Contextual Profiling of Homogeneous User Groups for Masquerade Detection

Pieter Bloemerus Ruthven

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2014

Contextual Profiling of Homogeneous User Groups for Masquerade Detection

Pieter Bloemerus Ruthven

2014/06/01

Abstract

The complexity of modern computer networks creates a number of information security challenges for organizations. As the use of computer systems increases they become more targeted by criminals. In order to limit damages, the ability to detect probable criminal activity as soon as it occurs is of paramount concern. Intrusion Detection System (IDS) is a technology that has been in existence for a number of decades. It aims to identify patterns indicative of an attack, or alternatively, behaviour that is suspicious compared to some notion of normality. In order to ensure the effectiveness of these systems research efforts are required to adapt them to the ever changing threat landscape.

Currently, internal threats pose a large risk to organizations bringing along with it additional challenges, as not all threats can be detected using known patterns. Behaviour based methods, know as anomaly detection, has the benefit of detecting previously unseen attacks. Profiling is a common technique used to establish a baseline for normal behaviour. However, normality can be difficult to define when considering individual profiles. Group profiling can offer additional context that can form the basis for better comparison in order to detect the presence of abnormal behaviour. It also reduces the scope of the IDS and in so doing removes some of the background noise.

This thesis evaluates the application of group profiling methods as a contextual means to detect internal threats, specifically masquerade attacks. It delves into related theoretical knowledge and derives a framework used for masquerade detection research. The study frames the masquerade detection challenge as a classification problem, primarily focusing on the profiling task. A relevant feature representation method is chosen. Features are extracted from a simulated data set using a script developed in Bro, and classified using Support Vector Machine as a machine learning method. Individual and group profiling results are presented.

Acknowledgements

I would like to extend my sincere gratitude to Prof. Dr. Katrin Franke for her guidance and support throughout this process. She has provided me with excellent opportunities for personal growth through stimulating challenges and the motivation to excel. I would also like to thank my co-supervisor, Prof. Dr. Slobodan Petrović for many fruitful discussions and valuable feedback. Thank you to Andrii Shalaginov for helpful discussions and his patient assistance with machine learning methods.

A special thanks to all my fellow students, especially John Erik Rekdal, André Nordbø and Espen Didriksen for their support, discussions, challenges and creating a pleasant environment.

Finally, thank you to my family for their support, motivation and understanding throughout my studies.

Contents

List of Figures

List of Tables

1 Introduction

The purpose of this chapter is to provide an overview of the researched topic in terms of the problem domain, a specific scope defined by the research questions, and methods used to conduct the research. An outline of the master thesis is provided at the end of this chapter.

1.1 Topics

We live in a highly complex computer networked environment. Computing has become nearly ubiquitous and an integral part of our modern civilization. The reliance on computer resources continues to grow, and many business wont be able to operate without it. Due to this reliance, and the economic value associate with it, these systems have naturally become more and more targeted by criminals. Thus, to protect the interests of society we need to ensure that these networks are as secure as possible.

Information security and digital forensics are ever evolving fields and many mitigation solutions exist and are being researched. This thesis topic is limited to one mitigation technique that addresses network security. Intrusion detection systems have long been used to protect computer networks against attacks. The basic principle behind these systems is to identify malicious network activity and provide a notification thereof in order to take the necessary actions.

Intrusion Detection Systems (IDS) can be configured as misuse or anomaly detection systems. Misuse systems rely on signatures, or known attack patterns, to identify malicious activities. Anomaly based systems are more dynamic, and can detect previously unknown types of attacks. This is achieved by analysing benign, or normal, activities and establishing a baselines of normal behaviour to which future observations can be compared against.

There are many challenges in this area. First and foremost being defining normality, and when we consider behaviour to be an anomaly. Even the same user can behave in different ways at different times, blurring the lines of abnormality. These systems are plagued by mistakes - identifying normality as abnormal. In real-world applications, this leads to a quick deterioration of trust from administrators which limits their practical applicability in production networks. There is thus great potential for continued research in this area.

Various techniques exist for deriving baselines, otherwise known as profiles. These include statistical models, data mining and machine learning approaches.

1.2 Keywords

Intrusion Detection, Anomaly Detection, Group Profiling, Behaviour Profiling, Masquerade Detection, Forensic Readiness, Bro IDS

1.3 Problem description

The ability to detect and react to the misuse of internal computing systems has historically been, and will continue to remain, an essential requirement of information security and forensic readiness. As the complexity of computing environments grow, the difficulty of finding such misuse increases.

Recent surveys indicate that internal threats represents one of the biggest security risks to organizations [1]. Motivation has been raised for user profiling to mitigate these risks [2]. There has also been increased focus on insider threats due to the pervasiveness of "Advanced Persistent Threats" and cyber criminal activities. In addition to computers compromised by outside attackers, one also need to be aware of legitimate user credentials being used to misuse resources and commit fraud. Called masquerade attacks. This is much harder to detect due to the absence of known attack indicators; such as malware or attack traces such as damaged systems. Seemingly benign behaviour can easily go unnoticed compared to blatant malicious activities, all of which is easily hidden in the vast amount of "noise" generated on networks. Early detection allows for gathering of valuable forensic evidence to support potential criminal cases, as well as mitigating the risks such as data theft and further compromise.

Profiling is a technique that can be used to limit this noise in the environment by providing abstractions of objects or entities on the network. Group profiling can add additional benefits. It enables better comparison between an individual profile and the group it belongs to, opposed to two unique individuals [3].

This thesis investigates anomaly detection, based on group profiling techniques, as a method to identify unusual behaviour. In theory, a behavioural profile can be established based on the common characteristics/features/attributes of a group of individuals.

1.4 Justification, motivation and benefits

In order to protect the interests of a business and its clients, organizations need the ability to detect misuse and hold the guilty party responsible. Masquerade detection is an evolving field, that requires ongoing research in order to stay current with threats. The complexity of today's networks creates many challenges for those trying to secure them. By strengthening the semantic relation between the human and technical solutions one is able to allow for better awareness of the environment. There is a need to understand behaviour within a context, not merely relying on known bad behaviour but also taking suspicious behaviour into consideration. Group profiling is one technique that aims to provide such improvements in the form of intuitive reasoning and contextual information. Groups can serve as a more relevant reference point to which individuals can be compared to.

The purpose of this thesis strays from the typical case of detecting malicious activities due to a security compromise. Instead focusing on anomalies observed compared to normal user behaviour. This includes spotting irregularities in the way a user performs his mandated tasks for which he has appropriate access to do; but does not usually do in this fashion, or when compared to his group. This is done by considering events at as high level of abstraction as possible in order to minimize noise.

1.5 Research questions

The following research questions define the breadth of research that this thesis takes.

RQ1 What are relevant features for defining normality in a group?

RQ2 Do features used for individual profiling also work of group profiling?

RQ3 How can such features be collected and measured?

RQ4 What impact does contextual data have on the characteristics of normal behaviour?

RQ5 What is the achievable performance for detecting abnormal behaviour of a homogeneous group of users?

RQ6 Can group profiling yield better performance than individual profiling within a well defined context?

RQ7 What is the achievable difference between one-class and multi-class classification results?

1.6 Methodology

An overview of the methodology used in this thesis is briefly detailed below. It should be noted that certain stages has occurred in parallel and iteratively improved upon as part of an exploratory phase. Such as feature extraction, where methods were revised based on the analysis of collected data. A combination of qualitative and quantitative methods has been used [4].

Literature study

An in-depth study of different profiling methods and techniques, including feature extraction and selection has been performed as a starting point. Documentation for Bro has been studied in order to learn how to utilise its scripting language for development of a proof-of-concept. A study has been performed to evaluate possible ways of performing data captures. This included the capability to replay captured data in order to have repeatable experiments. A consistent dataset is needed for robust testing and comparison between different implementations. Investigate alternative metrics for IDS performance measurement and comparison in order to provide as robust and unbiased test results as possible.

Preliminary testing

Continuous testing, using the knowledge gained from the literature study has been performed to ensure the capability to perform the experiment. This involved capturing and analysis of test data sets. The possibility of limiting the experiment on a specific service or application, such as web usage only was considered.

Experiments

A test environment was configured during this phase. The required hosts and servers have been configured using virtualization. Planning has been performed to define all assumptions made as well as to detail the configuration. Experiments are designed to be repeatable and reproducible in order for others to verify the obtained results. A test case consisting of two groups of users has been defined. Data is captured and analysed for feature selection. A Bro script has been developed and used to extract features from the data. This is used in a classifications task using machine learning techniques. A one-versus-others approach has been taken to simulate masqueraders.

Methods directly relating to research questions

Scientific research methods are typically categorized as either qualitative, dealing with descriptive data or quantitative, dealing with measurable data [5].

RQ1 *What are relevant features for defining normality in a group?*

A qualitative approach has been taken. Data has been collected from two sample groups. The selected participants was grouped to form a group based on their association, simulating a hypothetical work environment. Assumptions are based on a shared common characteristic, and as such individuals have to be related in some way.

RQ2 *Do features used for individual profiling also work of group profiling?*

The selection of features were based on a literature study of related work on individual profiling for masquerade detection. The performed empirical experimental studies assessed the viability of characterizing properties of the selected features.

RQ3 *How can such features be collected and measured?*

An in depth literature study has been performed. Existing available data sets has been evaluated according to how suited they were for this thesis. This has been combined with the application of different measurement methods on the used data set.

RQ4 *What impact does contextual data have on the characteristics of normal behaviour?*

A combination of qualitative and quantitative methods has been used. An in-depth literature study has been performed to identify possible sources of contextual data. Furthermore, experiments have been performed using a simulated data set.

RQ5 *What is the achievable performance for detecting abnormal behaviour of a homogeous group of users?*

Answers to this has been sought as part of a thorough literature study as well as analysis of the results achieved during the experiment.

RQ6 *Can group profiling yield better performance than individual profiling within a well defined context?*

A quantitative approach has been taken by performing a comparative study based on the selected data set.

RQ7 *What is the achievable difference between one-class and multi-class classification results?*

A quantitative approach has been taken by performing a comparative study based on the selected data set.

1.7 Contributions

The goal of this master thesis was to find and evaluate existing methods for user profiling and apply it on group profiling as a contextual approach for masquerade detection.

Theoretical

The methodology and framework applied during this thesis is presented in such a way that it can be used to perform future research in user and group profiling. A simulated environment and scenario was designed. Results are given on individual and group profiles, established using network monitoring and a machine learning classification approach.

Practical

A proof-of-concept design has been implemented using Bro IDS, Python and WEKA. The configuration of a virtual lab set-up, technical scenario design as well as a simulated data set that can be used for future experiments.

1.8 Scope

The scope of this thesis is limited to performing user and group behaviour profiling for application in masquerade detection, a subset of intrusion detection. This thesis does not implement a masquerade detection system, but focuses on the profiling task. The data used in the experiment is based on a simulated environment and scenarios generated during the study. Contextual information is considered in the abstract sense as a specific scope. In this thesis it specifically implies a users' group membership or a specific resource being accessed.

Feature selection is considered as a step in profiling and viable features are selected. However this thesis does not set out to compare different features. Also, only features visible from network traffic were considered. Bro has been chosen as the system to extract features. This thesis used Bro only to gather relevant features and not as a complete masquerade detection system.

Three different feature extraction methods have been implemented for preliminary experimentation, however only one method was selected for the final experiment. The possibilities of directly using Bro for masquerade detection is discussed in Chapter 7.

It is necessary to note a clear distinction between anomalies or abnormal behaviour and malicious activity. This thesis does not assume all anomalies to be malicious. Masquerade attacks focuses on compromised user credentials, however non-compromised users can also be responsible for anomalies. In other words, the question of attribution is not addressed, but a limited view is taken in order to detect abnormal behaviour performed using a user's credentials.

Performance aspects of different profiling methods have been considered, but has not been a deciding factor for the chosen methods.

1.9 Thesis outline

- Chapter 2 defines the taxonomy and high level concepts related to the research work. The aim is to ensure that the reader is familiar with the required foundation to conceptualise the rest of the material as intended.
- Chapter 3 narrows down the core concepts covered in this thesis by providing examples of what others have done in related fields. This chapter is concluded with examples of the current state of the art in masquerade detection.
- Chapter 4 puts forth a proposed methodology and theoretical framework for masquerade detection that was employed to perform the research in this thesis. It suggests an approach for each component of the framework, followed by the specific method that was selected for this thesis. Decisions are motivated by referring to previous chapters.
- Chapter 5 discusses the practical implementation of the selected methods and how it was performed as part of the experimentation.
- Chapter 6 presents and discusses the results of the experiment.
- Chapter 7 provides answers to the posed research questions and discusses the resulting theoretical and practical implications.
- Chapter 8 concludes the work by proposing future work and further research questions.

2 Background and Taxonomy

This chapter contains an introduction to the taxonomy of the problem domain. It discusses the required theory related to the area of research. It provides key definitions and descriptions of concepts needed as a foundation before moving on to more detailed discussions. Figure 1 provides an overview of a portion of the research domain taxonomy.

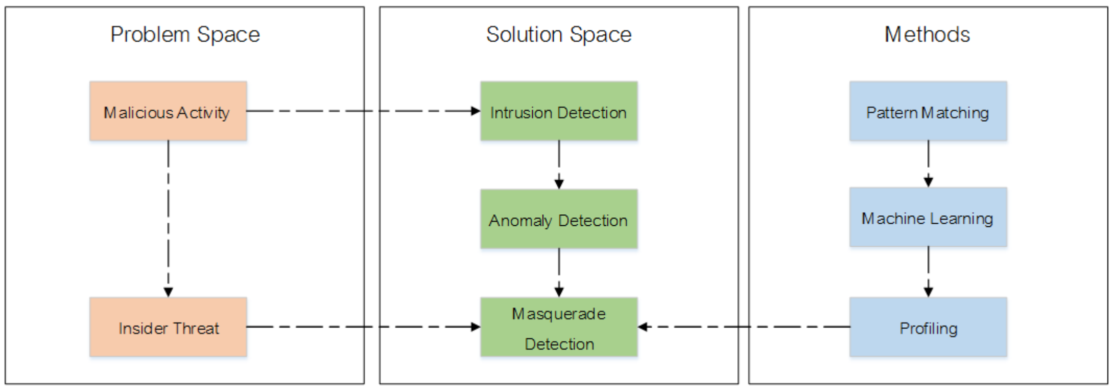


Figure 1: Overview of high-level taxonomy related to this thesis

2.1 Information Security Threats

Organizations currently face a complex threat landscape, from malware to high profile targeted attacks.

2.1.1 External

External attacks can be opportunistic in nature. Attackers can use various automated scanning tools to find vulnerabilities in systems and attempt to exploit them. Organizations with external facing services could be exposed to many such attempts throughout the day.

2.1.2 Internal

Internal threats arguably pose a higher level of risk. Attackers may already have acquired valid authentication credentials to systems. They may also have knowledge of system architecture and know which systems or users to target in order to achieve their objectives. Such attacks are usually harder to detect as attackers avoid using offensive attacks with the objective to stay undetected for prolonged periods.

The internal threat category do not exclude the risk an organization faces from employee actions against the security of the organization.

- Fraud
- Misuse of systems
- System compromise by an external party, misusing internal systems
- Advanced Persistent Threat

2.2 Forensic Readiness

Considering the proliferation of threats that organizations face, as mentioned above, one realizes the high level of likelihood of their occurrence. It is clear that organizations need the capability to efficiently and effectively detect and respond to such security incidents. The ability to respond to forensic cases in a corporate environment, and having good quality forensic evidence to leverage for further investigation. The goal of forensic readiness can be defined as:

> "...the ability of an organisation to maximise its potential to use digital evidence whilst minimising the costs of an investigation."[6]

A fundamental aspect of this ability, and a starting point for the whole process, is knowing when an incident has occurred. Organizations generate an enormous quantity of data, and it is not feasible to keep indefinite record of each and every action. It is crucial that an organization know what information equates to evidence, and when to start recording it.

2.3 Intrusion Detection Systems

Intrusion Detection Systems (IDS) are used to detect malicious or anomalous activities on network systems. They can be considered a key element in an organizations forensics readiness capability [7]. The generic model of an IDS is depicted in Figure 2. An IDS can be categorized according to the detection method used, which can either be misuse or anomaly detection[8].

Misuse systems rely on a database of known attack patterns and indicators called signatures. These systems use pattern matching techniques in order to detect attacks. Such systems offer good detection rates against known attacks, but are not effective against attacks that are not known. Detection signatures have to be regularly updated, and most systems allow for customization of signatures to take specific environment considerations into account. *Anomaly Based Detection systems* commonly utilises statistical techniques, amongst others, to build models of normal behaviour on the network at varying levels of abstraction. In addition to detection phases, the system includes a training phase to establish the normal model or baseline. It also has to consider keeping the behaviour model up to date in a dynamically changing environment. Such systems are capable of detecting previously unseen attacks, but are largely plagued by high false positive rates (expanded on in Section 2.3.2).

According to [9], anomaly based IDS methods can been categorised at a high level as:

- Statistical based
- Knowledge based
- Machine learning based

An IDS alerts an administrator to any detected attacks or anomalies based on configured system and environmental parameters.

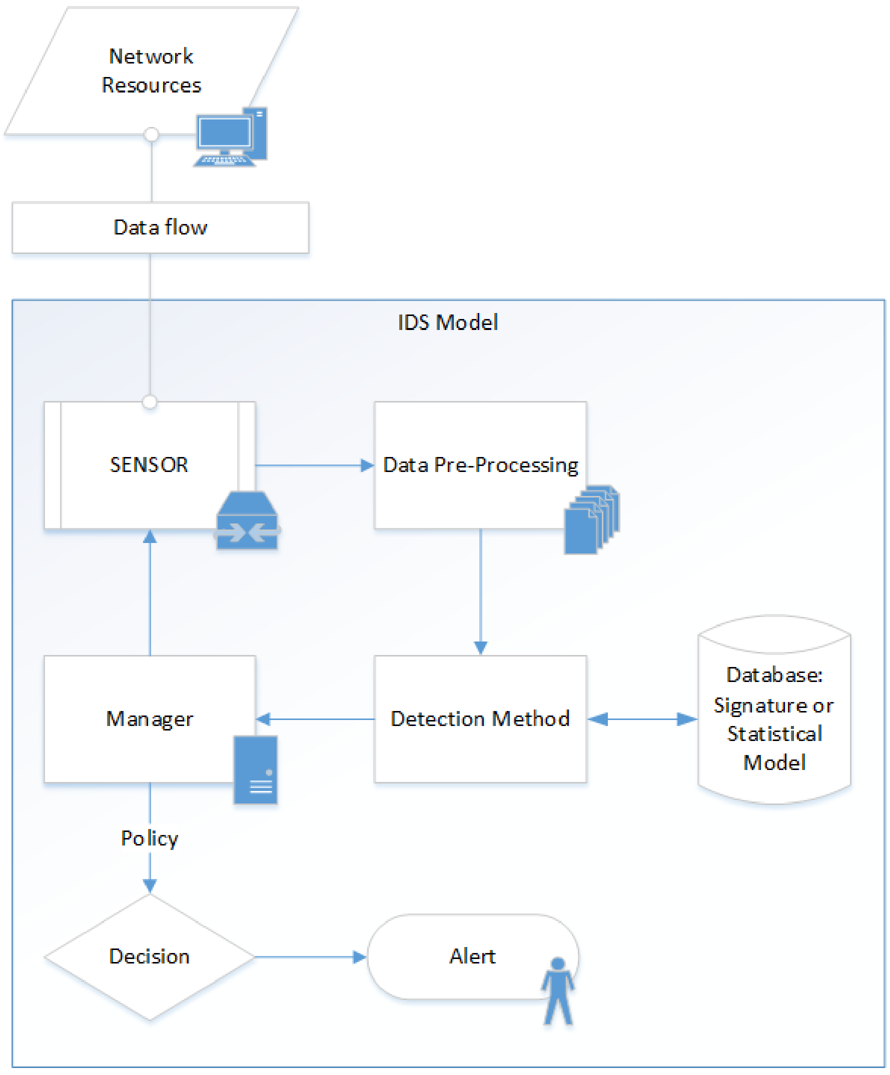


Figure 2: Generic Intrusion Detection Model

2.3.1 Masquerade Detection

The scope of an IDS can further be subdivided into two separate domains, *External* or *Internal* detection. IDS has been traditionally used for perimeter defence against outside threats, neglecting threats from insiders.

Masquerade detection is a subset of *Anomaly Based* Intrusion Detection that specifically deals with detecting unauthorised use of a user's credentials by another malicious party. This means that an attacker has gained knowledge of the targeted user's credentials and can leverage this legitimate access to system resources for malicious activity. The fundamental concept is to ascertain whether the user truly is who they claim to be. Detection is commonly achieved by monitoring certain behavioural aspects of a subject's (individual user, or group of users) interaction with the network or system.

The task of building a baseline for masquerade detection can best be likened to establishing a profile for each subject. The concept of profiling is elaborated on in Section 2.4. Masquerade detection is the primary are of focus for this thesis and is further discussed in Section 3.5.

2.3.2 Performance Metrics

The effectiveness of an IDS is determined by how accurate its detection rate is. The most basic quantitative measures used for IDS performance assessment are, *False Positive (FP), False Negative (FN), True Positive(TP), True Negative(TN)*.

False positives occur when an IDS incorrectly classifies benign traffic as malicious. False Negatives occur when an IDS does not detect an attack from observed traffic.

Commonly used comparative performance metrics can be derived from these measures in the following way:

True Positive Rate (TPR)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{2.1}$$

True Negative Rate (TNR)

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{FP} + \mathrm{TN}} \tag{2.2}$$

False positive rate (FPR)

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{2.3}$$

Performance can be visually represented by using a Receiver Operator Characteristic (ROC) Curve, depicted in Figure 3. This best portrays the trade-off between prioritizing detection at the risk of including false detections, or not missing detections rather than detecting false positives. The Area Under the Curve (AUC) can also be calculated from the ROC Curve to compare different systems.

Drawing direct comparisons between different IDS implementations are complicated due to configuration differences as well as datasets used for the assessment [9]. A vast number of results are still being based on outdated datasets such as KDD Cup 99, which have been shown to have a number of shortcomings [10]. Furthermore, the Base Rate Fallacy [1] poses more problems when considering performance due to highly unbalance datasets [11].

[1]An erroneous statistical assumption regarding the distribution of benign vs. malicious samples in a population

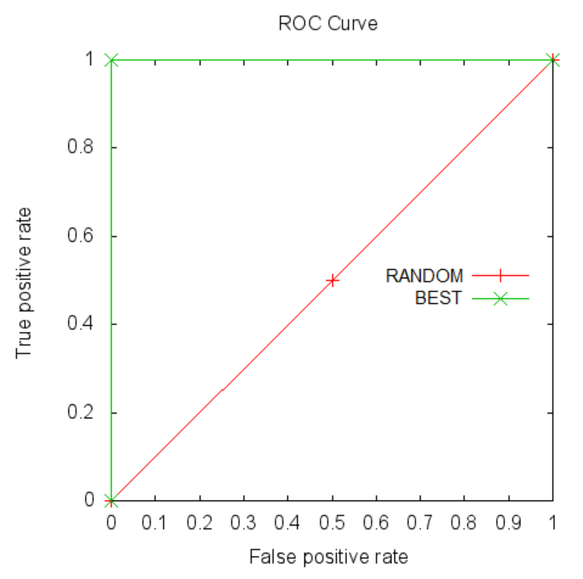


Figure 3: Receiver Operator Characteristic Example

Whilst computational complexity is also a key consideration for IDS evaluation and method selection it is not discussed as a topic for this thesis. The potential impact was still taken cognizance of to avoid the selection of unsuited methods.

2.4 Profiling

Profiling is a core concept used throughout this thesis. For the intent of this thesis it can be considered synonymous with base lining or establishing a normal profile relating to anomaly detection. Thus when profiling is mentioned, it implies analysing and capturing the behavioural aspects of user interactions.

2.4.1 Approaches

A differentiation between two high level approaches can be made as follows:

Individual Profiling

This form of profiling is limited to representing a single individual. However, there exists some similarities to group profiling, as in most cases the individual can be represented by a profile template that is in common with multiple individuals. [12]

Group Profling

There are considered to be two general group profiling approaches [3].

- Learning a profile according to a group of users, grouped by their roles or association.
- Grouping users based on the similarities to their individual behaviours, even though not necessarily associated.

In other words, groups can be defined as people who identify with each other, or people that have no connection but has been placed in a group as part of a profiling exercise.

Group profiles can also be derived from aggregating individual profiles. Alternatively this approach could be used as a cross-reference to evaluate the validity of an established group profile.[13]

2.4.2 Profile Granularity

The construction of a profile starts with choosing the characteristics which are to be considered to find an accurate descriptive set of attributes representing the profiled subject. The characteristics up for consideration can be evaluated at varying levels of abstraction:

- High level: Such as; activity characteristics, actions performed, decisions made, websites visited, music listened to (more indicative of intent)
- Low level: Such as; timing information, sequence information, keyboard and mouse interactions, the sub components required to perform a task (more nuanced)

Which characteristics to choose is also influenced by the type of approach described in Section 2.4.1. This thesis considers that the characteristics, and thus the level of abstraction, for establishing an individual vs. a group profile is intuitively different. Depending on the grouping, or the objective of the grouping profiling might require more general characteristics. Specific details such as timing might be too granular, and specific to individuals only. On the other hand, they could also be representative of a group of individuals such as elderly people or people of a similar experience level.

2.5 Machine Learning

Furthering the introduction of profiling, means of generating profiles from potentially large amounts of data have to be considered. Machine Learning is one approach, and as such the basic concepts thereof is discussed in this section.

Machine Learning is a method used to either classify or predict values (regression) based on an established model. It aims to automate a task that would normally be too tedious or too complex for a human to solve in a realistic time frame. It can be formally defined as:

> "A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*" [14]

There are many ways to approach problems, and when using machine learning it is paramount to select the method that best suits the problem domain and data space [15]. The main phases of machine learning, depicted in Figure 4, consists of *Training* and *Testing*. During training, the classifier learns a model based on a supplied data. Testing assesses the performance of the model by classifying known samples and comparing the outcome of the classification.

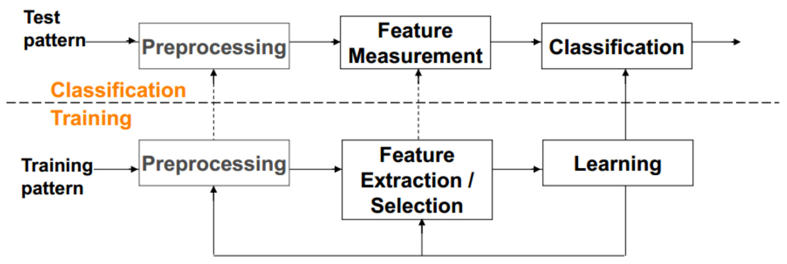


Figure 4: The phases of Machine Learning. Figure from [16].

Machine Learning can be subdivided into *Supervised* and *Unsupervised* categories. A brief overview of different methods follows next.

Supervised

Supervised methods rely on an adequate number of representative examples to learn a general model. They can be used to solve classification or regression problems. Some example methods being[17]:

- *k*-nearest neighbours (k-NN): Classification is based on the proximity of a new observation to other known examples. A higher number of near data points from one class increases the likelihood that the new observation belongs to the same class.
- Support Vector Machines (SVM): An optimized linear separating hyperplane is constructed that separates binary class examples. New observations are classified according to which side of the hyperplane they lie.
- Artificial Neural Networks (ANNs): A statistical method that simulates how the human brain functions. Specific output pathways are formed based on threshold activated *neurons*. Input received on these *neurons* are transformed and weighted, so that a specific range or combination of inputs will yield a classification as output.
- Decision Trees: A tree-like graph where input variables are nodes on a decision tree, each edge represents a variable value and the subsequent leaf is the classification label or regression value. Based on the input variables a path will form that leads to the correct leaf.

Unsupervised

Unsupervised methods make no assumptions on the data and attempts to organise data into meaningful groups. They can be applied to classification problems in terms of finding hidden patterns or structures in data. Some example methods being[17]:

- Expectation Maximization (EM): A parameter estimation technique that seeks to find the most likely statistical distribution model based on observed data instances.
- *k*-Means Clustering: Clusters of data points are created based on their similarity measure to a shared centroid. A *k* number of centroids are assigned. Through an iterative process, at the end of which the centroids move to the mean of the distance between itself and

its clustered data points. With each iteration the data points are assigned to the cluster of which centroid they are nearest to.

2.5.1 Classification

Classification can be viewed from different perspectives depending on the problems being addressed and the selected method. For a classification problem, a data set can be commonly represented as a set of observations in the form of vectors, $\Omega = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ where x_i is the feature vector relating to the *i*-th observation data point and y_i is the ouput of the classification task, indicating the class label. Classification problems can be categorize further according to the number of classifications a unique observation can belong to [17].

One Class

This is an outlier detection method where the classifier is trained with examples of a single class only.

Binary Class

The IDS classification problem is commonly represented by two classes. Benign network traffic is labled as *Negative* class samples and malicious network traffic as *Positive*. The classifier is trained with a dataset containing both negative and positive samples.

Multi-Class

Identification problems, such as biometrics, typically require multiple class labels. Each object is assigned an unique classification label.

2.5.2 Features

Features are the measurable characterising attributes that defines an object or behaviour, broadly mentioned in Section 2.4.2.

Features can be of the following types:

- Ordinal - Ranking information, such as first, second or third. This can provide sequential information, but can not be used for calculations.
- Binary - Positive or negative information, indicating the presence or absence of a specific feature or the pattern such as a signal.
- Categorical - Values are limited to a finite set of values such as colours or species of animal.
- Numerical - A count value that can be used in calculations.

Machine learning algorithms rely on a combination of features, called feature vectors, that are descriptive of the objects that are to be classified.

Feature Selection

Feature extraction and selection is the process by which the most descriptive set of characterising features are chosen, often from a much larger number of possibilities. Performance is a key consideration during this process, as high dimensionality in the feature spaces can increase computational complexity to infeasible levels. The objective is to find the least amount of features that will still provide accurate classification. Depending on the problem a different trade-off between accuracy and speed might be chosen [15].

2.5.3 Challenges

The general task of deriving a model for classification is faced with many challenges. Common problems are highlighted below.

Over-fitting

Over-fitting occurs when too many parameters are narrowly defined in a model over a specific training data set. This can occur when training was optimized for performance in stead of generalizing classification capability. The risk is that future example observations, not originally present in the training data set, will not be correctly classified [17].

Under-fitting

Under-fitting occurs when classifier training fails to identify causal relationships in the training data set, yielding a simple model. Under-fitting is less likely to occur than over-fitting [17].

Dimensionality

The dimensionality of features greatly affects a classifier's performance, both in terms of computational time and classification accuracy [17].

2.5.4 Cross Validation

Cross validation is a method used during the testing phase whereby the same data set can be subdivided in order to test the robustness[2] of the model. n-fold Cross validation is performed by splitting data into n pieces and perform the training task n times. During each iteration the *n-th* piece of data is held back and used for testing purposes. This ensures that all samples in the data set are equally likely to appear in the training or test data set and reduces the chances of over-fitting your model to the data [18].

2.5.5 Performance Measures

Machine Learning performance can be expressed using the same metrics as put forth in Section 2.3.2. Metrics are measured by evaluating the predicted classifier outputs obtained from a testing data set compared to the correctly pre-labelled classifications. In addition, common information retrieval measures, precision 2.4 and recall 2.4 are used [19]:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.4}$$

$$\text{Recall} = TPR = \frac{TP}{TP + FN} \tag{2.5}$$

Precision indicates how accurate the classification is in terms of correctly classified examples given the total class specific population identified by the classifier. Recall is similar to TPR in this context and indicates the number of correctly classified examples over the total class specific population.

[2]How well the model performs and how well it provides a general solution to the problem space.

3 Related work

This chapter builds on Chapter 2, by furthering the discussion in more depth. Important topics are revisited with a narrower scope on specific sub-domains, and focus on related research that has been carried out. The literature study consisted of reading about various aspects related to the research topic. Various components where specifically considered, and analysed in light of the overall topic and guided by the research questions.

3.1 Anomaly Based IDS

The seminal piece of work on anomaly based detection is found in [20]. The author defined an intrusion detection model that formed the basis for many future work. The fundamental concepts still remains applicable today. The author defined the following key components:

- **Subjects**: Initiators of activity on a target system (normally users)
- **Objects**: Resources managed by the system (files, commands, devices)
- **Audit records**: Generated by the target system in response to actions performed or attempted by subjects on objects – user Iogin, command execution, file access, etc.
- **Profiles**: Structures that characterize the behavior of subjects with respect to objects in terms of statistical metrics and models of observed activity. Profiles are automatically generated and initialized from templates.
- **Anomaly records**: Generated when abnormal behavior is detected.
- **Activity rules**: Actions taken when some condition is satisfied, which update profiles, detect abnormal behavior, relate anomalies to suspected intrusions, and produce reports. [20]

Some of the earliest work built on this model can be found in *Intrusion Detection Expert System (IDES)* [21] and its evolution, Next-Generation Intrusion-Detection Expert System (NIDES) [22]. These systems are some of the earliest network based anomaly detection intrusion systems incorporating profiling that showed promise. IDES and NIDES uses a statistical anomaly detector to establish a normal baseline based on the behaviour of individual users, groups, remote hosts and the entire system. Data observations are gathered from system audit logs. However, the statistical test used to detect anomalies assumes that observations are normally distributed.

As discussed in 2.3.2, anomaly based systems are typically plagued by a large number of false positives, impacting their practical applicability. Thus, a key area of research for these systems are in lowering their false positive rate. The authors of [23] use a technique that combines a number of classifiers in order to form a more robust anomaly detector and lower the false positive rate. In [24], further challenges faced in evaluating IDS performance are discussed. A systematic approach is suggested by [10] that focuses on dynamically generating data sets based on changing requirements. The authors used a profiling approach to represent the behaviour of different network protocols and services. They developed agents able to generate realistic data sets based of the established profiles.

Another frequently researched area is that of feature selection. The motivation being both achieving increased performance in terms of computational complexity as well as detection capability. Much work has been performed on feature selection, specifically for anomaly based IDS systems. Different algorithms are discussed in [25] and [26], with more generic feature selection measures discussed in [27]. It is clear from these that the selection of features have a big impact on performance and accuracy. Contextual methods have the potential to generate a vast amount of features. Thus special care has to be taken that the most suitable features are selected.

From the work evaluated during the literature study it is considered that most anomaly based systems are more focused on detecting low level anomalies such as unusual network traffic patterns or unusually high throughput. This could be indicative of malware or denial of service attacks being launched against the network. However the methods used are not necessarily effective at detecting abnormality in high level user behaviour. To address this, masquerade detection is discussed in Section 3.5.

3.2 Bro

Here a practical example is introduced to concretize the discussion of IDS in Section 2.3. This system was used during the experimentation performed as part of this thesis. Therefore additional details are provided that cover relevant aspects of its architecture and design.

Bro originally started out as an IDS research platform, but has evolved into what the developers call a network monitoring framework with many production deployments in existence [28]. It is still a popular research platform due to its flexibility in the sense that it can be configured as either a misuse detection system or as an anomaly detection system [29].

Architecture

As mentioned, Bro can be configured to provide both misuse or anomaly detection, achieved through a high-level scripting language.

The primary concept of Bro, and the reason it was used in this thesis, is its ability to provide abstraction of network activity. Bro takes raw network data, and tries to extract meaning from it and put it into context. It represents this information through *events*. *Events* are high level activities on the network, such as details on a user making a HTTP connection to retrieve a file. This event driven information can be used in the scripting environment in order to achieve IDS objectives, or other network operational tasks such as performance monitoring. This information flow through Bro's architecture layers is depicted in Figure 5.

Bro utilizes a number of Protocol Identification Analyzers (PIA) that enables it to identify a large number of protocols. The analyzers inspect network packets and provides useful insight into various changes in state on an application layer level [30].

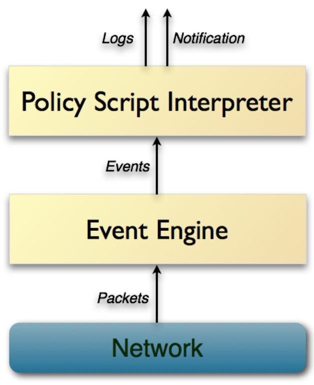


Figure 5: Bro architecture. Figure from [28].

Protocol Analyzers

Bro's PIA subsystem is capable of dynamically analysing incoming traffic streams. Bro uses a form of profiling in order to identify applications based on statistical properties of data packets observed over a certain period [29]. It starts by analysing the lowest level protocol, and further analyses subsequent streams in terms of their higher level protocols. It creates a tree structure that spawns additional PIA instances to continue the sub-identification up to the application specific level [30]. The tree structure resulting from an Internet Protocol (IP) stream can be seen in Figure 6. This process is applied to each session, however the tree structure can dynamically change as needed.

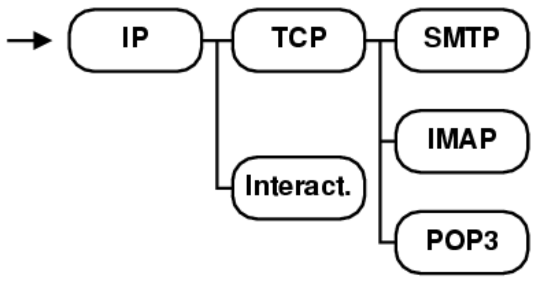


Figure 6: Protocol Analyzer Tree Structure. Figure from [30].

The default Bro implementation includes analysers for a large number of popular protocols, and the modular design of the system also allows for custom implementations.

Performance

Bro uses a distributed architecture that allows for the placement of different network monitors at key areas in a network. This increases its visibility of the network, and also increases the performance.

Frameworks

Bro provides the ability to extend its capabilities in the form of frameworks. As of writing this report, Bro included *File Analysis, GeoLocation, Input Framework, Intelligence Framework, Logging Framework, Notice Framework, Signature Framework, Summary Statistics*. These allow for a standardized abstraction of network concepts that can be used in the scripting environment. It also provides for different options in order to interface with Bro, and use other sources of information to supplement network information. This concept can be exemplified with the *Summary Statistics (SumStats)* Framework due to its relevance in anomaly detection [31].

SumStats

The Summary Statistics framework provide an efficient way of obtaining dynamic statistical metrics from live network flows. It handles any synchronization problems that might arise from having a distributed architecture, and provides results based on defined time intervals.

Multiple SumStat objects can be defined, each associated with a specific activity (or event) for which metrics are derived for. Each instance of such activity, an observation, can be passed to the SumStat object based on certain conditions that define the activity. Such as a host accessing a specific resource, or an HTTP connection being established from a specified host. Different statistical metrics such as *average, maximum, minimum, standard deviate, variance, unique values and streaming average* can be calculated from the recorded observations.

The power behind the SumStat framework lies not only it the memory efficiency of the algorithms, but also in its dynamic nature in the form of actions that can be defined based on the resulting statistical metrics once the time interval has elapsed. Actions can be explicitly performed after each interval, or can be conditional on a predefined or dynamically updated threshold. For example, given a time interval of an hour: if the number of connections made by a single host exceeds the average of connections made by all other hosts by two standard deviations an alert should be generated [31].

3.3 Profiling Strategies

This section looks at related work in the area of profiling and raises a few examples of different profiling strategies that aid with conceptualisation of the topic. Whilst this thesis focuses on group profiling strategies, a number individual profiling examples are raised for completeness due to the relevance of concepts and methods employed.

Recommender Systems

Profiling is heavily applied in the area of recommender systems. They are frequently employed by large companies that provide online service for huge amounts of users such as Netflix® and Amazon®. The aim of these systems are to find users who have similar personal preferences and behaviours, resulting in a profile. By using this established profile the company can improve marketing efforts and give a more personalised service to its users. Research in this area shows that most systems currently lack contextual awareness [32, 33]. Contextual information is an important aspect considered in this thesis and is elaborated on in Section 3.4.

Fraud Detection

Work on fraud detection is closely related to intrusion detection, in the sense that both aim to detect anomalies based on the deviation from a normal pattern of behaviour. One such method is the peer-group analysis technique described in [34]. This technique takes group characteristics into account in order to establish an aggregated baseline for normal behaviour, grouping similar individuals for comparison purposes. In this instance, the technique was used to detect credit card fraud based on transaction amounts. Where groups, or peers, were people with similar spending habits. An anomaly is detected by comparing the individuals with their peers. The principle is interesting, and relates to what this thesis aims to achieve, however the technique's ability to scale to more complex problems is unclear. [35] further matures this approach by using more features.

Identification

The identification task is also closely related to the approach required to perform anomaly based intrusion detection profiling, however the resulting profile is used to identify or authenticate a user.

Profiling methods have been suggested to be used for identification of users from web behaviour. Successful use of text mining techniques were used in [36]. The authors were able to positively identify users based on their DNS request patterns.

Another technique frequently found in the behavioural biometrics field is key-stroke and mouse dynamics. This can be applied for either authentication or identification purposes[37, 38, 39, 40]. These techniques rely on measuring temporal and velocity aspects of user behaviour through keyboard and mouse interactions. The basic assumption is that individuals have specific habits in interfacing with human input devices. This can be exemplified by typing style, both in terms of speed and pauses between keystrokes and mouse movements in terms of acceleration and mouse clicks.

Intrusion Detection

Profiling techniques have been shown to be quite popular for host based IDS [41]. This is likely due to the fact that more detailed information is available on this platform. Techniques such as system call analysis, monitoring temporal flow of operating system functions, are frequently implemented [42, 43, 44]. However they generally lack comparison of the profiles between individuals or a group, and could be seen as too isolated.

The benefits of analysing temporal aspects of web browsing was noted in the work [45]. [46] describes using Bayesian parameter estimation on web access patterns. The sequence of pages accessed during a session is used to determine a normal profile.

A slightly different point of view can be found in profiling the behaviour of a service or application instead of a user. Not only does this capture the inherent user interaction, but can also leverage the assumption that protocols follow a well defined standard implementation that has narrow operational thresholds. Deviations from this operational threshold can indicate misuse of the protocol in order to exploit a vulnerability such as buffer overflow. A higher level approach can be seen in [47] where the authors set out to identify web based services, such as offered by *www.flickr.com* and *www.google.com*.

3.4 Contextual Awareness

Contextual information is introduced as a separate section due to its importance in this thesis as a concept. For the purpose of this discussion the definition of context can be defined as two things:

- Contextual information: Supplementing data with knowledge to allow inference of more meaning
- Contextual scope: A limited scope, allowing more accurate assumptions to be made about the environment. Many isolated scopes can exist, and scopes can also be aggregated.

3.4.1 Contextual Information

Research on contextually supplemented profiles have shown promise. One of the original developers of Bro demonstrated one application of contextual information in IDS [48]. The authors gathered additional information from host computers in order to supplement IDS observations. The additional information also served as a way of vetting the IDS observations, as a form of cross-view analysis[1]. They motivated this method by listing the following benefits, *overcoming encryption, comprehensive protocol analysis, anti-evasion, adaptive security and IDS hardening* [48]. The contextual information was limited to HTTP information in this case. Comparing URL requests from the perspective of an Apache web server and Bro IDS. Differences were observed in the way URL's are processed on the web server compared to the IDS. Clear benefits of having additional information available to increase the accuracy of anomaly detection can be seen from this. Intuitively, knowing more about the operating environment allows easier creation of a normal baseline. The negative side to this is that even more information has to be analysed, and meaningful features extracted. It raises questions around the amount of complexity the additional data adds and any trade-offs that go along with it.

[50] aims to provide additional context from external sources. A framework was developed for Bro IDS. External data could be fed into the IDS, and expand its detection capability. Use case examples include dynamically updating a blacklist of known bad IP addresses obtained from attacks observed by other trusted sources. This is similar to other crowd-source techniques used in various areas in current times. This solution leverage existing knowledge about ongoing attacks. This highlights the potential benefits of contextual information.

Service, or application protocol, profiling [51] is another technique that provides contextual information specific to a service that shows promise. This is closely related to group profiling, as one can see the service as a group of similar data packets. However services are much more rigidly defined, and a derived profile could therefore carry more significance. It still demonstrates that focusing on the grouping of similar objects can potentially allow one to more easily detect anomalies.

Another, very thorough, survey [41] described the field of anomaly detection in general and exemplifies applications other than IDS. A broad range of anomaly detection use-cases are discussed. The authors describe contextual anomalies and state that profiling, either on a user or

[1]Viewing and comparing information pertaining to the same observation, obtained from different sources or by using different methods. Commonly used for malware detection. [49]

group basis, can be used to define contextual attributes. These attributes, in combination with behavioural attributes, can better help determine anomalies. The authors further state that the use of contextual information mostly applies to activity monitoring or fraud detection systems. The relation to IDS is not discussed, however it is mentioned that this technique is getting more attention and is being applied to other areas of anomaly detection.

In [52] the authors discusses the performance benefits of limiting processing to smaller, what the authors refer to as virtual work spaces. The work discusses user intent in terms of actions required to perform jobs.

3.4.2 Contextual Scope

The concept of a contextual scope can be elaborated on by discussing the concept of domain knowledge. Domain knowledge can be defined as information about the configuration and operation of a network environment as it relates to business activities. Contextual scopes can be establish from domain knowledge in the form of logically organized functional units in a business or roles of users. Domain knowledge can either be derived inherently from the system design, or extracted as latent information using data mining techniques. This creates a constrained, more defined area to which profiling can be applied.

[53] supports this view, the authors theorize that decreasing the scope for anomaly based IDS systems (and in this case machine learning specific implementations) will lead to lower misclassification. By narrowing the band of what is deemed normal; confining it into groups, abnormalities should more clearly stand out. Based on this description a group of users can be likened to a contextual scope.

3.5 Masquerade detection

The general concept of profiling has been discussed in section 2.4, and masquerade detection has been introduced as subset of anomaly detection in section 2.3.1. This section looks closer at masquerade detection, further defining it as a specific application of profiling. The focus is primarily on methods, but an general overview of the approach each researcher has taken to solve the masquerade detection problem is also provided. A brief discussion is included on any results that have been achieved.

3.5.1 Overview

Schonlau et al. (2000)

The most frequently referenced work on masquerade detection, and what can be considered the seminal paper on the topic, is found in [54]. The authors evaluated six different methods for masquerade detection, *Uniqueness, Bayes one-step Markov, Hybrid multistep Markov, Compression, Sequence-Match, IPAM*. Their assessment was performed on 15,000 UNIX command line interactions from a set of 50 users generated over a period of several months. In order to simulate a session to be evaluated, the commands were grouped into 150 blocks of 100 commands each.

The evaluated methods relied on classifying observed blocks as benign of malicious. Malicious observations were based on a simulated masquerade data derived from real user data. The data was a mixture and fusion of other users' data, that was then injected into each individuals command block sequence based a defined probability. The authors rationalise this decision by noting that the behaviour of a real masquerader would be unknown, and in this way removing bias from generating masquerade data that only models known attacks.

The authors used the first 50 blocks for training purposes, the following 100 remaining blocks either filled entirely with masquerade data or void completely thereof.

[54] describes each method's *Modelling* approach, detection *Thresholds* and means of *Updating* the model. The performance in terms of False Positive Rate is shown in Table 1.

Method	True Positive %	False Positive %
Uniqueness	39.4	1.4
Bayes one-step Markov	69.3	6.7
Hybrid multistep Markov	49.3	3.2
Compression	34.2	5.0
Sequence-Match	26.8	3.7
IPAM	41.1	2.7

Table 1: Schonlau et al. Performance Results. Table adapted from [54].

The authors concluded that statistical methods lend themselves well to addressing the masquerade detection problem. However, we have to note that the performance results, especially considering the True Positive Rate, leaves a lot of room for improvement. One of the primary contributions of their work was a research dataset for masquerade detection. A closer look is taken at their generated data set, "SEA", in Section 3.5.2.

Wang et al. (2003)

[55] performs a study on the capability of one-class classification methods for masquerade detection. The authors use one-class Naïve Bayes and one-class SVM methods on UNIX system call data. They note that one-class classification methods can offer great benefit to the real-world implementation of masquerade detection solutions. Training of each user's behaviour model can be performed by using their own data in the absence of masquerader data examples. The benefit is two-fold; less data is needed to estimate a model, and the challenge of finding realistic masquerader data is eliminated.

To obtain comparative results, the authors applied their method to the data set generated in [54]. They also followed the same experiment methodology. The authors noted that the methodology did not allow for the best comparison between different classification methods due to the random insertion of masquerade data. Individual user data sets contained varying levels of masquerade. For this reason they also applied an alternative method derive from [56], **1v49**. For each user, the first 5000 commands were used for training, the remaining 49 user's first 5000 commands was used as masquerade data.

The authors show that one-class classification methods can yield comparable, and in some cases better, results than their multi-class counterparts. Whilst offering valuable benefits such as being able to train on limited data that excludes potential attacker data which could be hard, or unrealistic, to simulate comprehensively.

Salem et al. (2011)

In [57] a slightly different approach based on file system search patterns can be seen. The authors hypothesize that a legitimate user of a system knowns their own file system in such a way that they traverse it in a concise manner. Whilst a masquerader would not have the same knowledge, and would thus demonstrate more sporadic search behaviours. These deviations in search behaviour are used to detect masqueraders against a normal search model. In contrast to other research, the authors wish to model a user's intent opposed to purely the execution of atomic commands. They postulate that user search behaviours are indicative of the intent shown to gain knowledge on a system.

The authors generated their own dataset called "RUU", consisting of normal and masquerader data, gathered from Windows machines using a custom developed sensor, further discussed in Section 3.5.2.

A one-class classification approach has been taken, thus training relies on benign observations only. The authors utilised one-class SVM classifier. The authors considered SVM models easily and efficient to update and accommodate changes in user behaviour. 80% of user data was used for training.

The obtained results, shown in Table 2, are very promising for the search behaviour method. The authors included another set of results, obtained from applying modelling using application frequency as a feature vector, for comparison. We do not consider the assessment of application frequency features to be fair. It is expected that careful preprocessing, as applied to the search intent feature set will outperform a more naive approach. For example, it was not clear whether system processes have been taken into account for the application frequency results.

Method	True Positive %	False Positive %
Search-behavior ocSVM	100	1.1
App.-freq. ocSVM	90.2	42.1

Table 2: Salem et al. Performance Results. Table from [57].

This research shows interest and inspired the scenario design aspects of this thesis. However, it has to be questioned whether it is safe to assume that an attacker has no knowledge of a file system. To circumvent such a detection method could prove trivial. A naive example, acknowledged by the authors, being an attacker obtaining means to perform a once-of copy of the data and analyse it from another system. Also, how background operating system activities impact the model is not clear. We consider it more promising to use command based data that is closely linked to the resources being acted upon.

Iglesias et al. (2012)

The authors of [58] take a sequential approach, named *Evolving Agent behaviour Classification based on Distributions of relevant events (EVABCD)*, to build user profiles based on command executed on a UNIX system. The research is an update to previous work performed by the authors of [59], focusing on the updating their established user model in a dynamically changing environment. Masquerade detection was not the direct goal of this research, but the authors did mention this as being a possible application.

A number of results were obtained for varying sequence lengths and number of commands issued. Unfortunately only the True Positive Rate was given as the research was mainly focused on the adaptive classification capability of their method. The method achieved average results, with a lowest classification rate of 58,3% and a highest rate of 72%. Being outperformed by methods used in comparison such as Naive Bayes and SVM. It should be noted that the method does provide benefits in terms of computational efficiency as past observations are not kept in memory and the model is updated in an online fashion.

The authors do not test their profiles against masquerader data as this was not their direct objective. However, the results assesses the average classification accuracy for all users. We can consider this to be indicative of its potential to detect a masquerader. However, careful consideration has to be taken to see whether the evolving properties of this method can be misused by a malicious user.

Eldardiry et al. (2013)

The following paper highlight conceptual ideas relating to masquerade detection using grouping approaches. [60] discussed methods to detect masqueradors according to two different notions. "*Blend-in anomalies*", where malicious users behaviour can be detected as abnormal in comparison to a group of users they belong to, and "*unusual change anomalies*".

The authors aim to utilise and combine different sources for behaviour observations. They refer to different sources, or "*domains*", such as *Device, Email, File Access, HTTP and Logon* domains[60]. They argue that detecting abnormal behaviour restricted to a single domain is ineffective due to the variance in behaviours observed by different users.

The authors make the assumption that users having a similar work role behave alike in each domain. Their first approach uses a K-Means clustering method in order to group similarly behaving individuals across separate domains. Features collected over the entire observation period is considered during the clustering task. An anomaly is generated when a user behaves differently than his cluster in any of the domains.

The second approach clusters user behaviour in each domain, and models the change in membership from one cluster to another over time using *Markov Model* and *Rarest Change Model* methods. The clustering approach differs slightly in that it only considers features collected for a day. Peers are assumed to transition between the same clusters. The estimated likelihood of cluster transitions is used as a score to assess abnormality of a single user compared to their peers.

Experiments were performed on synthesized and real data sets for approach 1 and 2 respectively. The authors used what they call "*work practice data*", which encompasses a wide range of user to system interactions such as file access, http access and email activity. All of which are supplemented with additional semantic tags, such as User ID, whether working hours normal or not etc.

Unfortunately the authors did not include results in terms of False Positive, or True Positive Rates, making it difficult to compare with previous work. Nonetheless, this research shows promise and provided conceptual support for the proposed approach of this thesis; highlighting the comparative power of peer group data derived from group profiling. However, we are not entirely convinced on the merit of comparing behaviours across disparate domains. This method could be too rigid for group profiling. We consider rather making the distinction on observations, and peer comparisons, in a single domain or context.

Other

Table 3 shows a summary of what we found to be the most relevant methods used in current state of art masquerade detection research. All research considered individuals opposed to groups.

Year	HMM	KNN	SVM	ocSVM	GMM	NB	EM	CRF	IBL	Other	Paper
2014		X	X								[61]
2013			X		X						[62]
2013						X	X				[63]
2012		X	X			X				X	[58]
2011			X								[57]
2011											[64]
2010				X						X	[65]
2007	X							X			[66]
2003	X								X		[67]
2003			X								[55]

Table 3: Masquerade Detection Methods

- **HMM** *Hidden Markov Model*
- **KNN** *K-Nearest Neighbour*
- **SVM** *Support Vector Machine*
- **GMM** *Gaussian Mixture Model*
- **N**B *Naïve Bayes*
- **EM** *Expectation Maximization*
- **IBL** *Instance Based Learning*

Role Based Access Control (RBAC) frequently employed in Identify and Access Management (IAM) solutions and database systems can be a way to group users. An example of an IDS employing this principle is discussed in [68] where the authors create profiles based on roles, grouping multiple users together based on the job function they perform in the database.

[69] describes a role-based approach, focusing on detecting internal threats using an profiling approach that is closely related to group profiling. In their work individual behaviour was compared to the expected behaviour of the group. Groups were defined using similar roles in the organization. Based on the detections, individuals who appear suspicious can be put under additional scrutiny by triggering additional monitoring. Feature selection was not discussed, so we can not determine on what basis their analysis was performed.

The benefit of grouping homogeneous systems in order to spot anomalies can be seen in [70]. The authors used this principle to leverage the similarities in systems and tasks in a corporate environment to identify rootkits[2].

3.5.2 Existing datasets

This section provides relevant details and summaries of existing datasets for masquerade detection. All of which were briefly introduced in Section 3.5.

In contrast with other anomaly based IDS systems that use network captures, datasets for masquerade detection are almost solely generated from host based audit data.

Schonlau - SEA

The work by *Schonlau et al.* [54] was introduced in Section 3.5.1. The dataset generated has been considered the *de facto* standard for masquerade detection research [71].

The dataset consists of data collected on 70 users. 50 Users were randomly selected as targets, and 20 users were used to simulate masqueraders by splicing different command blocks and injected them into a target command stream.

The dataset is generated from UNIX audit data containing only command data excluding any passed parameter information. Whilst this has been noted as one of the data set's weaknesses by many subsequent papers[61], the authors noted that parameters were purposeful omitted due to privacy concerns [54].

This motivated proposals for different methods. A large portion of subsequent research has been based on applying different methods to the dataset generated in [54].

RUU

[57] considered the lack of realistic masquerader data to be a severe drawback to the SEA dataset.

Activity data is gathered from registry access, processes, GUI window access, file access and DLL activity. Normal user data consisted of 18 Windows users. Masquerader data was obtained from three scenarios played out as exercises. The scenarios were designed to capture *malicious*, *benign* and *neutral* behaviour.

Information regarding DLL and processes were mapped to high level abstractions. Features were extracted from the dataset to represent search behaviour. The authors noted that the following features yielded best results during their experimentation [72]:

- Automated search actions (desktop search, index files accessed, etc.)
- Number of file touches (fetch, read, write, copy)
- File system navigation representing manual searching by the user

An observation window of 2-minutes were used, but no justification were given around this choice and could mean that it has been arbitrarily chosen or tweaked.

WUIL

One of the most recent publicly available data sets can be found in *Windows Users and Intruder simulations Logs (WUIL)* [61] that captures information on user file system navigation. The au-

[2]Persistent malware

thors were motivated to provide realistic examples of masquerader data.

The data set contains data from 20 users gathered over a period ranging from 5 to 10 weeks per user. All system generated file operation activity was excluded from the data set. The authors made use of the Windows *audit* tool to gather data from Windows users. Navigational information is gathered on all objects touched by a user. This includes *direct* file access where the user intentionally operates on a file, as well as *indirect* access where an application used by the user operates on a file. The information is represented as two graphs indicating access sequences as well as file structure depth.

In addition, the authors used a survey to gather age, gender, and title information from each user. More subjective information was also gathered in terms of what the authors described as:

> "how (un)tidy each user considers she keeps her working folders, how tidy she considers herself for organizing and carrying out general tasks in relation with her job, how computer knowledgeable she is, or how capable she is on customizing the underlying operating system"[61].

A group of information security students (under- and post graduate) were surveyed in order to derive scenarios. The students were posed with the hypothetical scenario of having access to an unattended computer. They were then asked to describe what they would do, detailing all steps. The authors developed three simulated masquerade attack scenarios based on attacker competence, *basic*, *intermediate* and *advanced*. Actions ranged from exploring interesting files in a target's My Documents folder to the automated copying of files (such as spreadsheets and images) to an USB drive. All masquerading attempts were simulated by one individual for each of the 20 users.

The WUIL data set does show promise for the use of new masquerade detection research. We consider the inclusion of user attributes such as role and age information to be very interesting. We believe this method can be valuable when analysing profiling results and help in creating more accurate groups and select better features for describing them. Gathering information regarding participant characteristics in the form of a survey, such as age, experience level, applications frequently used, times of days etc. can be another way of supplementing data with external information. In order to infer more accurate meaning from results it could be beneficial to corroborate with another data source. Alternatively, it can also serve as a initial phase for establishing how to group individuals.

In terms of the realism of masquerade data; how representative the students are of true masqueraders is unknown. It is certainly a good starting point could prove an interesting method for future data collection. The resulting scenarios are intuitive and quite basic, all of them fundamentally centring on exploring and ex-filtrating data from the computer.

3.5.3 Features

Depending on the scope of the profiler, different features will be selected. For example, when considering file access to a shared company file repository features relating to access in terms of time, file name, average file sizes, frequency, uniqueness and file type can be considered.

When considering user issued commands, such as Schonlau [54] and subsequently inspired research, features such as command sequence, time, frequency and uniqueness can be considered. This is also related to similar intent or action based outcomes such as web applications, where individual URL could be perceived as commands.

An opinion in [73] is that characteristics directly indicating user intent are more indicative than inferred characteristics such as the state of the CPU (utilization level, memory usage). The authors state that:

> "Interestingly, we found that the most useful characteristics for anomaly detection (and hence for user recognition more generally) were user-specific ones that directly reflect some user feature, such as typing habits and Web page usage, more so than application-specific features which only indirectly reflect user activity."[73]

Work on user identification in Section 3.3 noted the similarity of profile creation to text classification and the information retrieval domain. Furthermore, Zipf's law[3] has been said to apply to access frequency of web resources [74]. This theory was applied in [36] to motivate the selection of common text classification methods for user profiling. The results support the view that strong similarities with natural language processing exist. Parallels can thus be drawn between user profiling and text classification. The resource accessed or commands issued can be thought of as words in a lexicon. This relates to author attribution concepts[4], and thus lends nicely to user profiling for masquerade detection.

Bag of words

Bag of Words (BoW) is a common feature representation method for information retrieval, text classification, as well as image classification [75].

Document	Contents
1	The quick brown fox jumps over the fence
2	The slow red fox broke the fence

Table 4: Bag-of-Words Example: Documents

Dictionary	Index
the	1
quick	2
brown	3
fox	4
jumps	5
over	6
fence	7
slow	8
red	9
broke	10

Table 5: Bag-of-Words Example: Dictionary

[3]Statistical law regarding the distribution of word frequencies in linguistics
[4]Identifying individuals based on written texts

Document	1	2	3	4	5	6	7	8	9	10
1	2	1	1	1	1	1	1	0	0	0
2	2	0	0	1	0	0	1	1	1	1

Table 6: Bag-of-Words Example: Resulting Vectors of Documents

The basic concept of the method is as follows. A dictionary (see Table 5) is extracted from a corpus of documents (see Table 4). Finally, a vector representation of dictionary word frequencies per document is created, as exemplified in Table 6

Profiling can draw analogy to text classification when considering the commands issued by a user being synonymous with the user authoring a document. Each individual is authoring a document of commands, as illustrated in Figure 7. Different instances of these "documents", represented by BoW, can then be thought of as the profile of an individual or group.

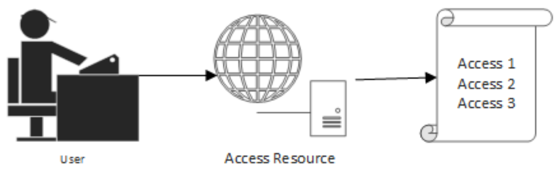


Figure 7: User access to resources (commands) likened to authoring a document

3.5.4 Support Vector Machine as a method

Support Vector Machine (SVM) is a popular supervised machine learning algorithm which has shown great promise in masquerade detection. It was originally designed to solve binary classification problems. [76] However, a number of robust multi-class implementations has been derived [77]. These methods commonly rely on one-to-one comparisons or voting methods to achieve the best classification accuracy.

SVM aims to find the best separation in training by constructing a hyperplane between two linearly separable classes. The separating hyperplane is formally defined as $w^T x + b = 0$, with:

- **w** being normal to the hyperplane
- $\frac{b}{\|w\|}$ being the perpendicular distance from hyperplane to the origin [78]

Support Vectors are chosen data points that are closest to the constructed hyperplane. Each class has Support Vectors indicating the class boundary. There are many possible hyperplanes that can be chosen to separate the data set, however SVM aims to achieve maximum separation. To do this it needs to maximize the distance from the hyperplane to each Support Vector. This distance is referred to as the margin [78]. The resulting optimizing problem can be reformulated as a minimization problem that can be solved by Quadratic Programming optimization using Lagrange multipliers. Figure 8 shows the separating hyperplane with the maximum margin in linear data [79].

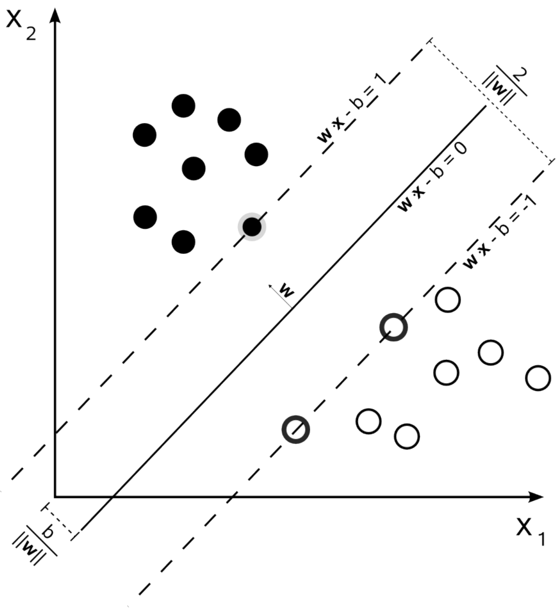


Figure 8: SVM Maximum Separation of Hyperplane With Margin - Linearly Separable Example. Figure from [80].

To overcome restrictions of data not being linearly separable it projects data into a higher dimension where separation can be optimally achieved. Figure 9 demonstrates how non-linear data can be separated by applying a kernel transformation function. This does add to the computational complexity of the classifier. To minimize this, SVM uses a kernel function that can be represented in terms of the original data, and thus can be more easily computed [81].

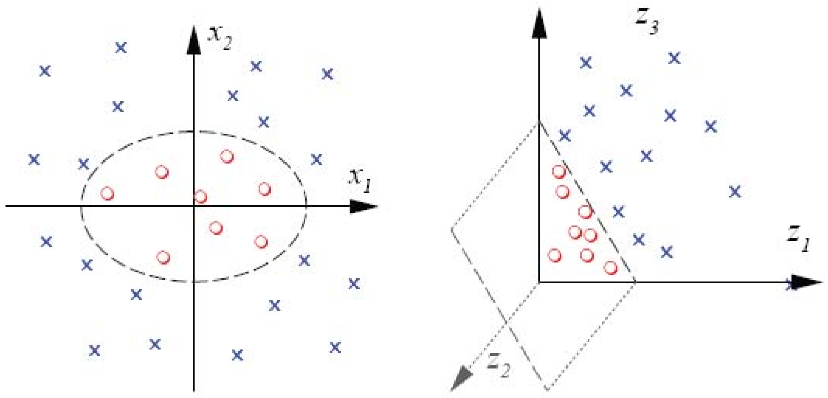


Figure 9: Example of Applying Kernel Mapping to Non-Linear Data. Figure from [82].

A penalty threshold can be defined and adjusted in terms of cost C. The number of data samples bordering the margin can either be included or excluded from calculating the hyperplane optimization. This is to avoid over fitting the solution around outlier data.

One-Class SVM

A one-class implementation of SVM (ocSVM) was presented in [83]. Unlike the original binary SVM implementation that requires positive and negative class examples for separation, the ocSVM implementation only requires examples from one class, typically positive examples. ocSVM uses a kernel function to transform the data set into a higher dimensional space. It then uses the origin as the only example of the negative class and maximizes the hyperplane distance to the origin. The original negative training examples forms a dense distribution further from the origin, a function is estimated which determines the probability of a new example belonging to the training distribution or to the negative class.

3.5.5 Challenges

Masquerade detection methods face various challenges by having to accommodate for dynamically changing behaviours and environments. Natural changes in behavioural patterns occurs as users transition between tasks and phases in of their roles. Certain periods might have a higher frequencies of certain tasks.

Incremental learning methods have been mentioned as research considerations in Section 3.5 where the behaviour model is updated to accommodate such changes in the environment. However this adaptive capability also opens the system up to the risk of malicious behaviour being updated into the profile. A commonly referenced technique used by masqueraders to avoid being detected is the mimicry attack [84]. Masqueraders use stealth to hide malicious actions amongst legitimate actions whilst remaining below the detection threshold. Slowly incrementing the use of malicious actions the normal model becomes tainted.

On the other hand; [60] takes an interesting approach that leverage changing behaviours in order to detect anomalies. The authors group users from similar role domains together, for example engineers and sales people. They then apply a clustering method. They consider that the different groups should remain clustered together over time. If policy changes it should affect all users in the group equally. It is considered an outlier if at one point a user is assigned to a different cluster than the original.

By considering a group of users, this malicious behaviour creep should be able to be detected when comparing the shift of individuals within the group.

4 Methodology and Framework for Masquerade Detection

This chapter discusses the theoretical methods used, and defines the detailed framework used for the applied masquerade detection methodology. The resulting framework is as modular as possible to encourage future research. Motivations for choosing specific methods is discussed and is based on topics introduced in Chapters 2 and 3. The selected method for each framework component is detailed in a practical sense. This is to conretize the specific instantiation of the framework used in this thesis, as there are many alternative options for each component. The technical details of the implementation is expanded on in Chapter 5.

Defining a framework is a crucial step of any research methodology. The reasoning is two-fold, firstly assisting to conceptualise the problem and solution space relating to the area of interest. Secondly, it allows for the presented work to be more easily digested, reused and evolved for future research work. The goal of this framework is to allow for experimentation on profiling within the context of masquerade detection. It aims to provide the theoretical structure as well as the technical means to achieve this. Figure 10 contains an overview of defined framework components.

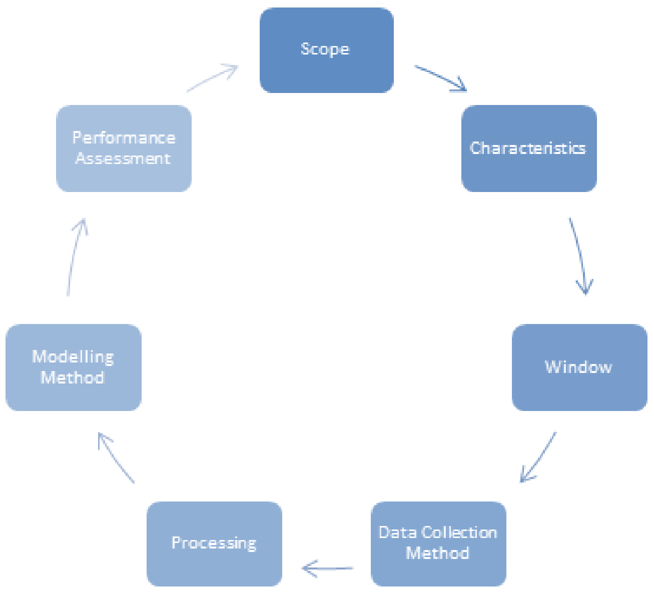


Figure 10: Framework Components Overview

4.1 Define Scope / Context

As a first step, the scope or context of the environment in which the detection mechanism is to operate has to be clearly defined. One also has to determine what has to monitored in terms of:

- objects,
- subjects and
- actions

This follows the approach as to what was originally discussed in [20]. One also has to select whether it will be limited to a simulated environment or real/authentic environment.

As introduced in Section 3.4, we believe that well selected context can add sufficient domain knowledge that would benefit detection capabilities. By limiting ourselves to isolated components of a system we can more accurately determine a baseline for normal behaviour. Resulting abnormalities can then in turn be combined from various sources, and individual scores correlated to infer further knowledge.

Peers for comparative power

When using a group based profiling approach, as is the primary focus of this thesis, the context component needs to define a group of users that are similar in an inherent or latent way. Each defined group constitutes a single context. This allows for the comparison between peers of the same group. Profiling strategies were discussed in Section 3.3 and 3.5 where arguments for the benefits of group profiling were highlighted. We consider the most important point to be that a group can serve as a reference point to compare individual behaviour against. Effectively reducing the complexity of the environment by focusing on smaller areas. This concept is similar to the Peer Group Analysis (PGA) technique [34].

The literature study revealed that the use of well defined roles and policies in an organization serving as a context for effective group profiling has not been extensively researched. Also, there are specific studies as to what level of granularity contextual data should be considered at. For example, a combination of specific commands within an application on a lower level, or merely access to resources at certain times on a higher level of abstraction.

Selected Method

We consider two primary contexts, a group of users and a shared resource. Adding contextual information in the form of grouping labelling and resource labelling. We chose to develop a scenario from simulated domain knowledge around the following:

Objects

A web-based application was chosen as the primary scope for the experiment. This decision was motivated due to the proliferation of such systems through organizations in the form of knowledge sharing portals, intra-nets and accounting platforms. They are also relevant when personal interactions as most activity a user has is through a web portal of some form, be it for communication through email, socializing, education or general entertainment.

Subject

Highly homogeneous group of users, operating in a well defined environment, were selected in order to establish a reliable ground truth.

Actions

A decision on which interactions to monitor was based closely on previous work in masquerade detection. Discussions in Section 2.3.1 found that most research has been performed using command and resources access data. HTTP access can be likened to a command being issued or to a resources begin accessed. Thus the selection can be interpreted as either.

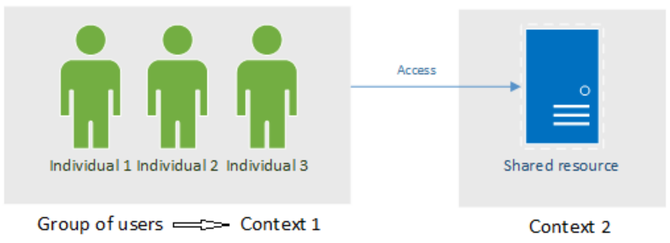


Figure 11: Overview of scope / context

4.2 Define Characteristics / Features

A set of descriptive characteristics have to be selected to represent a profile. Selecting appropriate characteristics is paramount to the success of creating a behavioural profile (3.3). The challenge is to capture the essence of behaviour relating to how the problem is framed. As seen in Section 2.4.1 and 3.5, characteristics can be categorized as *sequence* or *frequency* based. Furthermore, temporal aspects such as timing intervals between observed interactions can also be considered. Temporal characteristics can prove valuable for both individual and group profiling. In the group profiling case it could be used to profile user groups of different experience level or age.

Features can be selected either by intuitive reasoning, expert knowledge or by applying feature selection methods. Some examples are:

- Temporal data, data transmission times
- Source and destination IP addresses
- Application layer specific information, such as HTTP headers, request types etc.
- Packet payload information

Selected Method

Features were based on previous work, discussed in Chapter 2, as well as expert knowledge of what would be indicative of user behaviour that could be generalized for group comparison. No feature selection methods were used. We decided to group individuals based on the concept of similar roles and work related duties and as such required abstract characteristics. We aimed at profiling user intent at a high level, acting as a mechanism to isolate common group activity frequencies. We have selected the bag-of-words method, which has been used successfully in related work [65], making it a viable selection for this thesis. The use of more simplistic nature of this method is supported in [85]. The author demonstrates that better results can be obtained than more advanced techniques which require more processing resources. Temporal characteristics were not used due to the intuition that timing features are very nuanced, favouring individual behaviour. As dicussed in previous work, [73] strengthens the view that characteristics directly indicating user intent are more indicative than inferred characteristics.

4.3 Define Window Period

In order to create a robust model of user behaviour a metric across a time period in which behaviour is considered has to be defined. A single connection does not offer enough information. Therefore, individual observation instances have to be grouped together. This is a decision on how much observations to consider as a session. It directly affects the detection capabilities of the selected detection method in terms of what length of activity it considers when generating an alert. Selecting too small a window could increase False Positives. The core distinction between methods is to consider to performing anomaly detection on the entire window, an all-or-nothing approach. Alternatively the use of a score and threshold method to independently classify each observation and combining the final score can be used to determine the detection outcome. The size of the window can vary, being dependant on the specific security requirements for a specific resource.

A window can be defined over a time period or over a number of commands. Windows can also either be sliding, incrementally including each new observation, keeping the last *n-1* observations. Or distinct windows where each window is *n* new observations. Sequential methods, such as HMM lend themselves well to defining window lengths, as seen in Section 3.5. Alternatively, window representations can be built by combining individual observations into a single feature vectors.

Selected Method

A session is artificially created by grouping *n* amount of commands performed together. BoW discussed in Section 3.5.3 acts both as a feature representation method, as well as a defined window size. The number of words, or accesses in this case, determines frequency counts taken in one window.

Initial work performed as part of this thesis was focused on using the Bro *SumStats* framework, introduced in Section 3.2. Observations of data were made in a temporal window opposed to frequency window. However, deciding on the BoW feature representation method required that a feature vector comprised of a constant frequency window be used.

4.4 Define Data Collection Method

Once the scope and features have been defined, the observation source has to be determined. Interactions can be observed and monitored from the perspective of the subject or the object, or through an intermediate point. This can be exemplified as monitoring:

- user's host computer
- a server
- network traffic flows

The granularity of observations, determined in Section 4.2, will also impact the collection method.

Practical experimentation considerations can include: The decision to use an existing data set or to acquire one yourself. Depending on the type of data required, acquiring data is heavily reliant on the current stance on privacy.

As seen from Section 3.5, host and server data is usually collected by using built-in audit logging functionality provided by the operating system. Other methods involve custom developed sensors that observe events generated by the operating system. Client side information, such as a list of running processes and logged on users. Observing data directly on a host also allows overcoming encryption of payload information. Another approach is to develop a small lightweight process monitor to feed information to an IDS. However this can cause synchronisation problems, and be difficult to implement due to project time constraints. Bro has a user agent called Broccoli (Bro Client Communications Library), that supports scripting through Python and C++. Broccoli takes care of the communication between the host and the IDS, which saves time and reduces the possibility for synchronization errors [86].

For network flow data collection there are a number of tools, called packet sniffers, that allows collecting data, such as Wireshark, tcpdump, Microsoft Network Monitor or Bro. Most tools share the libpcap library in order to capture packets directly from the network [87]. This means that most captures contain similar data that can be compatible. The use of packet sniffers requires one to be on the same network segment or be connected to a SPAN port on the network switch [88]. Alternatively, in simulated cases, when using virtual machines on one host machine one is able to easily perform sniffing of all virtual hosts due to common shared resources. All traffic running across the network will be visible, including low level packet statistics and payload information.

Selected Method

The basic data set requirements are defined in terms of the following criteria:

- Group labels
- User labels
- Coarse granularity of interactions (a high level of abstraction in terms of user intent)
- Commonality of interactions, such as shared resources being accessed

The performed literature study revealed that no data sets contained labelled user group data. However, the generation of user profile based data sets were described in [10]. Various existing data sets have also been observed in Section 3.5, none of which met our criteria. The WUIL data set described in Section 3.5.2 did provide attributes from which grouping could be derived, however the data set was more suited for individual profiling as there was no shared resource.

For this reason a simulated environment is implemented as part of this thesis in order to obtain relevant data. A simulated work environment allows for continued and iterative testing that would be afforded on a production network. This ensured a controlled environment where data observations could be labelled according to our criteria. For this project we chose to use two different approaches. For preliminary data capture testing and data analysis we will use Wireshark. The decision comes down to ease of use, saving time during the initial phase of project. Wireshark is also portable and allow for rapid testing. This was mainly be used for preliminary analysis for feature extraction.

Further stages make use of Bro for data capturing in order to provide a more cohesive solution and ensure full data compatibility in subsequent profiling and anomaly detection phases. By performing final data captures through one system more assurance can be provided that it will be in the correct format.

Bro motivation

It is clear that one faces many difficulties when profiling behaviour based on the collective network information. One aspect is the large amount of noise that is associated with these methods. Bro's unique event driven nature provides the high level user intent relied on for profiling. The use of Bro can allow for rapid changes to extract features from a vast myriad of network resources, file, http in varying levels of granularity. Bro has a powerful scripting language that suited the project by being able to quickly make changes to the Proof-of-Concept. Bro can also draw additional information from hosts by using the Broccoli client, however this functionality was not used during the experiments.

4.5 Define Processing

Once data has been collected it needs to be processed in order to represent the characteristics defined in Section 4.2 in a way that profiling can be applied to it. In most cases log files are generated by the data collection phase, and different processing techniques are applied to extract latent information from the observations and constructing the final feature vector. This can equate to feature selection and feature extraction exercises, introduced in Section 4.

Data parsing can also be performed if different sources of log files are used to transform data into a standard format. Normalising data and converting data types in accordance with the requirements of the modelling method being applied.

Selected Method

A combination of live feature extraction using Bro scripting, and off-line pre-processing using a Python script were selected. The only downside considered to the approach is that changes to the Bro script requires reanalysis of the entire network capture. To overcome this we perform testing and script tuning on a smaller test dataset.

4.6 Define Profile Modelling Method

As the no free lunch theorem states, there is no single "best" solution that performs equally well for all different problems [89]. It is crucial that one compare the results of various modelling methods against the domain's unique data. The selection of method, *non-parametric* vs. *parametric*, will depend on various factors such as prior knowledge about the properties data population. The amount of data available also becomes a crucial factor. To accurately determine a distribution some non-parametric methods require a large amount of data [90]. Without prior knowledge initial stages should include exploratory efforts to understand a sample of the data population. Options such as unsupervised machine learning techniques, for example *k*-Means clustering, can be considered. These methods do not require labelled data and is usually computationally efficient.

Statistical profiling

Basic statistical methods typically involve measuring deviations from a defined threshold, such as standard deviation. Such methods generally rely on knowledge or assumptions of the underlying data distribution. The primary motivation for using basic statistical methods is performance. Statistical methods are mostly computationally efficient such as detailed in Section 3.2.

Classification based profiling

Section 3.5 highlighted that machine learning methods were the most common approach used in profiling for masquerade detection. One must however also be cognizant of the limitations involved. [53] discusses the effectiveness of machine learning techniques in anomaly detection, and it is clear that some challenges are faced in this area. However it mostly stems from naive usage without taking proper consideration of the problem space and the specific purpose of the technique.

Selected Method

No prior assumptions regarding the distribution of the observations were made. A method that would operate well under this requirement was selected. Machine learning and data mining techniques as detailed in Section 3.5.4 are primarily used. SVMs suits our primary requirement in that it makes no prior assumptions regarding the distribution of the data population. This creates a more robust model that does not produce too many classification errors based on incorrect distribution assumptions. In addition, models generated by SVM classifiers have been shown to be highly generalizable. This nearly eliminates risk of over-fitting a model based on limited training data [91]. The benefits of using this method has also been discussed in Section 3.5.1[Wang et al]. SVM has been designed to handle high dimensional feature spaces, and is therefore suitable to use against the BoW feature representation method.

The profiling problem can be formulated as a classification problem where each instance $x \epsilon X$ is defined by a feature vector construction in a constant period $p = window$. The period is defined by the length of observations of accesses made by a user. Each class, $y \epsilon Y$ can either represent an individual user, or a group of users.

User and group labels are preassigned. If an observed classification does not match to this label it can be considered to be an anomaly. We also differentiate between a one class classification problem (group profile) and a multi-class classification problem (user profiles, multiple groups).

Multi-class depends on a balanced population. Since it is normally very difficult to collect realistic masquerade data, an assessment is made of whether one class outlier methods are effective. However, by considering other groups as the differentiating factor we can use multi-class methods. The assumption is that in a work environment there will be an adequate number of groups in order to provide a robust model for each individual model. A user belonging to one group incorrectly classified as belonging to another group will then constitute an outlier.

4.7 Define Assessment of Performance

Once the experiment has been completed and all data collected, analysis of results has to be performed. This can be done using classification or detection error rates. Chosen measures are determined by the approach taken.

Thresholds

Depending on the profiling method, detection performance can also be affected by defined thresholds. Different measures can be used to determine when an anomaly is considered as such. This is based on a similarity measure inherent to your model. The ability to adjust the threshold is crucial to control the level of trade-off between detection and false positive rates. Thresholds based on observations, in the case of supervised learning is greatly determined by the dataset or observations. Defining the baseline threshold for normality differs for each confined context. If unsupervised methods such as clustering approach has been used thresholds can be established based on the quality of the resulting clusters.

An acceptable threshold has to be determined for each unique environment in line with specific security requirements. Intuitively highly homogeneous contexts should have a very small threshold for errors.

Selected Method

Quantitative methods were used, introduced in Section 2.3.2, based on results obtained from experimentation. This allows for easy interpretation by other researchers. It can also be used to compare re-performance of the results.

We used a classification approach to establish profiles. Thus we consider the accuracy of classification results in terms of:

- False Positive Rate
- False Negative Rate
- True Positive Rate
- True Negative Rate
- ROC curves for visualization of metrics

We do not consider impacts from a threshold for our experiment, as the SVM method used has been configured to output discrete classification values.

5 Experimental Design

This chapter describes the developed Proof-of-Concept implementation. It has been designed following the framework principles set forth in Chapter 4. The primary components consist of a controlled environment for data gathering and a processing environment for data analysis.

5.1 Overview

The experiments can be divided into two phases, data capturing and processing (sub phases Training and Testing). The designed scenario fits the framework defined in Chapter 4. Figure 12 shows the process flow of the experiment. Session length is defined as every 10 resource accesses.

5.2 Scope / Context

As discussed in Section 4.1, the context encapsulates the specific focus of where the masquerade detection capability is being applied to. In order to simulate a typical area of interest we opted to use a web based accounting application, NolaPro®[1]. The system allowed for deployment to a local web server which suited the virtual lab configuration and enabled easy monitoring of user interactions.

A detailed description of NolaPro® is not be provided, as it only serves as a generic system for the user to interact with. The experiments do not aim to capture any information that is specific to NolaPro®, but instead captures generic HTTP requests. This means that the system can easily be substituted with another, perhaps more relevant system.

5.3 Participants

The experiments were designed to be as easily performed as possible in order to get reliable results from participants. This also served to help participants complete their tasks, and not be too challenged and thus opt out of the experiment.

Users were assigned to two groups. In order to simulate a work environment we assigned each group a specific set of tasks (See Appendix C). We considered this to be the most feasible approach, opposed to monitoring real systems under operational use which could put user's privacy at risk.

Participants were fellow students who volunteered to partake.

5.3.1 Scenarios

Two distinct scenarios were developed. Each containing artificial tasks performed by subjects as part of the data gathering exercise. Accounting roles were used as an example as they would be commonly found in most organizations.

[1]http://www.nolapro.com/

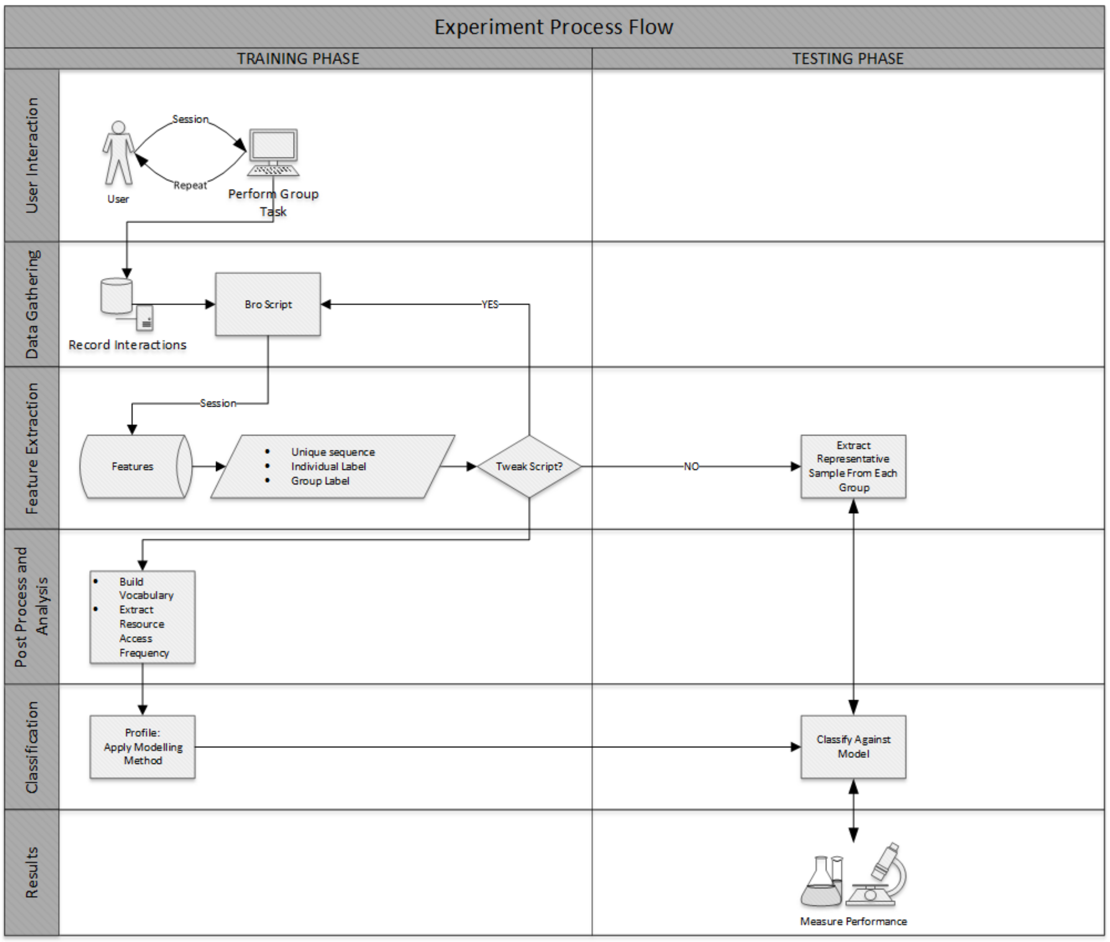


Figure 12: Experiment Process Flow

Users were given a small introduction to the system and asked to perform tasks according to their roles. They were given an overview of their role and guidance on how they should perform their tasks. However they could determine their own sequence of actions.

Participants had no prior knowledge of the system.

Group 1

This group can be likened to a simulation of experienced users who are performing well practised tasks. Users' roles were to maintain a client database.
Tasks to be performed were limited to:

- Create new clients
- Delete existing clients
- Update existing client details

Group 2

Group 2 was to simulate anomalous behaviour (not necessarily malicious). This was achieved by allowing the user a wider selection of tasks. However, an alternative interpretation can be that the group simulates inexperienced users who are exploring a new system.

The users were essentially given free reign on the system, and were asked to sporadically access any part of the system as they wished.
High level activities included, but were not limited to:

- Explore system functionality
- Insert *any* new data entries
- Delete *any* existing data entries
- Update *any* existing data entries

5.3.2 Privacy

By utilising a simulated environment there is a very limited risk in violating user privacy. Each user is also assigned a generic username, and no identifiable information is collected.

5.4 Virtual Lab

To collect the needed data an environment was built based on Oracle® VirtualBox virtualization software, hosted on a single physical workstation. This served as an infrastructure layer that hosted a combination of Windows® and Linux® virtual machine instances. Users were given remote access to the environment, each individual were issued with unique authentication credentials. VirtualBox's virtual network configuration tool was used to simulated a shared network. Promiscuous mode was also enabled allowing Bro to capture all network flowing on the virtual network segment.

Software version information is listed in Table 7.

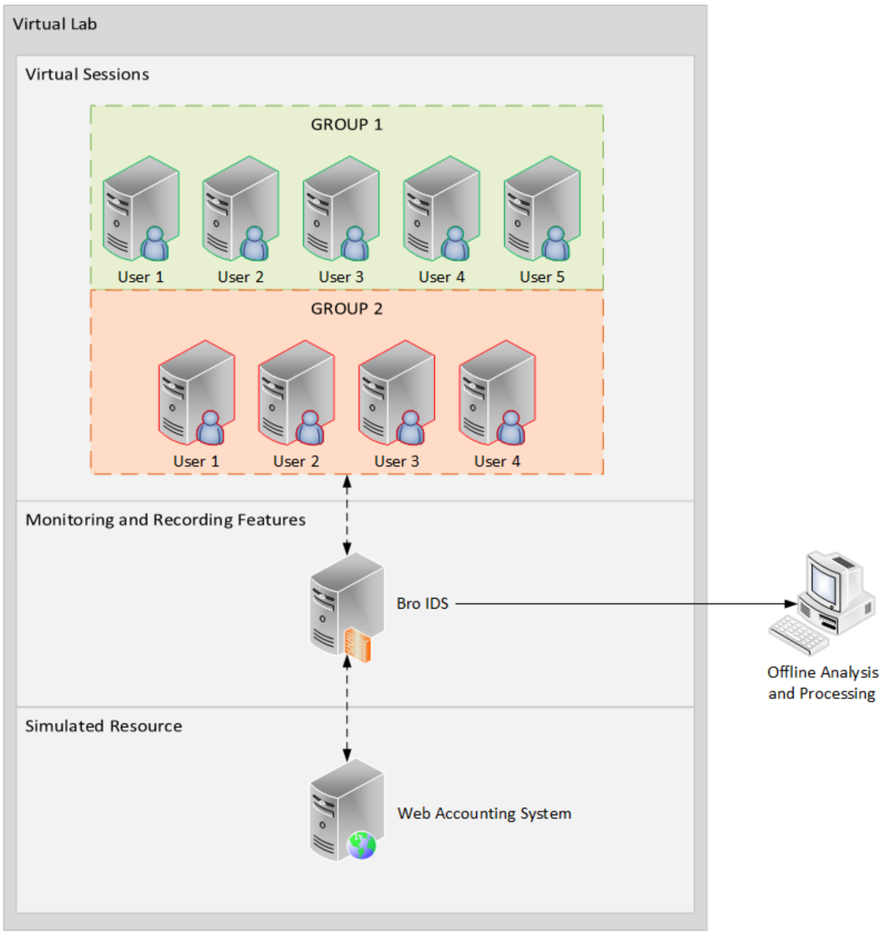


Figure 13: Controlled Environment

Workstation Host:	*Windows® 8*
	Oracle® VM VirtualBox 4.3.8
Virtual Workstation Sessions:	*Windows® XP*
	Mozilla FireFox® 29
Network Monitor:	*Ubuntu™ Server 12.04*
	Bro 2.2
NolaPro® Web Accounting System:	*Ubuntu™ Server 12.04*
	Apache™ 2.4, PHP5.2, Oracle® MySQL™
	NolaPro® 5.0.10654

Table 7: Virtual Lab Software Versioning

Figure 13 shows the layout of the controlled environment in which the experiments were conducted.

5.5 Analysis Platform

Primary data analysis was performed on a standalone machine using the WEKA[2] data mining package and Python to allow for rapid prototyping. Version information is detailed in Table 8.

Analysis Workstation	*Windows® 8*
	Weka 3.7
	LibSVM 1.0.5
	Python 2.7

Table 8: Analysis Platform Versioning

5.6 Data

5.6.1 Labelling

Individual users are labelled according to their IP address. Each address has been statically assigned to a single user, ensuring accurate collection of their data. Each group has unique IP range as per Table 9.

Group	**User**	**IP / User Label**
1	x	192.168.1.20x
2	y	192.168.1.10y

Table 9: User data labels

5.6.2 Collection

All interactions between users and the monitored resource were gathering using Bro IDS and the developed script, included in Appendix A.1. The script acts as a filter, isolating only the data relevant for our study, as well as applying pre-processing to data to obtain a preliminary feature set. Functionality was implemented to collect the following feature sets:

1. HTTP access statistics using SumStats during variable temporal windows
2. Sequential HTTP access in the form of n-grams. Each gram being a URI access.
3. Bag-of-words (BoW) frequency count

Number 1 and 2 have been used during preliminary experimentation and allows for future extensibility. The final experiment only made use of the BoW feature set. The script can be configured according to the environment through the following configurable parameters:

- Server to monitor
- User group assignment
- Length of sequence to record (the number of words in the bag)

[2]http://www.cs.waikato.ac.nz/ml/weka/

The script performs actions based on the *http_message_done* Bro event. This event is generated for each HTTP request that has been processed by Bro as follows:

> "http_message_done Type : event (c: connection, is_orig: bool, stat: http_message_stat) Generated once at the end of parsing an HTTP message. Bro supports persistent and pipelined HTTP sessions and raises corresponding events as it parses client/server dialogues. A "message" is one top-level HTTP entity, such as a complete request or reply. Each message can have further nested sub-entities inside. This event is raised once all sub-entities belonging to a top-level message have been processed (and their corresponding http_entity_* events generated)." [92]

The primary data collection logic that happens when a user (defined by host IP) accesses the monitored resource, is illustrated in Algorithm 1.

Definitions of the following terms are required to discuss the script's implementation:

- *Monitored resource:* The IP address of NolaPro® server
- *Resource URI:* URI portion of URL being accessed on monitored resource, as per Table 10
- *Host:* IP address of user, filtered on valid preassigned group labels
- *URI Index:* A mapped index number assigned to each unique URI access
- *Sequence Vector:* Collected for n subsequent connections established by a host in the form of URI Index
- n: the size of observation window in terms of the bag size, default 10

Full URL	**URI**
http://192.168.1.4/nolapro/index.php	*/nolapro/index.php*

Table 10: URI portion of URL that is indexed

Algorithm 1 Feature Extraction Pseudo Code

```
1: on http message done do
2:     Resource URI = tokenize(URL)
3:     for each host -> resource access observation do
4:     if (host has been seen) and (resource accessed is being monitored) then
5:         assign index number to Resource URI
6:         add index to vector until n limit researched
7:         output user and group labelled vector sequence to file
```

The system has to maintain a list of index values assigned to each unique resource URI. Intermediate sequence pattern for each host is kept, with a maximum size in memory of n per host, once output to a file it is reset.

An example of resulting data format is shown in Table 11.

User Label	Group Label	*Indexed Sequence Pattern of length n*										Timestamp
192.168.1.203	1	0	0	1	1	1	1	1	1	1	2	1396962893.730735

Table 11: Bro data output

URI Parameters

HTTP parameters are usually separated by a special character such as a question mark. Depending on the web application this could have different significance. Considering all parameters in feature extraction process could potentially generate a large number of unique records. An example is the same action could be parametrized by a random session number. Thus this could skew the profile by essentially considering the same action as different. For this reason a decision was made to exclude parameters and only include the root URI resource at this stage [93]. Tokenization has however been implemented in the Bro script to separate parameters, and thus allows for any number of parameters to be considered.

5.6.3 Post-Processing

Once final data output has been obtained from Bro post-processing was performed to obtain the Bag-of-Words feature vector, as discussed in Section 3.5.3. For this task a script was developed in Python, included Appendix A.2. The logic is presented in Algorithm 2. The script analyses data input, finding the highest index value, *fn = max(Indexed URI)*. A second phase of processing tokenizes[3] the sequence pattern attribute, creating a frequency count, *n*, for each individual Indexed URI, *f*. The resulting output is a sparse vector representation of access frequencies across the defined observation window (10 by default). The Bag-of-Words method disregards sequential information. In order to retain this attribute to differentiate between observations a concatenation of each sequence pattern was made.

Algorithm 2 Feature Post-Processing

```
1: for all observation entries do
2:     feature vector n x n = tokenize(observation)
3:     sequence vector = tokenize(sequence feature vector n)
4:     fn = max(sequence vector)
5: for all sequences entries do
6:     inc(index[sequence])
7: output feature vectors to training file
```

[3]Splitting a string into separate components

The final output of one observation sequence is exemplified in Table 12.

				Index Frequency Vector					
User Label	**Group Label**	**Sequence Pattern**	**Timestamp**	***f1***	***f2***	***f3***	***f4***	**...**	***fn***
192.168.1.203	1	11111112	1396962893.73074	2	7	1	0	...	5

Table 12: Post-processed Data output

Further processing is performed in WEKA during the import stage to ensure that data attribute types are imported correctly. Once data has been successfully processed it was exported in as an .ARFF[4] file for the modelling phase of the experiments.

Finally, depending on the classification task, either the *User Label* or the *Group Label* were removed from the data set to avoid bias.

5.7 Modelling

WEKA was used in order to perform the classification task as defined in 4.6. The LibSVM wrapper was used with a Radial Basis function Kernel. The parameters used can be see in Appendix B, Figure 22.

Modelling steps included:

- Import prepared data set $\mathcal{X}$ into WEKA
- Apply LibSVM C-SVM / OC-SVM classifier
- Apply 10-Fold cross validation
- Export model
- Export results

LibSVM can provide output in terms of probabilities of class membership. This can be used to adjust the detection threshold, however we used the discreet output to assess performance. Both C-Support Vector Classification and Distribution Estimation (One-class SVM) formulations was used as implemented in LibSVM [94].

[4]Attribute-Relation File Format used by WEKA

6 Results and Discussion

This chapter presents the findings resulting from our experiments. Results are presented first, followed by a discussion around our interpretation thereof. The results represented is based on our data set, discussed in Section 5. We gathered interactions from five different individuals. The individuals were assigned to different groups during the experiment period. Tasks were assigned to them as per the group scenario.

6.1 Data Set Analysis

Prior to feature extraction the dataset consisted of a network trace file of 485 MB in size. Features were extracted by running the Bro script on this data set in an off-line manner in case smaller tweaks to the script had to be performed. Runtime for off-line processing was approximately 5 minutes. However, on-line feature extraction can be done in real-time. Figures 14 and 15 shows the number of instances that has been extracted from the user interactions. Each instance consists of ten sequential resource accesses. To assess the distribution of access to different resources for each group a histogram and binned the index ranges in groups of 20 was used. For comparison, the scale of each histogram has been normalised by the number of instances in the group. Figures 16 and 17 shows the normalised histograms for both groups. The normalised unbinned figures are also included from which we can estimate the probability distribution of the sample.

6.2 Experiment 1: Individual Profiling

Experiment 1 presents the performance of individual profiling task. This serves as a comparison with the group profiling task performed in Experiment 2. However, it should be noted that there are data population disparities between the individual and group data distributions. We still believe that it offers the benefit of perspective when considering group profiling results. It also provides a baseline for individual profiling performance based on chosen method and dataset.

MultiClass SVM

Only MultiClass SVM results have been included for the profiling of individual users. Table 13 shows the performance metrics for each individual with the best performance highlighted and asterisked. Overall, performance is comparable to that achieved in related studies.

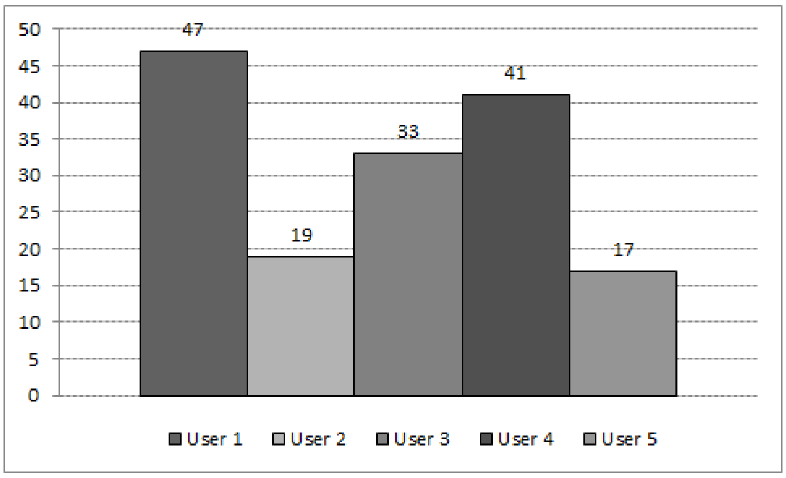


Figure 14: Individual instances recorded for Group 1

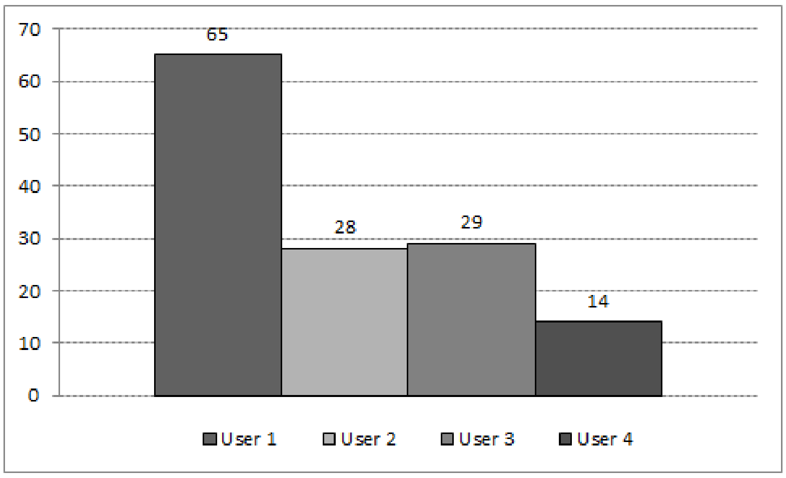
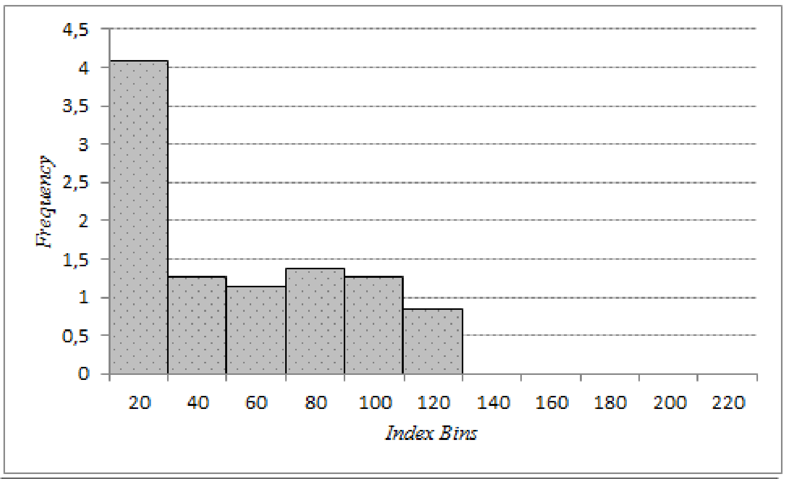


Figure 15: Individual instances recorded for Group 2



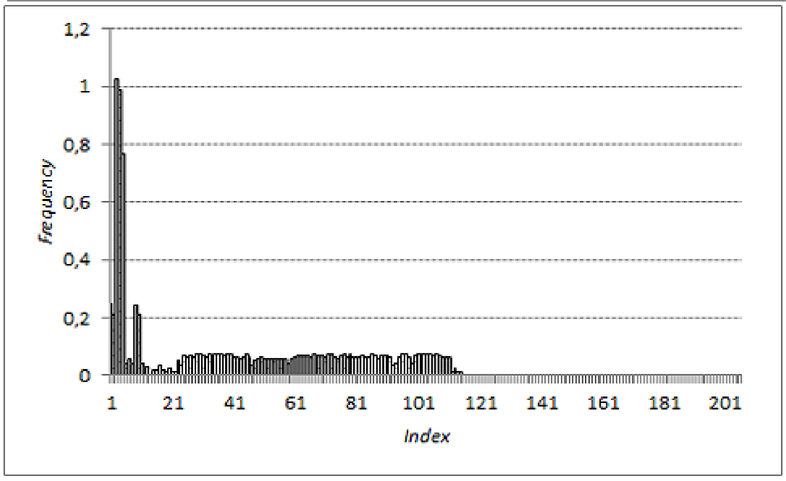
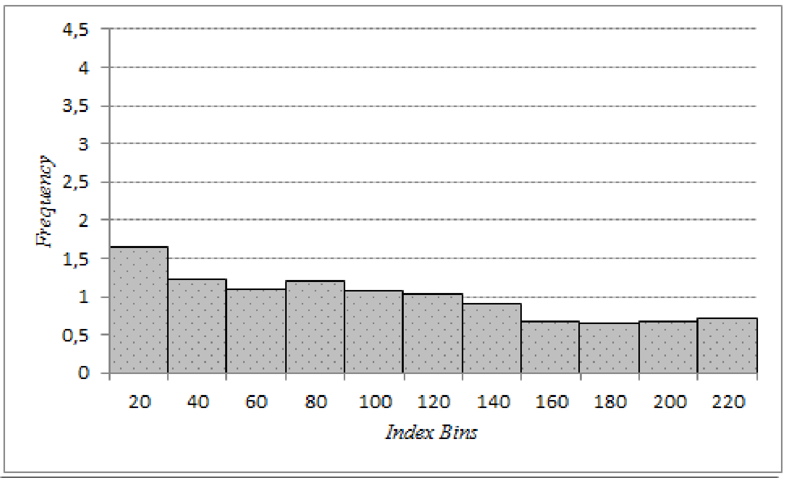


Figure 16: Resource access histogram for Group 1



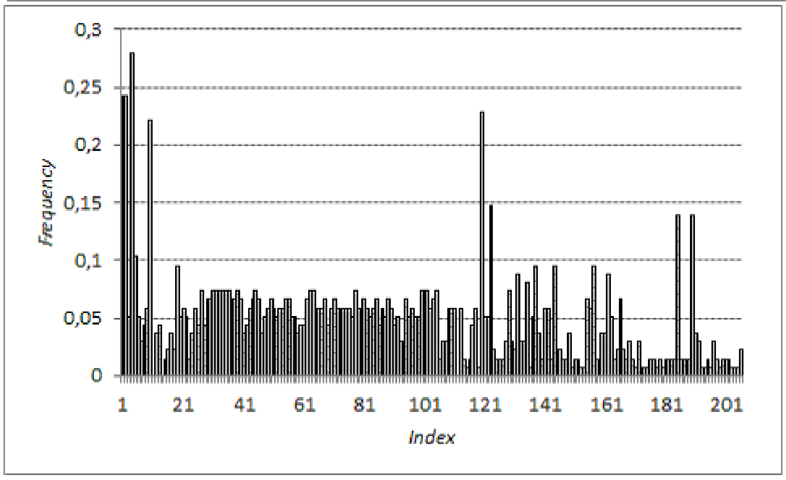


Figure 17: Resource access histogram for Group 2

True Positive Rate	False Positive Rate	ROC Area	Class / User
0.969	0.039	0.965	192.168.1.101
0.893	0.000	0.946	192.168.1.102
0.690	0.015	0.837	192.168.1.103
0.929	0.004	0.962	192.168.1.104
0.809	0.004	0.902	192.168.1.201
0.579	0.040	0.769	192.168.1.202
0.818	0.027	0.896	192.168.1.203
0.976	0.008	0.984	192.168.1.204
1.000	0.014	0.993*	192.168.1.205

Table 13: Individual profiling performance

Figure 18 shows the performance in terms of Area Under the Curve (AUC) values each individual. User *192.168.1.202* and *192.168.1.103* achieved the lowest overall accuracy, but still achieved a fair performance result. We could argue that this is due to having lower observation examples. However we can not make a definite correlation between instance frequency and classification results when comparing the two.

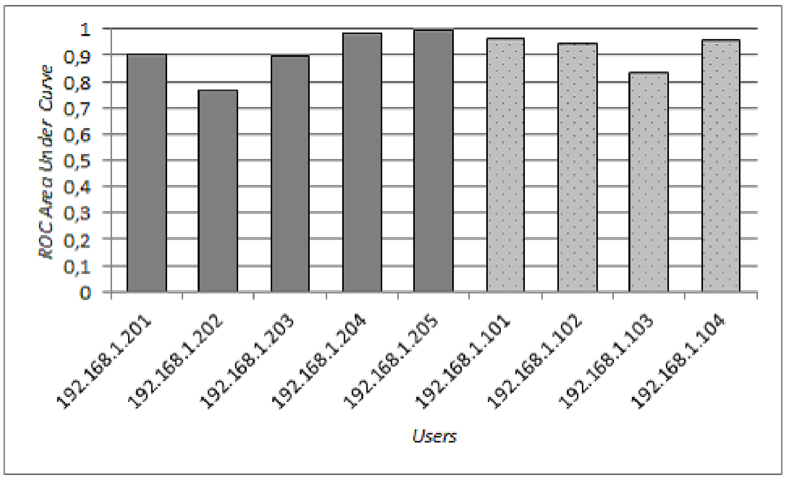


Figure 18: Area Under the Curve for each individual user

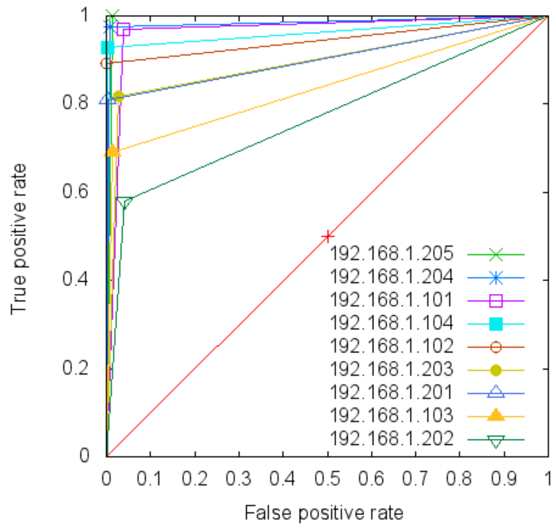


Figure 19: ROC for each individual user

6.3 Experiment 2: Group Profiling

Experiment 2 presents the results achieved during the group profiling task. It can be noted that the observation examples are nearly equal for both groups. Both MultClass SVM and One-Class SVM results were included.

MultiClass SVM

Table 14 shows the performance achieved by using SVM by applying 10 fold cross validation. Both groups achieve very high accuracy results.

True Positive Rate	False Positive Rate	ROC Area	Group
0.994	0.007	0.993	1
0.993	0.006	0.993	2

Table 14: Group profiling performance

Figure 20 shows both groups achieved the same Area Under the Curve performance.

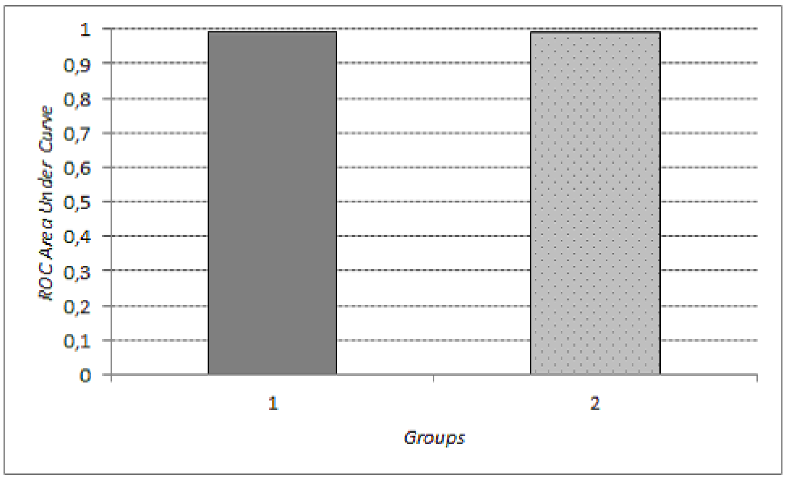


Figure 20: Area Under the Curve for each group

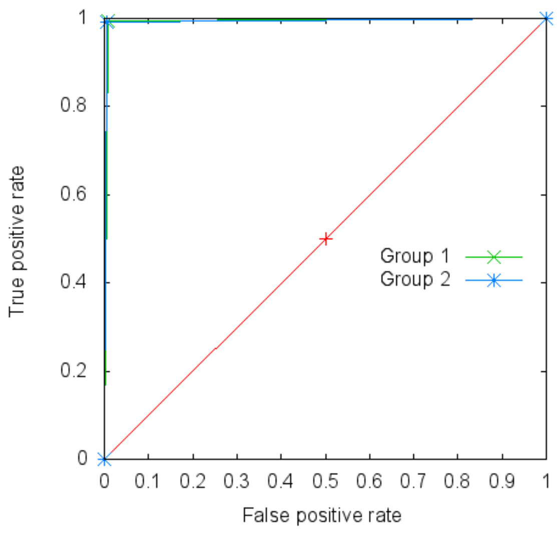


Figure 21: ROC for each group

One-Class SVM

One-class results are expressed in the rate of correct classifications and the rate of unclassified instances representing outliers, or potential masquerader attempts. The model for each class was built using positive observations, 10-fold cross validation was used for testing. Table 15 shows the results. When testing with the opposite class as a test data set all instances were correctly unclassified. The achieved performance is considerably lower than that achieved using MultiClass SVM.

Correctly Classified %	UnClassified %	Group
24.20	75.79	1
30.88	69.11	2

Table 15: One-Class SVM Group Results

6.4 Discussion

As expected from the results achieved in related research, discussed in Section 3.5.1, the application of Support Vector Machine performed well in our profiling task. The benefits of SVM were highlighted in Section 4.6, and our assumption was proved correct in the results achieved in our limited study. Overall results indicate that the selected features and methods for gathering them are suitable in our experiments, as posed in *RQ1*, *RQ2* and *RQ3*. Computation complexity has been raised as a concern, however we feel that masquerade detection is a less intensive process than other low level intrusion detection approaches. For this reason we find SVM well suited for the task. In addition, detection windows, discussed in Section 4.3 can have further impact in lowering computational requirements.

Individual profiling results showed some variance of accuracy in different users. This could be due to imbalances in individual observations or how individual users performed their tasks. Although no correlation beteen the number of observation samples and classification accuracy was visible from the results. With the best and worst performing profiles having the same number of observations.

The group profiling experiment obtained near-perfect classification results. This could have been influenced by the artificial nature of our defined groups. Purposefully designed in such a way to analyse the potential differences between individual and groups given a well defined context, as posed in *RQ4*, *RQ5* and *RQ6*. It shows the potential for performance improvements from group profiling compared to individual profiling. Better individual profiling results could possibly be obtained by optimizing feature selection, but was not the focus of this thesis.

The one-class SVM results obtained where surprisingly poor. Similar studies using ocSVM for individual profiling yielded much more favourable results [57]. This could potentially be addressed by optimizing the SVM parameters.

7 Conclusion

This thesis set out to answer questions on group profiling, guided by specific research questions as posed in Chapter 1. The interpretation of both the literature study, as presented in Chapter 3 and experiments, put forth in Chapter 5 are discussed below. This is followed by theoretical and practical implications as can be ascertained from the work performed in this thesis, highlighting both positive aspects and areas where alternative solutions might be better suited.

7.1 Answers to Research Questions

RQ1 What are relevant features for defining normality in a group?

Different levels of profiling granularity were introduced in Section 2.4.2. During this research evaluation of features were confined to indicative of user intent, which can be consider closer to what others have done in masquerade detection as discussed in Section 3.5. Subsequently, we chose features related to users' interactions with a web application which can be likened to the execution of commands on a shared system. Our results in Section 6.3 shows promise through the good performance achieved on our data set.

RQ2 Do features used for individual profiling also work fo group profiling?

The comparison of results between individual and group profiling, presented in Section 6.2 and 6.3 respectively, indicates that the selected features are applicable in both cases.

RQ3 How can such features be collected and measured?

Previous work in Intrustion Detection, and more specifically Masquerade Detection, discussed in Chapter 3, revealed many similarities in collection and measurement methods. In terms of data collection only a small number of publicly available data sets exist. Most related work on masquerade detection have made use of the SEA 3.5.2 data set, which has been noted to have a number of shortcomings, and only allows for a very limited set of features to be measured. Collection methods normally utilised custom developed tools that relied on host gathered audit logs. The mentioned shortcomings and mismatches in data requirements necessitated the generation of a new data set. Features were collected and extracted using Bro as a monitor, we feel having a central point for observations allows for more flexibility. Even though we only considered a simplified simulated environment, our results indicate that this method can be successfully employed.

RQ4 What impact does contextual data have on the characteristics of normal behaviour?

Motivation for the importance of contextual data has been discussed in Section 3.4. We strongly believe that a well defined scope can add meaningful contextual data to aid the profiling of normal behaviour. This thesis was limited to considering a homogeneous group of users as one context, and a shared resource being access by this group to be a second context. This provided supplementary data in terms of class labels, and set the confines within which normality was to be considered. Our developed scenario, discussed in Section 5.3.1 was partially modelled on work performed by [57], detailed in Section 3.5.1. Although simplistic in design, and artificial in the nature of user interactions, we feel it does demonstrate the capabilities that context can add to the profiling task.

RQ5 What is the achievable performance for detecting abnormal behaviour of a homogeneous group of users?

For the purpose of this thesis, any member of a group being misclassified as another group is considered an anomaly. Results are interpreted in light of this assumption. The achieved classification accuracy equates to abnormality detection capability. This thesis only considered the existence of two groups, however a very high level of performance was achieved when using the MultiClass SVM method, results are presented in Table 14.

RQ6 Can group profiling yield better performance than individual profiling within a well defined context?

No direct comparative studies could be found during a literature review. Therefore comparison can only be drawn from the results achieved in the experiment where higher results were achieved from group profiling. However it should be noted that observation imbalances exist between the number of user instances versus the number of group instances observed.

RQ7 What is the achievable difference between one-class and multi-class classification results?

When comparing One-Class SVM results, Table 15 to Multi-Class SVM results, Table 14 a large difference in performance can be seen. In terms of the experiment and limited data set, One-Class SVM can be deemed as infeasible. However, it was noted that One-Class SVM achieved high performance levels in other studies relating to individual profiling as discussed as part of masquerade detection in Section 3.5.

7.2 Theoretical Implications

Based on the experiment data set group profiling can in some instances outperform individual profiling. Although we have to be mindful that the experiment could be an isolated case due to limited data and imbalances in the data set, but does motivate for further research consideration. A potential shortcoming of group profiling that we have raised in Section 4.1 is that not all environments have distinct groups that perform highly homogeneous tasks. However, we argue that given a adequately narrow focus in scope similar tasks can be found. Even if this means having more profiling tasks, relating to different applications or resources.

From related work we deem it fair to gather that individual profiling can be a good method of masquerade detection. Neither group profiling nor individual profiling should be considered in isolation. However, group profiling can act as a compliment to individual profiling by enabling comparison in how similar peers are behaving. Whereas individual profiling could be focused on more finely grained characteristics, unique to each individual. The comparative power that could be leveraged from such a strategy could be especially important to consider in scenarios where individuals maliciously retrain their profile over time. By comparing to peers one can achieve a level of confidence if their behaviour is normal.

Another caution one has to be considerate of is the flattening effect from measuring the average of an entire group of individuals. This could in turn result in removing the small subtleties that could be indicative of malicious actions, allowing for an attacker to go unnoticed. This is of paramount concern during the training phase of such a system.

Based on the work performed in this thesis, it is considered that in order to provide robust group profiling capabilities a system should have a clearly defined, and restricted scope. In order for a group profile to be of value it should be representative of a collection of individuals with common characteristics.

Organizational considerations

Group profiling capabilities should motivate the use of role based access and security design. We speculate that adequate identification of similar roles and grouping of users can aid group profiling capabilities and allow for more robust anomaly detection measures to be viable. As always, it is not a silver bullet solution, but strengthens the security arsenal by giving additional insight based on behaviour in an user group context.

A group profiling system can be imagined containing a component that would first attempt to establish the appropriate grouping of users. A monitoring phase can be used to gather a sufficient amount of data, afterwards data mining techniques can be employed to find users showing similarities in behaviour. The grouped users can then be manually vetted to ascertain whether the grouping is logical; or reliance can be placed on the system if it has been proven to yield accurate results. This is closely related to research on role mining and identity and access management. One can thus realize that these groups do not necessarily have to consist of users in a logical business department. Groups can be formed from other commonalities shared between individuals. Intuitively one can speculate that this could include experience with specific applications and systems, age, role within the organization etc. By performing such an initial phase more robust and reliable results could be achievable. Once groups have been formed and appropriately categorized subsequent phases will perform group profiling based on features identified.

Other Forensic Applications

In terms of digital forensics, similar group profiling techniques can potentially be applied to identify users or groups of users that are of interest during an investigation. For example fraud detection. The classification problem can be utilized to act as a filter,identifying all users behaving similarly to a group or individual. Vetted anomalies can be placed into a database, building a realistic profile of misuse across many organizations.

7.3 Practical Implications

The use of Bro allows for easy modification to the script in order to experiment using different features. This forms part of future work where deeper socio-technical studies can be performed in order to compare feature sets. Bro does not natively support machine learning methods for anomaly detection, but serves as a powerful feature extraction tool. Alternative systems can be used to gather additional contextual information directly from hosts.

During our experiments we have assigned group and user labels according to IP address. An alternative could be to use Active Directory groupings or user groups defined on network systems.

Participation in data collection is a challenge, if real world data can be collected it should be considered. This is especially true for narrowly scoped experiments where high level interactions are extracted instead of aggregated network flow information which could more easily contain abnormal activities resulting in incorrectly modelled behaviour.

7.4 Summary

From the work performed in this thesis it can be seen that masquerade detection is still an evolving field. The thesis highlights many similarities found in the literature in terms of approaches researchers are taking in trying to solve the masquerade detection problem. Common challenges have also been observed. One of the primary challenges faced was finding a suitable data set to test methods on. This issue has been brought up by various authors, and recent work such as WUIL discussed in Section 3.5.2 does shown promise. However, we feel there is a definite need for a more realistic data set. It is also clear that supplementary information, frequently absent from existing data sets are paramount to conceptualising new approaches in research. As we have seen in Section 3.5.2, with the exception of the WUIL data set, most data sets typically only include observation examples. Descriptive details regarding the environment and users are usually brushed over. Typically stated privacy concerns are an important consideration, therefore closely modelled context rich simulations could be an option. It needs to be stated what applications are used, what logical group associations there are, what tasks do users perform etc.

Furthermore, with the lack of commonly available data sets the importance and benefits of a common platform for feature gathering allows for easier information sharing of applied methods and a baseline to compare results with becomes clear. This allows for research discussions to take place around well defined concepts with practical applicability. Also by allowing and encouraging practical implementations in real networks more accurate results can be obtained. Bro is one such system that shows promise. It has its roots in research, and many intrusion detection studies have been performed using this platform. This thesis highlights the benefit that such a system can have for masquerade detection by utilising Bro in the experimental design. We have also made a small contribution in creating a scenario driven data set that explores the addition of contextual information. Hopefully this can inspire others to contribute their efforts in generating more realistic data sets.

The study on group context used in this thesis shows potential. Other work has leveraged concepts of grouping, such as uniqueness of commands shared in a population in [54], however the objective was still to differentiate individuals from each other. There is a void in masquerade detection of group profiling methods with the specific purpose of peer comparison and detecting an outlier in a group. We hope this work motivates for further research into realistic data set generation that not only represents individual users, but also groups of users. This can serve the research community by enabling and supporting much needed research in light of increasing internal threats.

8 Future Work

In order to better evaluate the practical usability of masquerade detection the use of Bro IDS as a monitoring platform should be expanded on. This could include a full implementation of the profiling task directly in Bro. Assessing the viability of using the Bro's *SumStat* framework, coupled with a Kernel Density Estimation based method on access frequency histograms. Exploring threshold based outlier detection methods based on unbalanced populations, could also prove a more realistic representation of what can be encountered in a production system.

The conceptual likeness of profiling to text classification has been discussed in Section 3.5.3, and the good results achieved by applying such methods for masquerade detection. We found that even a very simple method such as Bag-Of-Words feature representation delivered good results in the experiment. As for our feature vector, we would like to consider utilizing Term Frequency–Inverse Document Frequency (TF-IDF) to weigh features for better clustering performance. Other forms of normalization should also be considered [36]. We would also like to extend prelimary analysis of our dataset and future datasets using Spherical K-Means using Cosine Similarity as a distance measure which is better suited for our sparse feature vector [36].

In terms of the data set generation task, more realistic data should be gathered over an extended period of time and include more participants. Depending on privacy concerns, obtaining access to a real world data would be ideal. There is also further work required in devising a threat score to increase the detection window size and reduce false positives.

Further research into abstract features, representing user intent is an important area. An example being building a taxonomy of user interactions based on interactions with a web-based application, creating a set of sequences that relates to different intentions. A feature vector could then be derived indicating the frequencies of these sequences in an observation window. This could allow for better understanding when analysing alerts.

Other use cases for user and group profiling can be considered. Such as identifying groups based on their observed activity. This could further aid forensic purposes such as recognizing known attack groups.

Suggested future research questions:

1. In which way can detection thresholds be dynamically adapted to the tolerance bandwidth of normal and context?

2. Follow the previous question, can normal behaviour, given a tolerance bandwidth, still be recognized in a dynamically changing environment? Can the modelling method used in this thesis be efficiently updated, and what would the implications be against mimicry attacks.

3. Does organizational roles, policy or current logical groupings offer enough similarity in characteristics to allow for group based masquerade detection? A real world data set would be required to fully evaluate the feasibility of this.

4. What is acceptable level of data that can be collected considering privacy? This would require an assessment of the different laws and views on data privacy. Though a bit removed from the technical nature of this thesis this is an important issue as it directly affects whether such a system would have real world applicability.

5. Subsequent to the previous question, is data sets derived from real world systems a feasible option? Should better simulated data sets rather be the pursuit?

6. Can a simplified statistical method, such as KDE on mean bag-of-words vector yield acceptable performance results to make for a realistic real world application?

More focus should be given to masquerade detection using Bro as a feature extraction method. This can inspire extensible research scenarios, and enable better comparison between results.

As discussed in Chapter 6.4, the simulated environment was artificial in nature. As a next step it is very important to test our assumption in a more realistic setting. One of the most pressing problems being correct scoping of anomaly based methods, systems that are aware of contextual information require a lot more research.

Bibliography

[1] Algosec network security report 2013. `http://www.algosec.com/en/resources/network_security_2013`. Last visited May 2014.

[2] Gander, M., Felderer, M., Katt, B., Tolbaru, A., Breu, R., & Moschitti, A. 2013. Anomaly detection in the cloud: Detecting security incidents via machine learning. In *Trustworthy Eternal Systems via Evolving Software, Data and Knowledge*, Moschitti, A. & Plank, B., eds, volume 379 of *Communications in Computer and Information Science*, 103–116. Springer Berlin Heidelberg.

[3] Hildebrandt, M. 2006. Profiling: from data to knowledge. *Datenschutz und Datensicherheit-DuD*, 30(9), 548–552.

[4] Petrović, F. Høgskolen i gjøvik - research project planning - course material 2013.

[5] Leedy, P. D. & Ormrod, J. E. 2005. *Practical research*. Pearson Merrill Prentice Hall Columbus, OH.

[6] Rowlingson, R. 2004. A ten step process for forensic readiness. *International Journal of Digital Evidence*, 2(3), 1–28.

[7] Tan, J. 2001. Forensic readiness. *Cambridge, MA:@ Stake*, 1–23.

[8] Axelsson, S. Intrusion detection systems: A survey and taxonomy. Technical report, Technical report, 2000.

[9] Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., & Vázquez, E. 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1), 18–28.

[10] Shiravi, A., Shiravi, H., Tavallaee, M., & Ghorbani, A. A. 2012. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers Security*, 31(3), 357 – 374.

[11] Axelsson, S. August 2000. The base-rate fallacy and the difficulty of intrusion detection. *ACM Trans. Inf. Syst. Secur.*, 3(3), 186–205.

[12] Godoy, D. & Amandi, A. 2005. User profiling in personal information agents: a survey. *The Knowledge Engineering Review*, 20(04), 329–361.

[13] Senot, C., Kostadinov, D., Bouzid, M., Picault, J., & Aghasaryan, A. 2011. Evaluation of group profiling strategies. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, 2728–2733. AAAI Press.

[14] Mitchell, T. M. 1997. *Machine Learning*. McGraw-Hill, New York.

[15] Duda, R. O., Hart, P. E., & Stork, D. G. 2000. *Pattern Classification (2Nd Edition)*. Wiley-Interscience.

[16] Franke, K. Hig machine learning 2: Lecture 1 slide 25, 2013.

[17] Kononenko, I. & Kukar, M. 2007. *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Horwood Publishing Limited.

[18] Kohavi, R. et al. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, volume 14, 1137–1145.

[19] Olson, D. L. & Delen, D. 2008. *Advanced Data Mining Techniques*. Springer Publishing Company, Incorporated, 1st edition.

[20] Denning, D. E. 1987. An intrusion-detection model. *Software Engineering, IEEE Transactions on*, (2), 222–232.

[21] Lunt, T. F., Tamaru, A., Gilham, F., Jagannathan, R., Jalali, C., Neumann, P. G., Javitz, H. S., Valdes, A., & Garvey, T. D. 1992. *A real-time intrusion-detection expert system (IDES)*. SRI International, Computer Science Laboratory.

[22] Anderson, D., Frivold, T., & Valdes, A. 1995. *Next-generation intrusion detection expert system (NIDES): A summary*. SRI International, Computer Science Laboratory.

[23] Perdisci, R., Ariu, D., Fogla, P., Giacinto, G., & Lee, W. 2009. Mcpad: A multiple classifier system for accurate payload-based anomaly detection. *Computer Networks*, 53(6), 864–881.

[24] Tavallaee, M., Stakhanova, N., & Ghorbani, A. 2010. Toward credible evaluation of anomaly-based intrusion-detection methods. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 40(5), 516–524.

[25] Nguyen, H., Franke, K., & Petrovic, S. 2010. Improving effectiveness of intrusion detection by correlation feature selection. In *Availability, Reliability, and Security, 2010. ARES '10 International Conference on*, 17–24.

[26] Onut, I.-V. & Ghorbani, A. A. 2007. A feature classification scheme for network intrusion detection. *IJ Network Security*, 5(1), 1–15.

[27] Nguyen, H. T., Franke, K., & Petrovic, S. 2010. Towards a generic feature-selection measure for intrusion detection. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, 1529–1532.

[28] Bro website. `http://bro.org/documentation/faq.html#what-is-broq`. Last visited May 2014.

[29] Paxson, V. 1999. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23–24), 2435 – 2463.

[30] Dreger, H., Feldmann, A., Mai, M., Paxson, V., & Sommer, R. Dynamic application-layer protocol analysis for network intrusion detection.

[31] Bro frameworks: Sumstats. `http://www.bro.org/sphinx/frameworks/sumstats.html`. Last visited May 2014.

[32] Adomavicius, G. & Tuzhilin, A. 2005. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6), 734–749.

[33] Adomavicius, G. & Tuzhilin, A. 2011. Context-aware recommender systems. In *Recommender Systems Handbook*, Ricci, F., Rokach, L., Shapira, B., & Kantor, P. B., eds, 217–253. Springer US.

[34] Bolton, R. J. & Hand, D. J. Peer group analysis–local anomaly detection in longitudinal data. Technical report, Citeseer, 2001.

[35] Ferdousi, Z. & Maeda, A. 2006. Anomaly detection using unsupervised profiling method in time series data. In *ADBIS Research Communications*. Citeseer.

[36] Banse, C., Herrmann, D., & Federrath, H. 2012. Tracking users on the internet with behavioral patterns: Evaluation of its practical feasibility. In *Information Security and Privacy Research*, 235–248. Springer.

[37] Ahmed, A. A. E. & Traore, I. 2007. A new biometric technology based on mouse dynamics. *Dependable and Secure Computing, IEEE Transactions on*, 4(3), 165–179.

[38] Bergadano, F., Gunetti, D., & Picardi, C. 2002. User authentication through keystroke dynamics. *ACM Transactions on Information and System Security (TISSEC)*, 5(4), 367–397.

[39] Monrose, F. & Rubin, A. D. 2000. Keystroke dynamics as a biometric for authentication. *Future Generation computer systems*, 16(4), 351–359.

[40] Yu, E. & Cho, S. 2004. Keystroke dynamics identity verification—its problems and practical solutions. *Computers & Security*, 23(5), 428–440.

[41] Chandola, V., Banerjee, A., & Kumar, V. July 2009. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), 15:1–15:58.

[42] Liu, A., Martin, C., Hetherington, T., & Matzner, S. 2005. A comparison of system call feature representations for insider threat detection. In *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC*, 340–347. IEEE.

[43] Kruegel, C., Mutz, D., Valeur, F., & Vigna, G. 2003. On the detection of anomalous system call arguments. In *Computer Security – ESORICS 2003*, Snekkenes, E. & Gollmann, D., eds, volume 2808 of *Lecture Notes in Computer Science*, 326–343. Springer Berlin Heidelberg.

[44] Maggi, F., Matteucci, M., & Zanero, S. 2010. Detecting intrusions through system call sequence and argument analysis. *Dependable and Secure Computing, IEEE Transactions on*, 7(4), 381–395.

[45] Abramson, M. & Gore, S. 2013. Associative patterns of web browsing behavior. In *2013 AAAI Fall Symposium Series*.

[46] Cho, S. & Cha, S. 2004. Sad: web session anomaly detection based on parameter estimation. *Computers and Security*, 23(4), 312 – 319.

[47] Gill, P., Arlitt, M., Carlsson, N., Mahanti, A., & Williamson, C. 2011. Characterizing organizational use of web-based services: Methodology, challenges, observations, and insights. *ACM Transactions on the Web (TWEB)*, 5(4), 19.

[48] Dreger, H., Kreibich, C., Paxson, V., & Sommer, R. 2005. Enhancing the accuracy of network-based intrusion detection with host-based context. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, 206–221. Springer.

[49] Rutkowska, J. 2005. Thoughts about cross-view based rootkit detection. *Unveröffentlichtes Memorandum*.

[50] Amann, B., Sommer, R., Sharma, A., & Hall, S. 2012. A lone wolf no more: supporting network intrusion detection with real-time intelligence. In *Research in Attacks, Intrusions, and Defenses*, 314–333. Springer.

[51] Salem, B. & Karim, T. 2008. Context-based profiling for anomaly intrusion detection with diagnosis. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, 618–623. IEEE.

[52] Chinchani, R., Upadhyaya, S., & Kwiaty, K. 2002. Towards the scalable implementation of a user level anomaly detection system. In *MILCOM 2002. Proceedings*, volume 2, 1503–1508. IEEE.

[53] Sommer, R. & Paxson, V. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on*, 305–316. IEEE.

[54] Schonlau, M. & Theus, M. 2000. Detecting masquerades in intrusion detection based on unpopular commands. *Information Processing Letters*, 76(1), 33–38.

[55] Wang, K. & Stolfo, S. 2003. One-class training for masquerade detection.

[56] Maxion, R. A. & Townsend, T. N. 2002. Masquerade detection using truncated command lines. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, 219–228. IEEE.

[57] Salem, M. B. & Stolfo, S. J. 2011. Modeling user search behavior for masquerade detection. In *Recent Advances in Intrusion Detection*, 181–200. Springer.

[58] Iglesias, J. A., Angelov, P., Ledezma, A., & Sanchis, A. 2012. Creating evolving user behavior profiles automatically. *Knowledge and Data Engineering, IEEE Transactions on*, 24(5), 854–867.

[59] Iglesias, J. A., Ledezma, A., & Sanchis, A. 2009. Creating user profiles from a command-line interface: A statistical approach. In *User Modeling, Adaptation, and Personalization*, 90–101. Springer.

[60] Eldardiry, H., Bart, E., Liu, J., Hanley, J., Price, B., & Brdiczka, O. 2013. Multi-domain information fusion for insider threat detection. In *Security and Privacy Workshops (SPW), 2013 IEEE*, 45–51. IEEE.

[61] Camiña, J. B., Hernández-Gracidas, C., Monroy, R., & Trejo, L. 2014. The windows-users and -intruder simulations logs dataset (wuil): An experimental framework for masquerade detection mechanisms. *Expert Systems with Applications*, 41(3), 919 – 930. Methods and Applications of Artificial and Computational Intelligence.

[62] Song, Y., Salem, M. B., Hershkop, S., & Stolfo, S. J. 2013. System level user behavior biometrics using fisher features and gaussian mixture models. In *Security and Privacy Workshops (SPW), 2013 IEEE*, 52–59. IEEE.

[63] Gold, K., Priest, B., & Carter, K. M. 2013. An expectation maximization approach to detecting compromised remote access accounts. In *The Twenty-Sixth International FLAIRS Conference*.

[64] Corney, M., Mohay, G., & Clark, A. 2011. Detection of anomalies from user profiles generated from system logs. In *Proceedings of the Ninth Australasian Information Security Conference-Volume 116*, 23–32. Australian Computer Society, Inc.

[65] Salem, M. B. & Stolfo, S. J. 2010. Detecting masqueraders: A comparison of one-class bag-of-words user behavior modeling techniques. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 1(1), 3–13.

[66] Vail, D. L., Veloso, M. M., & Lafferty, J. D. 2007. Conditional random fields for activity recognition. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 235. ACM.

[67] Lane, T. & Brodley, C. E. 2003. An empirical study of two approaches to sequence learning for anomaly detection. *Machine learning*, 51(1), 73–107.

[68] Bertino, E., Terzi, E., Kamra, A., & Vakali, A. Dec 2005. Intrusion detection in rbac-administered databases. In *Computer Security Applications Conference, 21st Annual*, 10 pp.–182.

[69] Park, J. S. & Giordano, J. 2006. Role-based profile analysis for scalable and accurate insider-anomaly detection. In *Performance, Computing, and Communications Conference, 2006. IPCCC 2006. 25th IEEE International*, 7–pp. IEEE.

[70] Bianchi, A., Shoshitaishvili, Y., Kruegel, C., & Vigna, G. 2012. Blacksheep: Detecting compromised hosts in homogeneous crowds. In *Proceedings of the 2012 ACM conference on Computer and communications security*, 341–352. ACM.

[71] Razo-Zapata, I. S., Mex-Perera, C., & Monroy, R. 2012. Masquerade attacks based on user's profile. *Journal of Systems and Software*, 85(11), 2640 – 2651.

[72] Salem, M. B. & Stolfo, S. 2011. Data collection and analysis for masquerade attack detection: Challenges and lessons learned.

[73] Pannell, G. & Ashman, H. 2010. *User modelling for exclusion and anomaly detection: a behavioural intrusion detection system*. Springer.

[74] Adamic, L. A. & Huberman, B. A. Zipf's law and the internet. *Glottometrics*, 3(1), 143–150.

[75] Salton, G. & McGill, M. 1983. *Introduction to modern information retrieval*. McGraw-Hill computer science series. McGraw-Hill.

[76] Cortes, C. & Vapnik, V. 1995. Support vector machine. *Machine learning*, 20(3), 273–297.

[77] Duan, K.-B. & Keerthi, S. S. 2005. Which is the best multiclass svm method? an empirical study. In *Multiple Classifier Systems*, 278–285. Springer.

[78] Fletcher, T. 2009. Support vector machines explained.

[79] Sukalpa Chanda, K. F. Hig machine learning 2: Support vector machine 2013.

[80] Wikipedia: Svm example diagram. Last visited May 2014.

[81] Press, W. H. 2007. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.

[82] Thornton, C. University of sussex - svm. `http://www.sussex.ac.uk/Users/christ/crs/ml/lec08a.html`. Last visited May 2014.

[83] Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., & Williamson, R. C. 2001. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7), 1443–1471.

[84] Wagner, D. & Soto, P. 2002. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 255–264. ACM.

[85] Kang, D.-K., Fuller, D., & Honavar, V. 2005. Learning classifiers for misuse and anomaly detection using a bag of system calls representation. In *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC*, 118–125. IEEE.

[86] Sommer, R. & Paxson, V. 2005. Exploiting independent state for network intrusion detection. In *Computer Security Applications Conference, 21st Annual*, 13–pp. IEEE.

[87] Garcia, L. M. 2008. Programming with libpcap±sniffing the network from our own application. *Hakin9-Computer Security Magazine*, 2–2008.

[88] Li, Z., Sanghi, M., Chen, Y., Kao, M.-Y., & Chavez, B. 2006. Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience. In *Security and Privacy, 2006 IEEE Symposium on*, 15–pp. IEEE.

[89] Wolpert, D. H. & Macready, W. G. 1997. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1), 67–82.

[90] Chhabra, P., Scott, C., Kolaczyk, E. D., & Crovella, M. 2008. Distributed spatial anomaly detection. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*. IEEE.

[91] Montgomery, D., Peck, E., & Vining, G. 2007. *Introduction to Linear Regression Analysis, Student Solutions Manual*. A Wiley-Interscience publication. Wiley.

[92] Bro documentation. `http://www.bro.org/sphinx/scripts/proto-analyzers.html?highlight=http_message_done#id-http_message_done`. Last visited May 2014.

[93] Wikipedia http query string. `http://en.wikipedia.org/wiki/Query_string`. Last visited May 2014.

[94] Chang, C.-C. & Lin, C.-J. 2011. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3), 27.

A Source Code

A.1 Bro Script

```
##
# MSc Information Security 2014
# Pieter B. Ruthven
# Group Profiling – Bro Script for feature extraction
# v1.1
##

@load base/frameworks/sumstats

module Test;

export
  {
  # New log
  redef enum Log::ID += { LOG };
  redef enum Log::ID += { LOG2 };
  redef enum Log::ID += { LOG3 }; # NGRAM INTERVAL
  redef enum Log::ID += { LOG4 }; #
  redef enum Log::ID += { LOG5 }; #
  redef enum Log::ID += { LOG6 }; # Training period feature vector ngram frequency
  redef enum Log::ID += { GROUPAGR };
  redef enum Log::ID += { URI_INSTANCE };
  redef enum Log::ID += { URI_INSTANCE_BLOCK }; # BOOL PARAMETER uri_index_enabled :
    COMPATIBLE WITH WEKA MULTI INSTANCE – KEEP INDEX OF URI AND ASSIGN NUMERICA VALUE
    TO EACH.

  # Log file columns
  type Info: record
  {
    who:        string &log;
    resource:    string &log;
    qty:        string &log;
    description:   string &log;
    timestamp:     time &log;
    humantime:     string &log;
    # add entries of addr type for origin etc
  };

  # Log file columns – TEST
  type InfoTest: record
  {
    who:        string &log;
    description:   string &log;
    qty:        count &log;
    timestamp:     time &log;
    # add entries of addr type for origin etc
  };

  # URI_INSTANCE
  type URI_Info: record
  {
    who       : string &log; #host
    class     : string &log; #group
    uri       : string &log;
    timestamp : time &log;
  };

  }

# Adjust output of log file, include seperator etc. This has to be redef outside of an
    event, as they are constants that cannot be modified outside of runtime
redef LogAscii::include_meta = F;
redef LogAscii::separator = ",";
redef LogAscii::set_separator = ";";

# For SMB testing - include all SMB ports
const ports = {
  445/tcp, 139/udp, 137/tcp, 137/udp, 138/udp
};

#----
# PARAMETERS BEGIN - include &redef
const time_interval: interval = 5min; # Metric/Window period
const monitored_resource: addr = 192.168.1.4; # Resource to restrict feature extraction
     on for experiment
const n_of_grams: count = 2; # number of ngrams - CURRENTLY NOT USED everywhere - will
    break script if changed
const bag_size: count = 10; # size of bag of words

# Group of IP's to monitor
const monitored_group = {
  192.168.1.201,
  192.168.1.202,
  192.168.1.203,
  192.168.1.204,
  #192.168.1.100,
  192.168.1.205
};

# More than one group can be defined
const monitored_group2 = {
  192.168.1.101,
  192.168.1.102,
  192.168.1.103,
  192.168.1.104
};
# html, php etc for filtering connections - if resource is in this type then count
    connection
# not implemented
const resource_type = {
  "html", "php", "htm"
};
const uri_filter: bool = T; # apply parameter filter to uri requests. disregard uri
    request parameters.
const uri_index_enabled: bool = T; # enable numerical indexing - classifier
    compatibility and privacy conserving
# PARAMETERS END
# ----

# Method 3 feature vector: frequencies of n unique (host + ngrams as key) access
    sequence per host over entire training period
# Transpose data afterwards
# ngram recorded as feature - missing values in other hosts replaced with 0 value
```

```
global sequence_vector: table[string] of count;
global this_gram: string; # storage for each individual sequence gram

global last_seen: time;

# temp storage for last seen time per host
global unique_resources: table[string] of time;

# temp storage for average time since last visit for each host
global unique_resources_avg: table[string] of double;

# store last 3 resource access sequence
global ngram_trans: vector of string; # concat 3 hosts to single string
global ngram_hosts: table[string] of string; # concat 3 hosts to single string
global ngram_temp: table[string] of string; #store transition between sequences
global ngram_block: table[addr] of string;
global ngram_block_count: table[addr] of count;

global x: count = 0;  # Initialize counter/index for ngram
#global y: count = 0;
global y: table[string] of count;

# URI NUMERICAL INDEX
global uri_index: table[string] of count; # keep track of assigned uri numbers, higher
    indexed is most recently seen
global uri_n: count = 0; # assign unique number to each unique uri

#test, keep track of the interval window
global interval_count: count = 0;

## FUNCTIONS ##

#output features onces interval has passed and retrievalbe from different reducers
function write_sequence_vector()
  {
  for (i in sequence_vector)
    {
    local field: table[count] of string;
    field = split(i, /(\ )/);

    local hostfilter3: Log::Filter = [$name="hostsplit3", $path=fmt("sequence_vector %s
    ", field[1]), $include=set("description", "qty", "timestamp")]; # this keeps the
    original LOG3 file, and creates new seperate file
    Log::add_filter(Test::LOG6, hostfilter3);

    # make compatible with readjustable ngram length
    #local fullsequence: string = "";
    #for (subsequence in field)
    # {
        # reconstruct sequence into one concatenated string, but skip the first part
    which is the host
        #if (fmt("%s", subsequence) != fmt("%s", field[1])) # convert to string
    literals
        # {
    #     fullsequence = string_cat(fullsequence, subsequence, " ");
        # }
    # }

    #Log::write( Test::LOG6, [$who = field[1] ,$description = fmt("%s", fullsequence),
    $qty = sequence_vector[i], $timestamp = network_time()] );
    Log::write( Test::LOG6, [$who = field[1] ,$description = field[2] + " " + field[3],
     $qty = sequence_vector[i], $timestamp = network_time()] );
    #Log::write( Test::LOG6, [$who = i ,$description = "", $qty = sequence_vector[i],
    $timestamp = network_time()] );
    }
  }

## FUNCTIONS END ##

event bro_init()
  {

  # SMB testing - enabled the SMB analyzer on specified ports
  Analyzer::enable_analyzer(Analyzer::ANALYZER_SMB); # not needed
  Analyzer::register_for_ports(Analyzer::ANALYZER_SMB, ports);
  #Analyzer::register_for_ports(Analyzer::ANALYZER_CONTENTS_SMB, ports); - ERROR,
    dynamically allocated?

  # Create log files
  Log::create_stream(LOG, [$columns=Info]);
  Log::create_stream(LOG2, [$columns=Info]); # Testing adding another stream
  Log::create_stream(LOG3, [$columns=Info]); # Testing adding another stream
  Log::create_stream(LOG4, [$columns=Info]); # Testing adding another stream
  Log::create_stream(LOG5, [$columns=Info]); # Testing adding another stream
  Log::create_stream(LOG6, [$columns=InfoTest]); # Testing adding another stream
  Log::create_stream(GROUPAGR, [$columns=Info]); # Testing adding another stream
  Log::create_stream(URI_INSTANCE, [$columns=URI_Info]); # Testing adding another
    stream
  Log::create_stream(URI_INSTANCE_BLOCK, [$columns=URI_Info]); # Testing adding another
     stream

  # is this the right place for the local filter def?
  #trying to split log file, log ngrams seperately
  #only log qty and resource fields in new log file
  local filter: Log::Filter = [$name="ngrams", $path="ngrams", $include=set("qty", "
    resource")]; # this keeps the original LOG3 file, and creates new seperate file
  Log::add_filter(Test::LOG4, filter);

  # Add SumStats. Key pair internal host + http uri. Consider interval inbetween access
     too.

  # Basis of structure from https://github.com/anthonykasza/scratch_pad/blob/master/
    txbaseline/txbaseline.bro
  # Reducer - summing function
  local r1 = SumStats::Reducer($stream="window", $apply=set(SumStats::SUM));
  local r2 = SumStats::Reducer($stream="window.average", $apply=set(SumStats::AVERAGE))
    ;
  # Reducer - Average time interval between connections of known host
  local r3 = SumStats::Reducer($stream="interval.average", $apply=set(SumStats::AVERAGE
    ));
  # Sumstat - collector of data ; epoch result is performed on final result

  local r4 = SumStats::Reducer($stream="ngram.unique", $apply=set(SumStats::SUM));
  local r5 = SumStats::Reducer($stream="ngram.unique.test", $apply=set(SumStats::SUM));

  # test recording only the number of uniqe ngrams - can actually be done with above
    reducers, by only logging count per interval
  local r6 = SumStats::Reducer($stream="ngram.fx", $apply=set(SumStats::UNIQUE));


  # FEATURE EXTRACTION
  # Individual instances such as unique resources accessed
  local r7 = SumStats::Reducer($stream="features.instances", $apply=set(SumStats::
    UNIQUE, SumStats::SUM, SumStats::AVERAGE));
```

```
# Seperate log – unique sequences of access. Labels of the unique n number of
  sequences per host with frequency count
local r8 = SumStats::Reducer($stream="features.sequences", $apply=set(SumStats::
  UNIQUE, SumStats::SUM));
# GROUP
local r9 = SumStats::Reducer($stream="groupfeatures.sequences", $apply=set(SumStats::
  UNIQUE, SumStats::SUM));

SumStats::create([$name = "Average time between connections",
          $epoch = 1min, #in–line with summer
          $reducers = set(r3),
          $epoch_result(ts: time, key: SumStats::Key, result: SumStats::Result) =
          {
            #Log::write( Test::LOG, [$resource = fmt("%s", key$str), $qty = fmt("%f",
   result["interval.average"]$average),$description = "Time interval", $timestamp =
  ts, $humantime = fmt("%T", network_time())] );
            #
            unique_resources_avg[key$str] = result["interval.average"]$average; #
  keyed table entry for average interval access time
          }
    ]);

SumStats::create([$name = "Counting HTTP connections in a window",
          $epoch = 1min,
          $reducers = set(r1),
          $epoch_result(ts: time, key: SumStats::Key, result: SumStats::Result) =
          {
            #Write to log, %float value formated to string
            #Log::write( Test::LOG, [$resource = $description = fmt("%s: %f HTTP
  connections in 1min", key$host, result["Successful HTTP connections"]$sum)] );

            # Testing use of result["window"]$end (13.03.2014)
            # Log::write( Test::LOG, [$resource = fmt("%s", key$str), $qty = fmt("%f
  ", result["window"]$sum), $description = fmt("HTTP connections in 1min last
  connection %T", result["window"]$end), $timestamp = network_time(), $humantime =
  fmt("%T", network_time())] );

            Log::write( Test::LOG, [$resource = fmt("%s", key$str), $qty = fmt("%f",
  result["window"]$sum), $description = fmt("HTTP connections in 1min avg interval:
 %f", unique_resources_avg[key$str]), $timestamp = network_time(), $humantime = fmt
  ("%T", network_time())] );

            #Log::write( Test::LOG, [$description = fmt("What is in result: %s",
  result["window"] )] );
            #SumStats::observe("window.average", SumStats::Key($str=key$str),
  SumStats::Observation($dbl=result["window"]$sum) );
            SumStats::observe("window.average", SumStats::Key($str=key$str),
  SumStats::Observation($num=result["window"]$num) );

          },

          # After result
          $epoch_finished(ts: time) =
          {
          # Reset stats for next window period
          clear_table(unique_resources);
          clear_table(unique_resources_avg);
          }

      ]);

# Average of connections over all window periods
```

```
SumStats::create([$name = "Average HTTP connections over all windows",
            $epoch = 6min,
            $reducers = set(r2),
            $epoch_result(ts: time, key: SumStats::Key, result: SumStats::Result) =
            {
              #Write to log, %float value formated to string
              #Log::write( Test::LOG, [$description = fmt("%s: %f HTTP connections in 1
  min", key$host, result["Successful HTTP connections"]$sum)] );
              Log::write( Test::LOG2, [$resource = fmt("%s", key$str), $qty = fmt("%f",
   result["window.average"]$average),$description = "HTTP connections 5min average
  of 1 min windows", $timestamp = ts, $humantime = fmt("%T", network_time())] );
              #Log::write( Test::LOG, [$description = fmt("What is in AVERAGE result: %
  s", result["window.average"] )] );
              #Log::write( Test::LOG, [$resource = "", $qty = "", $description = fmt("%
  s",unique_resources), $timestamp = ts, $humantime = fmt("%T", network_time())] );
            }
     ]);

# To reduce log size - look at ngram in period of time - same window size as other
  reducers. See distribution over month, in x window sizes.
# number of unique sequences observed in period
SumStats::create([$name = "Uniqe ngram pairs",
            $epoch = 10min,
            $reducers = set(r4),
            $epoch_result(ts: time, key: SumStats::Key, result: SumStats::Result) =
            {
              local n: table[count] of string;
              n = split(key$str, /(\ )/);
              # BEFORE INTERVAL ACCOUNT Log::write( Test::LOG3, [$resource = fmt("%s,%s
  ,%s", n[1], n[2], n[3]), $qty = fmt("%f", result["ngram.unique"]$sum),$description
   = "NGRAMS", $timestamp = ts, $humantime = fmt("%T", network_time())] );
              Log::write( Test::LOG4, [$resource = fmt("%s,%s,%s", n[1], n[2], n[3]),
  $qty = fmt("%f", result["ngram.unique"]$sum),$description = "NGRAMS", $timestamp =
   ts, $humantime = fmt("%s", interval_count)] );

              #unique_resources_avg[key$str] = result["interval.average"]$UNIQUE;
            },
            $epoch_finished(ts: time) =
            {
            # Increase interval for next round
            interval_count = interval_count + 1;
            }
    ]);

SumStats::create([$name = "Uniqe ngram pairs - test",
            $epoch = 60min,
            $reducers = set(r5),
            $epoch_result(ts: time, key: SumStats::Key, result: SumStats::Result) =
            {
              local n2: table[count] of string;
              n2 = split(key$str, /(\ )/);
              # BEFORE INTERVAL ACCOUNT Log::write( Test::LOG3, [$resource = fmt("%s,%s
  ,%s", n[1], n[2], n[3]), $qty = fmt("%f", result["ngram.unique"]$sum),$description
   = "NGRAMS", $timestamp = ts, $humantime = fmt("%T", network_time())] );
              Log::write( Test::LOG4, [$resource = fmt("%s,%s,%s", n2[1], n2[2], n2[3])
  , $qty = fmt("%f", result["ngram.unique.test"]$sum),$description = "NGRAMS TEST
  PERIOD", $timestamp = ts, $humantime = "1"] );

              #unique_resources_avg[key$str] = result["interval.average"]$UNIQUE;
            }
    ]);
```

```
SumStats::create([$name = "Uniqe ngram pairs fx",
          $epoch = 10min,
          $reducers = set(r6),
          $epoch_result(ts: time, key: SumStats::Key, result: SumStats::Result) =
          {
            # BEFORE INTERVAL ACCOUNT Log::write( Test::LOG3, [$resource = fmt("%s,%s
  ,%s", n[1], n[2], n[3]), $qty = fmt("%f", result["ngram.unique"]$sum),$description
   = "NGRAMS", $timestamp = ts, $humantime = fmt("%T", network_time())] );
            Log::write( Test::LOG4, [$resource = fmt("%s", interval_count) , $qty =
  fmt("%d", result["ngram.fx"]$unique),$description = "UNIQUE", $timestamp = ts,
  $humantime = "1"] );

            #unique_resources_avg[key$str] = result["interval.average"]$UNIQUE;
          }
    ]);

SumStats::create([$name = "FEATURE EXTRACTION – basic frequency",
# Highest level stats
# host | # unique resources | # total accesses
        $epoch = time_interval, #SET THE TIME INTERVAL ABOVE
        $reducers = set(r7),
        $epoch_result(ts: time, key: SumStats::Key, result: SumStats::Result) =
        {
          # BEFORE INTERVAL ACCOUNT Log::write( Test::LOG3, [$resource = fmt("%s,%s,%
  s", n[1], n[2], n[3]), $qty = fmt("%f", result["ngram.unique"]$sum),$description =
   "NGRAMS", $timestamp = ts, $humantime = fmt("%T", network_time())] );

          local hostfilter: Log::Filter = [$name="hostsplit", $path=fmt("frequency %s
  ", key$host), $include=set("who", "resource", "qty")]; # this keeps the original
  LOG3 file, and creates new seperate file
          Log::add_filter(Test::LOG3, hostfilter);

          #Log::write( Test::LOG3, [$who = fmt("%s", key$host), $resource = fmt("%d",
   result["features.instances"]$unique), $qty = fmt("%f", result["features.instances
  "]$sum),$description = fmt("%s", result["features.instances"]$unique_vals),
  $timestamp = ts, $humantime = fmt("%f", result["features.instances"]$average)] );
          Log::write( Test::LOG3, [$who = fmt("%s", key$host), $resource = fmt("%d",
  result["features.instances"]$unique), $qty = fmt("%f", result["features.instances"
  ]$sum),$description = "", $timestamp = ts, $humantime = "" ]);

          # Update stats table for host
          # Add function call to write to log file
          # Clear table
          # need seperate log with all resources accessed to compare

          #Log::write( Test::LOG3, [$who = fmt("%s", key$host), $resource = "features
  .instances", $qty = "features.instances",$description = "Unique and Sum",
  $timestamp = ts, $humantime = "1"] );
          #unique_resources_avg[key$str] = result["interval.average"]$UNIQUE;
        }
    ]);

SumStats::create([$name = "FEATURE EXTRACTION – sequence patterns (ngrams)",
  #NGRAM SEQUENCE features
  # host | resource 1 | resource 2 | resource 3 | frequency for this sequence
  # AVERAGE AND COMPARE TO GROUP NORMAL VECTOR COSIN DISTANCE
        $epoch = time_interval, #SET THE TIME INTERVAL ABOVE
        $reducers = set(r8),
        $epoch_result(ts: time, key: SumStats::Key, result: SumStats::Result) =
        {
          # BEFORE INTERVAL ACCOUNT Log::write( Test::LOG3, [$resource = fmt("%s,%s,%
  s", n[1], n[2], n[3]), $qty = fmt("%f", result["ngram.unique"]$sum),$description =
    "NGRAMS", $timestamp = ts, $humantime = fmt("%T", network_time())] );

          #local hostfilter2: Log::Filter = [$name="hostsplit-ngram", $path=fmt("
  ngram %s", key$host), $include=set("who", "qty", "resource")]; # this keeps the
  original LOG3 file, and creates new seperate file
          #Log::add_filter(Test::LOG5, hostfilter2);

          # Log::write( Test::LOG5, [$who = fmt("%s", key$str), $resource = fmt("%d",
   result["features.sequences"]$unique), $qty = fmt("%f", result["features.sequences
  "]$sum),$description = fmt("%s", result["features.sequences"]$unique_vals),
  $timestamp = ts, $humantime = "1" ]);

          local field: table[count] of string;
          field = split(key$str, /(\ )/);

          # Split hostnames
          local hostfilter2: Log::Filter = [$name="hostsplit sequence", $path=fmt("
  sequence %s", field[1]), $include=set("who", "resource", "qty", "timestamp", "
  humantime")]; # this keeps the original LOG5 file, and creates new seperate file
          Log::add_filter(Test::LOG5, hostfilter2);

          # changed - key$host to key$str. added the use of field variable
          # Log::write( Test::LOG5, [$who = fmt("%s", key$str), $resource = fmt("%d",
   result["features.sequences"]$unique), $qty = fmt("%f", result["features.sequences
  "]$sum),$description = "sequence", $timestamp = ts, $humantime = "1" ]);
          Log::write( Test::LOG5, [$who = field[1], $resource = field[2], $qty =
  field[3] ,$description = "", $timestamp = ts, $humantime = fmt("%f", result["
  features.sequences"]$sum) ]);

          # Update stats table for host
          # Add function call to write to log file
          # Clear table
          # need seperate log with all resources accessed to compare

          #Log::write( Test::LOG3, [$who = fmt("%s", key$host), $resource = "features
  .instances", $qty = "features.instances",$description = "Unique and Sum",
  $timestamp = ts, $humantime = "1"] );
          #unique_resources_avg[key$str] = result["interval.average"]$UNIQUE;
        }
    ]);

SumStats::create([$name = "FEATURE EXTRACTION - GROUP sequence patterns (ngrams)",
  #NGRAM SEQUENCE features for entire GROUP
  # GROUP | resource 1 | resource 2 | resource 3 | frequency for this sequence
  # AVERAGE TO CREATE NORMAL VECTOR
        $epoch = time_interval, #SET THE TIME INTERVAL ABOVE
        $reducers = set(r9),
        $epoch_result(ts: time, key: SumStats::Key, result: SumStats::Result) =
        {
          # BEFORE INTERVAL ACCOUNT Log::write( Test::LOG3, [$resource = fmt("%s,%s,%
  s", n[1], n[2], n[3]), $qty = fmt("%f", result["ngram.unique"]$sum),$description =
   "NGRAMS", $timestamp = ts, $humantime = fmt("%T", network_time())] );

          #local hostfilter2: Log::Filter = [$name="hostsplit-ngram", $path=fmt("
  ngram %s", key$host), $include=set("who", "qty", "resource")]; # this keeps the
  original LOG3 file, and creates new seperate file
          #Log::add_filter(Test::LOG5, hostfilter2);

          # Log::write( Test::LOG5, [$who = fmt("%s", key$str), $resource = fmt("%d",
   result["features.sequences"]$unique), $qty = fmt("%f", result["features.sequences
  "]$sum),$description = fmt("%s", result["features.sequences"]$unique_vals),
  $timestamp = ts, $humantime = "1" ]);
```

```
            local field2: table[count] of string;
            field2 = split(key$str, /(\ )/);

            # changed - key$host to key$str. added the use of field variable
            # Log::write( Test::LOG5, [$who = fmt("%s", key$str), $resource = fmt("%d",
      result["features.sequences"]$unique), $qty = fmt("%f", result["features.sequences
    "]$sum),$description = "sequence", $timestamp = ts, $humantime = "1" ]);
            # start at index 2 - leading " " equates an EMPTY field
            Log::write( Test::GROUPAGR, [$who = "GROUP 1", $resource = field2[2], $qty
    = field2[3] ,$description = "", $timestamp = ts, $humantime = fmt("%f", result["
    groupfeatures.sequences"]$sum) ]);


            # Update stats table for host
            # Add function call to write to log file
            # Clear table
            # need seperate log with all resources accessed to compare

            #Log::write( Test::LOG3, [$who = fmt("%s", key$host), $resource = "features
    .instances", $qty = "features.instances",$description = "Unique and Sum",
    $timestamp = ts, $humantime = "1"] );
            #unique_resources_avg[key$str] = result["interval.average"]$UNIQUE;
          }
      ]);

  #end bro_init
  }

# Log all network connections

#event connection_established(c:connectsion)
# {
# Log::write( Test::LOG, [$description = fmt("%s", c)] );
# }

#event http_request(c: connection, method: string, original_URI: string, unescaped_URI:
      string, version: string)
event http_message_done(c: connection, is_orig: bool, stat: http_message_stat)
    {
  # Two requests are generated, second request includes status code.
  # For now only logging successful entries - status code 200 (temp solution - revisit)
  # Generates a Poisson process

  # DECIDE ON FILTER
  # Some observations are going missing
  #
  if ( c$id$resp_h == monitored_resource ) # Include uri containing only php, html, htm
     - only interested in command resources
    #if ( c$id$resp_h == monitored_resource && c$http$method == "GET" &&
    c$http$status_code == 200 ) # Include uri containing only php, html, htm - only
    interested in command resources
    #if ( c$id$resp_h == monitored_resource && c$http$uri in resource_type ) # Include
    uri containing only php, html, htm - only interested in command resources
        {

          local orig_h_str: string = fmt("%s", c$id$orig_h); # convert orig_h to string
      value for key

      # FILTER URI to remove parameters
          # Limit focus on the underlying command
          # Parameters can be random, and distort the statistics
      local current_uri: string = fmt("%s", c$http$uri); # assign current uri value. if
      uri filter applied, value will be updated below.

          if (uri_filter)
        {
        local split_uri_parameter: table[count] of string;
        split_uri_parameter = split(current_uri, /(\?)/); # split on parameter
    seperator
        current_uri = split_uri_parameter[1]; # disregard parameters after the
    seperator character
        }

      if (uri_index_enabled) # index uri numerical (allow compatibility with other
    classifiers)
        {
        if (current_uri !in uri_index)
          {
          uri_index[current_uri] = uri_n; # assign uri index resource number
          uri_n = uri_n + 1;
          }
        current_uri = fmt("%d", uri_index[current_uri]); # replace string
    representation with numerical representation

        }

          # URI INSTANCE LOGGING BEFORE CONTINUING - METHOD 2: DOCUMENT BLOCK NB or
    ocSVM FEATURESET
          # BOOL PARAMETER - ONE OF TWO LOGS: BAG OF WORDS, MULTI INSTANCE COMPATIBLE
    NUMERICAL INDEXED
          # FOR EXTERNAL PROCESSING IN WEKA
          # URI BLOCK
          # First assign class label
          local group_label: count = 0; # label for group
          if (c$id$orig_h in monitored_group)
        {
        group_label = 1;
        }
      else if (c$id$orig_h in monitored_group2)
        {
        group_label = 2;
        }
          if (group_label > 0) # only IP matching monitored group, test based on
    assigned label
            {

        Log::write( Test::URI_INSTANCE, [$who = orig_h_str, $class = fmt("%d",
    group_label), $uri = " " + current_uri + " ", $timestamp = network_time()] );

        # URI BLOCK OF bag_size (default 10) PER HOST -> OUTPUT
        # INITIALIZE
        if (c$id$orig_h !in ngram_block_count)
          {
          ngram_block_count[c$id$orig_h] = 0;
          ngram_block[c$id$orig_h] = "";
          }

        # ADD TO BLOCK
        ngram_block[c$id$orig_h] = ngram_block[c$id$orig_h] + " " + current_uri + " ";
    # UPDATE BLOCK
        ngram_block_count[c$id$orig_h] = ngram_block_count[c$id$orig_h] + 1;

        if (ngram_block_count[c$id$orig_h] == bag_size) # CHECK SIZE OF BAG
```

```
        {
        Log::write( Test::URI_INSTANCE_BLOCK, [$who = orig_h_str, $class = fmt("%d",
group_label), $uri = ngram_block[c$id$orig_h], $timestamp = network_time()] );
        ngram_block_count[c$id$orig_h] = 0; # RESET COUNTER
        ngram_block[c$id$orig_h] = "";  # RESET block
        }
      # URI BLOCK END

      }

    # INITIALIZE host counter
        if (orig_h_str !in y)
      {
      y[orig_h_str] = 0; # initialize the set counter per host
      }

    # INITIALIZE NEW SEQUENCE
        if (y[orig_h_str] == 0)
      {
      ngram_trans[x] = ""; # init each new vector - BUG? No results if not
initialised, perhaps use escape function prior to assignment.

      ngram_hosts[fmt("%s", c$id$orig_h)] = "";

      if (orig_h_str in ngram_temp && ngram_temp[orig_h_str] != "") #if this host has
 completed a sequence recording before, start new sequence with previous ending
        {
        ngram_hosts[orig_h_str] = ngram_temp[orig_h_str];
        y[orig_h_str] = y[orig_h_str] + 1; # one ngram added, increase index
        #Log::write( Test::LOG, [$who = fmt("%s", y[orig_h_str]), $resource = fmt("%s
",ngram_temp), $qty = "",$description = fmt("---FINDME---"), $timestamp =
network_time(), $humantime = ""] );
        ngram_temp[orig_h_str] = "";# CLEAR TABLE between observations, otherwise
risk of duplicating sequences. Maybe not neccessary.
        }

      }
    # remove after testing complete
        ngram_trans[x] = strip(string_cat(ngram_trans[x]," ", c$http$host));



        # if considering URI - remove paremters after ? in order to have too many
unique values
        ngram_hosts[fmt("%s", c$id$orig_h)] = string_cat(ngram_hosts[fmt("%s",
c$id$orig_h)]," ", current_uri); # strip? Using URI
        y[orig_h_str] = y[orig_h_str] + 1; # counter for each host entry
        if (y[orig_h_str] == n_of_grams) # max nr grams recorded per sequence
      {
      # Add observation before counter reset
      #SumStats::observe("ngram.unique", SumStats::Key($host=c$id$orig_h), SumStats::
Observation($str=ngram_trans[x])); # count number of unique ngram pairs per host
-- TEMP DISABLED

      #SumStats::observe("ngram.unique", SumStats::Key($str=ngram_trans[x]), SumStats
::Observation($num=1)); # count number of unique ngram pairs per host
      #SumStats::observe("ngram.unique.test", SumStats::Key($str=ngram_trans[x]),
SumStats::Observation($num=1)); # count number of unique ngram pairs per host -
entire test period

      #test unique
    #SumStats::observe("ngram.fx", SumStats::Key($str=fmt("%s", interval_count)),
SumStats::Observation($str=ngram_trans[x])); # count number of unique ngram pairs
per interval count

    #new one and only remove the rest
    Log::write( Test::LOG, [$resource = orig_h_str, $qty = "",$description = fmt("%
s", ngram_hosts[orig_h_str]), $timestamp = network_time(), $humantime = fmt("%T",
network_time())] );
    #SumStats::observe("features.sequences", SumStats::Key($host=c$id$orig_h),
SumStats::Observation($str=ngram_hosts[fmt("%s", c$id$orig_h)]));
    # added - ngram + orig_h as the key, this enables individual frequency count
for each host/ngram combination
    SumStats::observe("features.sequences", SumStats::Key($str=fmt("%s",
c$id$orig_h) + ngram_hosts[fmt("%s", c$id$orig_h)]), SumStats::Observation($str=
ngram_hosts[fmt("%s", c$id$orig_h)]));


    # aggregate group data. have to still consider each host, to record distinct
sequences, but count as one no matter which host
    if (c$id$orig_h in monitored_group)
      {
      # only record the sequence pattern, and not the host as key
      SumStats::observe("groupfeatures.sequences", SumStats::Key($str= ngram_hosts[
fmt("%s", c$id$orig_h)]), SumStats::Observation($str=ngram_hosts[fmt("%s",
c$id$orig_h)]));
      }

    # method 3 feature vector of training period
    this_gram = fmt("%s", c$id$orig_h) + ngram_hosts[fmt("%s", c$id$orig_h)];
    if (this_gram !in sequence_vector)
      {
      sequence_vector[this_gram] = 0;
      }
    sequence_vector[this_gram] = sequence_vector[this_gram] + 1;
    #--

    # Submit ngram, then wipe string
    # Reset counters

    # wipe current ngram pattern once it has been observed
    # ngram_hosts[fmt("%s", c$id$orig_h)] = "";

    x = x + 1;
    y[orig_h_str] = 0;

    #memorize last ngram to transition to new sequence
    #per host as not to mix up sequences between concurrent connections
    local split_temp: table[count] of string;
    split_temp = split(ngram_hosts[orig_h_str], /(\ )/);
    #ngram_temp[orig_h_str] = " " + split_temp[2]; # hard coded, 2nd ngram for
starting next sequence. Add leading space for future split from hostname in
reducer
    ngram_temp[orig_h_str] = " " + current_uri; # split gives same value repeatedly
 - http uri accurate?

    }


      #ngram[x][y] = c$http$host;
      #y = y + 1;
      #if (y == 3)
  # {
      # x = x + 1;
      # y = 0;
      # }

          # Log each resources access — disabled temporarily
            #Log::write( Test::LOG, [$description = fmt("%s%s ACCESSED", c$http$host,
    c$http$uri), $timestamp = network_time(), $humantime = fmt("%T", network_time())]
    );
      #Log::write( Test::LOG, [$description = fmt("%s", c$id$orig_h), $timestamp =
    network_time(), $humantime = fmt("%T", network_time()) ] );

      # PERFORMANCE CONCERN using table — is there a way to get the data from the
    observation stream?

      # initialize first seen time
      #Add concat of orig_h
      if ( c$http$host !in unique_resources ) # Rather user Known Host function in Bro?
        {
        # First seen or last seen? how to update per host — include in str part of
    observation result to be able to recall per entry
        #Log::write( Test::LOG, [$resource = c$http$host, $qty = "",$description = "
    First time seen — added time", $timestamp = network_time(), $humantime = fmt("%T",
     network_time())] );
        unique_resources[c$http$host] = network_time();
        }

          SumStats::observe("window", SumStats::Key($str=fmt("%s",c$id$orig_h)),
    SumStats::Observation($str=c$http$host));

          # OVERLORD STRIKES AGAIN — USING URI
          SumStats::observe("features.instances", SumStats::Key($host=c$id$orig_h),
    SumStats::Observation($str=current_uri));


          #DEBUG# Log::write( Test::LOG, [$resource = c$http$host, $qty = "",
    $description = fmt("TIME SINCE LAST SEEN: %f", network_time() — unique_resources[
    c$http$host]), $timestamp = network_time(), $humantime = fmt("%T", network_time())
    ] );
          SumStats::observe("interval.average", SumStats::Key($str=c$http$host),
    SumStats::Observation($dbl=network_time() — unique_resources[c$http$host]));

      # Key — combination of http host and uri
      #SumStats::observe("window", SumStats::Key($str=c$http$host + c$http$uri),
    SumStats::Observation($num=1));
      if ( c$http$host in unique_resources ) # Rather user Known Host function in Bro?
        {
        # First seen or last seen? how to update per host — include in str part of
    observation result to be able to recall per entry
        unique_resources[c$http$host] = network_time(); # update last seen time
        }

      # Observe network time, build if statement to determine last time site was
    accessed
      #SumStats::observe("window", SumStats::Key($str=c$http$host), SumStats::
    Observation($dbl=network_time()));
        }


    }

event bro_done()
  {
```

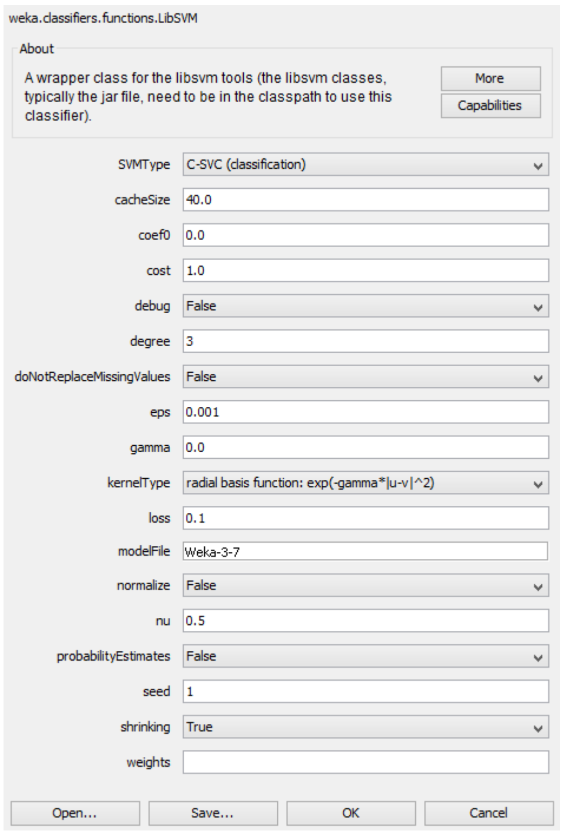
```
# DEBUG
Log::write( Test::LOG, [$resource = "", $qty = "",$description = fmt("——DEBUG INFO
  ——"), $timestamp = network_time(), $humantime = fmt("%T", network_time())] );
#Log::write( Test::LOG, [$resource = "", $qty = "",$description = fmt("AVG %s",
  unique_resources_avg ), $timestamp = network_time(), $humantime = fmt("%T",
  network_time())] );

#Log::write( Test::LOG, [$resource = "", $qty = "",$description = fmt("NGRAM 1: %s",
  ngram_hosts ), $timestamp = network_time(), $humantime = fmt("%T", network_time())
  ] );
Log::write( Test::LOG, [$resource = fmt("Sequence Vector: %s", sequence_vector ),
  $qty = "",$description = "", $timestamp = network_time(), $humantime = fmt("%T",
  network_time())] );
Log::write( Test::LOG, [$resource = fmt("URI INDEX: %s", uri_index ), $qty = "",
  $description = "", $timestamp = network_time(), $humantime = fmt("%T",
  network_time())] );
#Write sequence frequencies of training period to disk
write_sequence_vector();


clear_table(unique_resources);
clear_table(unique_resources_avg);
}
```

A.2 Post Process Script

```
##
# MSc Information Security 2014
# Pieter B. Ruthven
# Group Profiling — Post—Process Bro Log
# v1.1
##

# Bag of word generator
# Build global index
# Frequency vector for each host in a 10 command span
# Includes unique sequence pattern feature

import sys
import csv
from collections import defaultdict

def is_float(s):
# Check if a string can be converted to a float
    try:
        float(s)
        return True
    except ValueError:
        return False

def print_feature(ftr_x):
    for line in ftr_x:
        print line

# Find highest maximum resource index used
def find_max_index(ftr_x):
    max_index = 0
    for y in ftr_x:
        for x in y:
            if int(x) > max_index:
                max_index = int(x)

    return max_index

# Build dictionary for bag of words frequency count
def build_dict(ftr_x):

    # find highest indexed element
    # compare this with Bro script index_count

    ftr_bow = [] # bag of words resource frequency count for each interval
    temp_bow = defaultdict(int)
    max_index = find_max_index(ftr_x)

    for num in range(max_index+1): # NB +1 0 — 111, not 0 — 110 (this was bug)
        temp_bow[num] = 0 # Python 0—index, +1 to align with Bro 1—index
        print num,

    for line in ftr_x:
        for x in line: # each indexed uri resource
            # for each entry, increment relevant frequency count in dictionary
            temp_bow[int(x)] += 1
        # add line frequency counts
        ftr_bow.append(temp_bow.values())
        # clear temp for next round
        # better way to do this im sure?
        temp_bow.clear() # NB
        for num in range(max_index+1): # NB +1 0 — 111, not 0 — 110 (this was bug)
            temp_bow[num] = 0 # Python 0—index, +1 to align with Bro 1—index

    print "BOW entries:", len(ftr_bow)
    print "Last BOW:", ftr_bow[len(ftr_bow)—1]
    #print type(temp_bow.keys()[0])

    return ftr_bow

def read_features():
    f = open('bro.log', 'rU')
    reader = []
    for line in f:
        # List of lists. Each line —1 to exclude \n, split up
        reader.append(line[:—1].split(','))

    # Split sequence commands
    # Todo: Change seperator in Bro Script
    # Will become redundant

    # DEFINE FEATURE VARIABLES
    ftr_user = [] # class by IP
    ftr_group = [] # class by group number
    ftr_sequence = [] # sequence of access based on bag_size in Bro script
    ftr_sequence_concat = [] # unique sequence feature, support seperability
    ftr_time = [] # network time timestamp
    ftr_bow = [] # bow

    for line in reader:
        ftr_user.append(line[0])
        ftr_group.append(line[1])
        ftr_sequence.append(line[2].split()) # split sequence indexes into seperate
    elements
        ftr_sequence_concat.append(line[2].replace(" ", ""))
        ftr_time.append(line[3])

    print "MAX Index:", find_max_index(ftr_sequence)

    ftr_bow = build_dict(ftr_sequence)

    # ZIP to transpose columns of data together from different lists
    # Write data to csv file

    for bow in ftr_bow:
        print len(bow)

    output_file = zip(ftr_user, ftr_group, ftr_sequence_concat, ftr_time, ftr_bow)
    with open('processed.csv', 'wb') as fin:
        writer = csv.writer(fin, delimiter = ',')
        writer.writerows(output_file)

def main():

    read_features()

if __name__ == '__main__':
    main()
```

B Figures

WEKA CSV importer and LibSVM settings are shown in Figure 22 and 23.

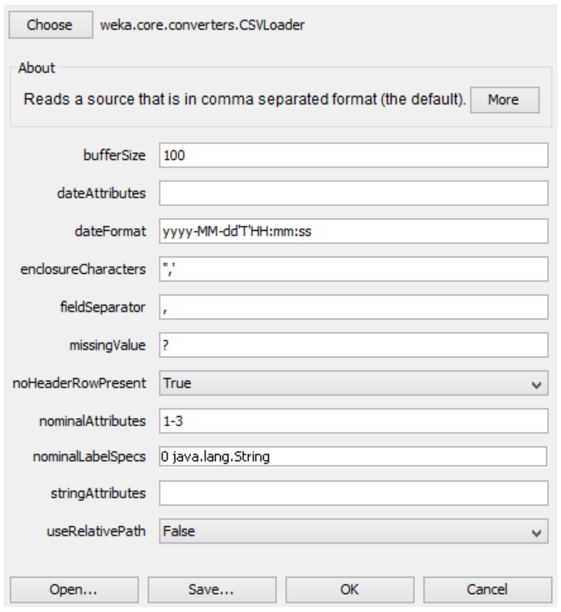

weka.classifiers.functions.LibSVM

About

A wrapper class for the libsvm tools (the libsvm classes, typically the jar file, need to be in the classpath to use this classifier).

SVMType	C-SVC (classification)
cacheSize	40.0
coef0	0.0
cost	1.0
debug	False
degree	3
doNotReplaceMissingValues	False
eps	0.001
gamma	0.0
kernelType	radial basis function: exp(-gamma*|u-v|^2)
loss	0.1
modelFile	Weka-3-7
normalize	False
nu	0.5
probabilityEstimates	False
seed	1
shrinking	True
weights	

Figure 22: WEKA LibSVM Default Settings



weka.core.converters.CSVLoader

About

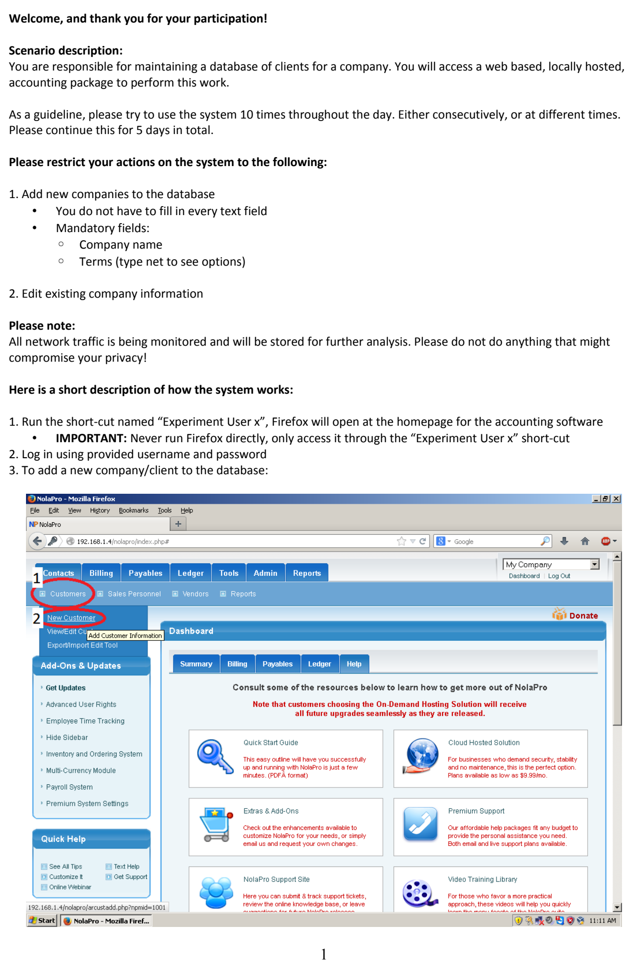
Reads a source that is in comma separated format (the default).

bufferSize	100
dateAttributes	
dateFormat	yyyy-MM-dd'T'HH:mm:ss
enclosureCharacters	",'
fieldSeparator	,
missingValue	?
noHeaderRowPresent	True
nominalAttributes	1-3
nominalLabelSpecs	0 java.lang.String
stringAttributes	
useRelativePath	False

Figure 23: WEKA CSV Converter Settings

C Participant documentation

The instructions given to participants of Group 1, detailing the use of the NolaPro® system, can be seen below.



Welcome, and thank you for your participation!

Scenario description:
You are responsible for maintaining a database of clients for a company. You will access a web based, locally hosted, accounting package to perform this work.

As a guideline, please try to use the system 10 times throughout the day. Either consecutively, or at different times. Please continue this for 5 days in total.

Please restrict your actions on the system to the following:

1. Add new companies to the database
   - You do not have to fill in every text field
   - Mandatory fields:
     - Company name
     - Terms (type net to see options)

2. Edit existing company information

Please note:
All network traffic is being monitored and will be stored for further analysis. Please do not do anything that might compromise your privacy!

Here is a short description of how the system works:

1. Run the short-cut named "Experiment User x", Firefox will open at the homepage for the accounting software
   - **IMPORTANT:** Never run Firefox directly, only access it through the "Experiment User x" short-cut
2. Log in using provided username and password
3. To add a new company/client to the database:

Consult some of the resources below to learn how to get more out of NolaPro

Note that customers choosing the On-Demand Hosting Solution will receive all future upgrades seamlessly as they are released.

Quick Start Guide: This easy outline will have you successfully up and running with NolaPro is just a few minutes. (PDFÂ format)

Cloud Hosted Solution: For businesses who demand security, stability and no maintenance, this is the perfect option. Plans available as low as $9.99/mo.

Extras & Add-Ons: Check out the enhancements available to customize NolaPro for your needs, or simply email us and request your own changes.

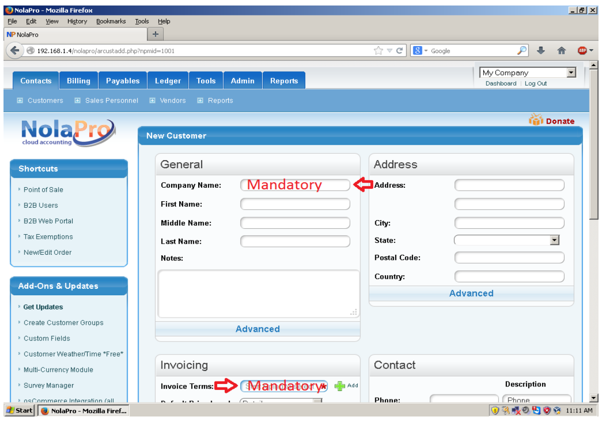
Premium Support: Our affordable help packages fit any budget to provide the personal assistance you need. Both email and live support plans available.

NolaPro Support Site: Here you can submit & track support tickets, review the online knowledge base, or leave

Video Training Library: For those who favor a more practical approach, these videos will help you quickly

1

Figure 24: Participant instructions 1


NolaPro - Mozilla Firefox
File Edit View History Bookmarks Tools Help
NolaPro
192.168.1.4/nolapro/arcustadd.php?npmid=1001
Contacts Billing Payables Ledger Tools Admin Reports
Customers Sales Personnel Vendors Reports
My Company
Dashboard Log Out
Donate
NolaPro cloud accounting
Shortcuts
Point of Sale
B2B Users
B2B Web Portal
Tax Exemptions
New/Edit Order
Add-Ons & Updates
Get Updates
Create Customer Groups
Custom Fields
Customer Weather/Time *Free*
Multi-Currency Module
Survey Manager
Advanced
Add
Start
NolaPro - Mozilla Firef...
11:11 AM

New Customer

General

Company Name: Mandatory

First Name:

Middle Name:

Last Name:

Notes:

Address

Address:

City:

State:

Postal Code:

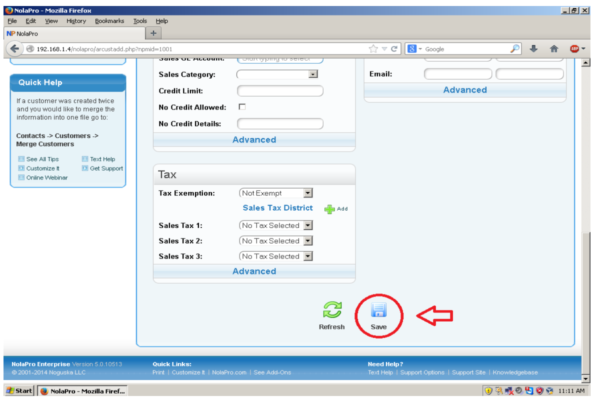
Country:

Invoicing

Invoice Terms: Mandatory*

Contact

Phone: Description Phone


NolaPro - Mozilla Firefox
File Edit View History Bookmarks Tools Help
NolaPro
192.168.1.4/nolapro/arcustadd.php?npmid=1001
Quick Help
See All Tips
Text Help
Customize It
Get Support
Online Webinar
Advanced
Sales Tax District
Add
Refresh
Save
NolaPro Enterprise Version 5.0.10513
© 2001-2014 Noguska LLC
Quick Links:
Print | Customize It | NolaPro.com | See Add-Ons
Need Help?
Text Help | Support Options | Support Site | Knowledgebase
Start
NolaPro - Mozilla Firef...
11:11 AM

If a customer was created twice and you would like to merge the information into one file go to:

Contacts -> Customers -> Merge Customers

Sales Category:

Credit Limit:

No Credit Allowed: ☐

No Credit Details:

Email:

Tax

Tax Exemption: Not Exempt

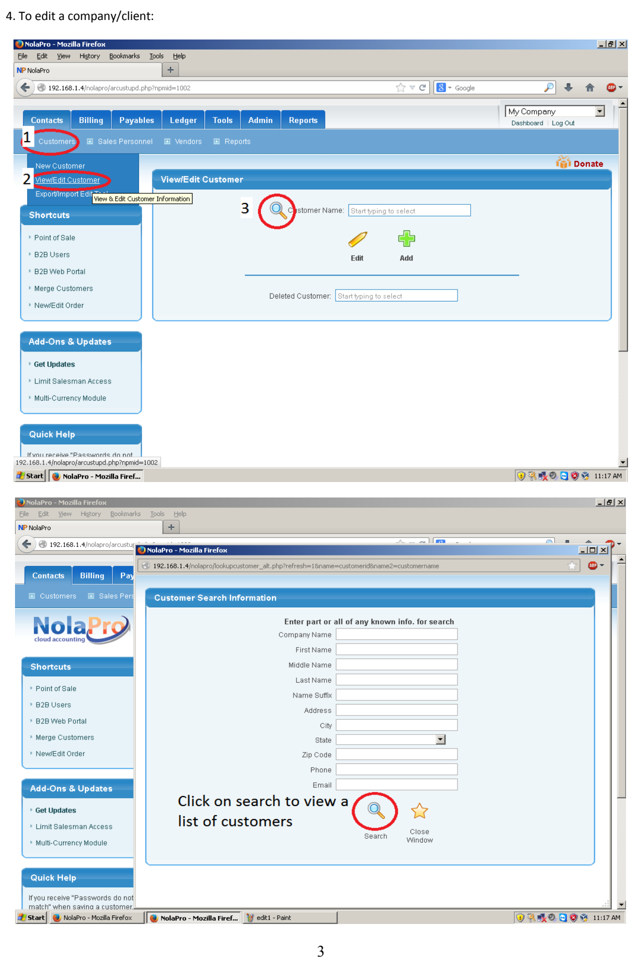
Sales Tax 1: No Tax Selected

Sales Tax 2: No Tax Selected

Sales Tax 3: No Tax Selected

2

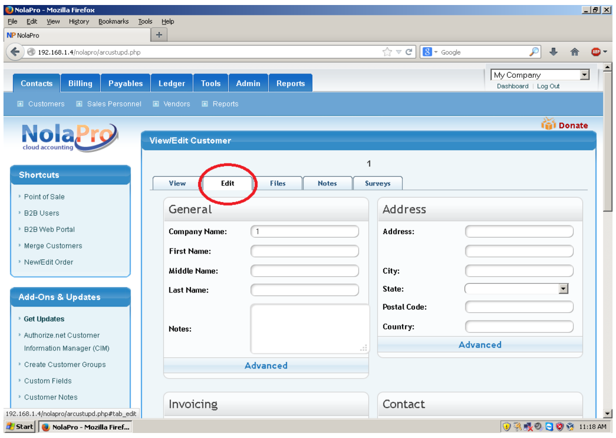
Figure 25: Participant instructions 2


4. To edit a company/client:

Click on search to view a list of customers

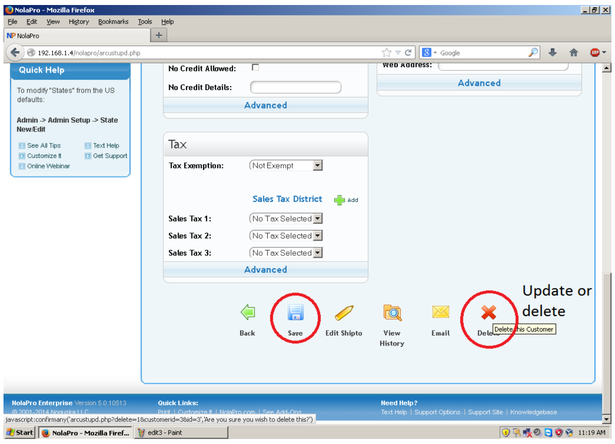
3

Figure 26: Participant instructions 3



Company Name: 1
First Name:
Middle Name:
Last Name:
Notes:
Address:
City:
State:
Postal Code:
Country:

192.168.1.4/nolapro/arcustupd.php#tab_edit


NolaPro - Mozilla Firefox
File Edit View History Bookmarks Tools Help
192.168.1.4/nolapro/arcustupd.php
Quick Help
To modify "States" from the US defaults:
Admin -> Admin Setup -> State New/Edit
See All Tips
Text Help
Customize It
Get Support
Online Webinar
Advanced
Tax
Sales Tax District
Add
Back
Save
Edit Shipto
View History
Email
Delete
Delete this Customer
Update or delete
NolaPro Enterprise Version 5.0.10513
Quick Links:
Need Help?
Text Help | Support Options | Support Site | Knowledgebase
Start
edit3 - Paint
11:19 AM

No Credit Allowed:
No Credit Details:
Web Address:
Tax Exemption: Not Exempt
Sales Tax 1: No Tax Selected
Sales Tax 2: No Tax Selected
Sales Tax 3: No Tax Selected

javascript:confirmany('arcustupd.php?delete=1&customerid=3&id=3','Are you sure you wish to delete this?')

4

Figure 27: Participant instructions 4