32_t i = 0; i < sampleSize; ++i)
  {
    relu(startSample + i);
  }
  auto reluEnd = std::chrono::high_resolution_clock::now();

    // Print results
  std::cout << ”Time elapsed (Sigmoid): ” << std::chrono::duration_cast<std::chrono::nanoseconds>(
    sigmoidEnd - sigmoidStart).count() << std::endl;
  std::cout << ”Time elapsed (RelU): ” << std::chrono::duration_cast<std::chrono::nanoseconds>(reluEnd -
    reluStart).count() << std::endl;

  return 0;
}

// Output (formatted) as nanoseconds:
// Time elapsed (Sigmoid): 970200
// Time elapsed (RelU):    129200
```

E. Experiment Results Detector

Table XI provides the results of the detector experiments, based on the last iteration. One iteration consists of a sliding window of four parts, hence the 25% validation set.

TABLE XI
EXPERIMENT RESULTS FROM VALIDATION SET OF DETECTOR IN TEST SETUP

Reference setup: 2272 Normal Behavior logs (N), 2648 suspicious logs (S), 380 malicious logs (M), 4 hidden layers, 20 nodes per hidden layer, Threshold 0.5

Exp.	Hidden layers	Hidden layer nodes	Data sets	Threshold	TP	TN	FP	FN	F1 Score	Delta Score	Training Score	TPR	FPR
A	4	20	N, M	0.5	376	2090	182	4	0.8015	0.9299	0.8657	0.9895	0.0801
A	4	20	N+S, M	0.5	251	4556	364	109	0.513	0.9236	0.7183	0.6972	0.074
B	1	20	N, M	0.5	372	2090	182	8	0.7962	0.9281	0.8621	0.9789	0.0801
B	2	20	N, M	0.5	372	2090	182	8	0.7962	0.9284	0.8623	0.9789	0.0801
B	4	20	N, M	0.5	376	2090	182	4	0.8015	0.9299	0.8657	0.9895	0.0801
B	8	20	N, M	0.5	372	2085	187	8	0.7919	0.9265	0.8592	0.9789	0.0823
B	12	20	N, M	0.5	376	2087	185	4	0.7988	0.9287	0.8638	0.9895	0.0814
C	4	10	N, M	0.5	376	2090	182	4	0.8015	0.9298	0.8657	0.9895	0.0801
C	4	20	N, M	0.5	376	2090	182	4	0.8015	0.9299	0.8657	0.9895	0.0801
C	4	40	N, M	0.5	376	2090	182	4	0.8015	0.9299	0.8657	0.9895	0.0801
D	4	20	N, M	0.0	380	0	2272	0	0.2506	0.9284	0.5895	1.0	1.0
D	4	20	N, M	0.1	372	2085	187	8	0.7919	0.9265	0.8592	0.9789	0.0823
D	4	20	N, M	0.2	372	2085	187	8	0.7919	0.9265	0.8592	0.9789	0.0823
D	4	20	N, M	0.3	319	2092	180	61	0.7259	0.9091	0.8175	0.8395	0.0792
D	4	20	N, M	0.4	372	2071	201	8	0.7803	0.9212	0.8508	0.9789	0.0885
D	4	20	N, M	0.5	376	2090	182	4	0.8015	0.9299	0.8657	0.9895	0.0801
D	4	20	N, M	0.6	369	2090	182	11	0.7924	0.9272	0.8598	0.9711	0.0801
D	4	20	N, M	0.7	369	2090	182	11	0.7924	0.9272	0.8598	0.9711	0.0801
D	4	20	N, M	0.8	376	2089	183	4	0.8009	0.9288	0,8648	0.9895	0.0805
D	4	20	N, M	0.9	372	2085	187	8	0.7919	0.9265	0.8592	0.9789	0.0823
D	4	20	N, M	1.0	372	2090	182	8	0.7962	0.9284	0.8623	0.9789	0.0801
E	4	20	N, M	0.5	376	2090	182	4	0.8015	0.9299	0.8657	0.9895	0.0801

F. Experiment Results Classifier

The results of the classifier experiments are presented in Table XII based on the last iteration. One iteration consists of a sliding window of four parts (4-fold cross validation), hence the 25% validation set.

TABLE XII
EXPERIMENT RESULTS FROM VALIDATION SET OF CLASSIFIER IN TEST SETUP

Reference setup: 376 entries malicious (training set and validation set), 17 models (techniques), 4 hidden layers, 20 nodes per hidden layer, Threshold 0.5

Exp.	Hidden layers	Hidden layer nodes	Threshold	TP	TN	FP	FN	F1 Score	Delta Score	Training Score	TPR	FPR
B	1	20	0.5	285	15311	177	683	0.3985	0.7287	0.5636	0.2944	0.0114
B	2	20	0.5	336	15196	292	632	0.4211	0.7301	0.5756	0.3471	0.0189
B	4	20	0.5	331	15275	213	637	0.4379	0.7903	0.6141	0.3419	0.0138
B	8	20	0.5	276	15200	288	692	0.3603	0.7051	0.5327	0.2851	0.0186
B	16	20	0.5	258	15321	167	710	0.3704	0.7532	0.5618	0.2665	0.0108
C	4	10	0.5	241	15279	209	727	0.3398	0.7158	0.5278	0.249	0.0135
C	4	20	0.5	331	15275	213	637	0.4379	0.7903	0.6141	0.3419	0.0138
C	4	40	0.5	268	15289	199	700	0.3735	0.7885	0.581	0.2769	0.0128
D	4	20	0.0	968	0	15488	0	0.1111	0.8221	0.4666	1.0	1.0
D	4	20	0.1	348	15037	451	620	0.3939	0.7898	0.5918	0.3595	0.0291
D	4	20	0.2	319	15076	412	649	0.3752	0.7675	0.5713	0.3295	0.0266
D	4	20	0.3	319	15283	205	649	0.4277	0.7978	0.6128	0.3295	0.0132
D	4	20	0.4	291	15223	265	677	0.3816	0.7288	0.5552	0.3006	0.0171
D	4	20	0.5	331	15275	213	637	0.4379	0.7903	0.6141	0.3419	0.0138
D	4	20	0.6	239	15289	199	729	0.3399	0.7171	0.5285	0.2469	0.0128
D	4	20	0.7	228	15327	161	740	0.336	0.7923	0.5641	0.2355	0.0104
D	4	20	0.8	303	15300	188	665	0.4154	0.6816	0.5485	0.313	0.0121
D	4	20	0.9	132	15315	173	836	0.2073	0.7715	0.4894	0.1364	0.0112
D	4	20	1.0	208	15305	183	760	0.306	0.7419	0.5239	0.2149	0.0118
E	4	20	0.5	331	15275	213	637	0.4379	0.7903	0.6141	0.3419	0.0138