A Toolset for Intrusion and Insider Threat Detection

Markus Ring, Sarah Wunderlich, Dominik Grüdl, Dieter Landes, Andreas Hotho

Abstract Company data are a valuable asset and must be protected against unauthorized access and manipulation. In this contribution, we report on our ongoing work that aims to support IT security experts with identifying novel or obfuscated attacks in company networks, irrespective of their origin inside or outside the company network. A new toolset for anomaly based network intrusion detection is proposed. This toolset uses flow-based data which can be easily retrieved by central network components. We study the challenges of analysing flow-based data streams using data mining algorithms and build an appropriate approach step by step. In contrast to previous work, we collect flow-based data for each host over a certain time window, include the knowledge of domain experts and analyse the data from three different views. We argue that incorporating expert knowledge and previous flows allow us to create more meaningful attributes for subsequent analysis methods. This way, we try to detect novel attacks while simultaneously limiting the number of false positives.

Markus Ring
Department of Electrical Engineering and Computer Science, Coburg University of Applied Sciences and Arts, 96450 Coburg, Germany, e-mail: markus.ring@hs-coburg.de

Sarah Wunderlich
Department of Electrical Engineering and Computer Science, Coburg University of Applied Sciences and Arts, 96450 Coburg, Germany, e-mail: sarah.wunderlich@hs-coburg.de

Dominik Grüdl
Department of Electrical Engineering and Computer Science, Coburg University of Applied Sciences and Arts, 96450 Coburg, Germany, e-mail: dominik.gruedl@stud.hs-coburg.de

Dieter Landes
Department of Electrical Engineering and Computer Science, Coburg University of Applied Sciences and Arts, 96450 Coburg, Germany, e-mail: dieter.landes@hs-coburg.de

Andreas Hotho
Data Mining and Information Retrieval Group, University of Würzburg, 97074 Würzburg, Germany e-mail: hotho@informatik.uni-wuerzburg.de

1 Introduction

Information security is a critical issue for many companies. The fast development of network-based computer systems in modern society leads to an increasing number of diverse and complex attacks on company data and services. However, company data are a valuable asset which must be authentic to be valuable and inaccessible to unauthorized parties [27]. Therefore, it is necessary to find ways to protect company networks against criminal activities, called intrusions. To reach that goal, companies use various security systems like firewalls, security information and event management systems (SIEM), host-based intrusion detection systems, or network intrusion detection systems.

This chapter focuses on anomaly-based network intrusion detection systems. Generally, network intrusion detection systems (NIDS) try to identify malicious behaviour on network level and can be categorized into misuse and anomaly detection [25]. Misuse detection utilizes known attacks and tries to match incoming network activities with predefined signatures of attacks and malware [14]. Consequently, only known attacks can be found and the list of signatures must be constantly updated [18]. The increasing trend of insider attacks complicates this challenge even more. It is harder to identify signatures that indicate unusual behaviour as these behaviours may be perfectly normal under slightly different circumstances. Anomaly detection systems on the other hand assume that normal and malicious network activities differ [18]. Regular network activities are modelled by using representative training data, whereas incoming network activities are labelled as malicious if they deviate significantly [14]. Thus, anomaly detection systems are able to detect novel or obfuscated attacks. However, operational environments mainly apply misuse detection systems [52]. Sommer and Paxson [52] identify the following reasons for the failure of anomaly-based intrusion detection systems in real world settings:

1. high cost of false positives
2. lack of publicly available training and evaluation data sets
3. the semantic gap between results and their operational interpretation
4. variability of input data
5. fundamental evaluation difficulties

In this contribution, we propose a novel approach for anomaly based network intrusion detection in which we try to consider the challenges identified by Sommer and Paxson [52]. We report on our ongoing work that aims to develop an interactive toolset which supports IT security experts by identifying malicious network activities. The resulting toolset Coburg Utility Framework (CUF) is based on a flexible architecture and offers a wide range of data mining algorithms. Our work addresses various aspects that may contribute to master the challenge of identifying significant incidents in network data streams, irrespective of their origin from inside or outside of a company network. In particular, our approach builds upon flow-based data. Flows are meta information about network communications between hosts and can be easily retrieved by central network components like routers, switches or firewalls. This results in fewer privacy concerns compared to packet-based approaches and the amount of flow data is considerably smaller in contrast to the complete packet information.

The resulting approach is multistaged: We first propose an enrichment of flow-based data. To this end, we collect all flows within a given time window for each host and calculate additional attributes. Simultaneously, we use additional domain knowledge to add further information to the flow-based data like the origin of the *Source IP Address*. In order to detect malicious network traffic, the enriched flow-based data is analysed from three different perspectives using data mining algorithms. These views are adjusted to detect various phases of attacks like *Scanning* or *Gaining Access*. We are confident that collecting flows for each host separately for a certain time window and the inclusion of domain knowledge allows us to calculate more meaningful attributes for subsequent analysis methods. Further, the three different analysis views allow us to reduce the complexity in each single view. We also describe our process to generate labelled flow-based data sets using *OpenStack* in order to evaluate the proposed approach.

The chapter is organized as follows: The next section introduces the general structure of our toolset Coburg Utility Framework (CUF). Section 3 proposes our data mining approach to analyse flow-based data streams using CUF. Then, the generation of labelled flow-based data sets for training and evaluation is described in Section 4. Section 5 discusses related work on flow-based anomaly detection. The last section summarizes the chapter and provides an outlook.

2 Coburg Utility Framework

We use the following definitions: A data stream $X_S = \{X_1, X_2, ..., X_n\}$ is an unbounded set ($n = \infty$) of data points X_i. Each data point X_i is characterized by a set of attributes. A data set X_D consists of a fixed number of data points X_i. When talking about network data, each flow can be considered a data point.

The objective of our research is a methodology for analysing flow-based data streams to detect malicious network activities. To this end, a combination of various data mining methods (clustering, classification and visualization) needs to be explored. Different configurations are then compared and evaluated with respect to their quality. An appropriate workflow can only be found by experimenting with different processing steps on real world data. By assessing the results we are able to figure out the best possible configuration. Hence, a highly flexible experimental environment is required to allow for an easy setup of different methods. Hereby, a wide range of tools for modelling, visualization and evaluation is used.

The Coburg Utility Framework (CUF) aims to fulfil these requirements. The underlying core architecture of CUF is presented in the following Section. Section 2.2 provides an overview of available algorithms for offline and online analysis. CUF also incorporates offline algorithms which we developed and applied in earlier work [27] since these algorithms can be used for further and deeper investigations of malicious network activities.

2.1 Architecture of CUF

CUF implements a pipes-and-filters architecture which is often used in software applications that handle and process data streams. Filters constitute independent working steps that manipulate incoming data. Pipes connect pairs of filters and pass data on to other filters. The architecture of CUF ensures optimal encapsulation of pipes and filters, and both are realized as services with a common interface. Services are implemented by a service provider and loaded dynamically by a service loader. In addition, this architecture provides an easy way of integrating new filters, e.g., when integrating a new clustering algorithm as a filter, all other filters and pipes stay unchanged.

As an enhancement to the pure pipes-and-filters pattern, CUF allows to split and merge workflows. Thus, different clustering filters or multiple instances of one filter with different parameter settings can be combined to process data in a single run and compare their results directly. Since each of these steps is implemented as an individual filter in CUF, different workflows can easily be set up and executed.

Figure 1 shows a simple data mining workflow in CUF to cluster network data. At

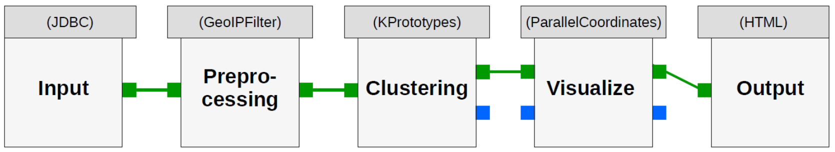


Fig. 1 Representation of a data mining workflow in CUF.

first, an input filter reads the data from a database. Then, a preprocessing filter adds additional information to each data point. In this case, the *GeoIPFilter* adds to each data point the corresponding geographical coordinates of the *Source IP Address* and *Destination IP Address* using an external database. The third filter in the processing chain is the clustering algorithm *k-Prototypes*. This filter sorts the incoming data points in a predefined number of k clusters (groups) according to their similarity. The fourth filter visualizes the results in form of *Parallel Coordinates* and the last filter writes the results to disk. Input and output interfaces of the filters are represented by the coloured rectangles in Figure 1. Input interfaces are on the left side and output interfaces are on the right side. Green rectangles transport data points, whereas blue rectangles transport cluster objects. Figure 1 shows that the *Parallel Coordinates* filter is able to process data points (green input rectangle) or cluster objects (blue input rectangle). The ability of filters to read and generate different output formats allows us to easily split and merge workflows.

One of the major challenges in analysing network data is the large amount of data generated by network devices. Company networks generate millions of network flows per hour. Consequently, an efficient and fast processing chain is required. Here, the pipes-and-filters architecture of CUF itself has a big advantage. Since filters work independently, each of them is executed in its own thread such that multicore architectures may easily be utilized to reduce execution time.

2.2 *Filters in CUF*

This section provides an overview of available filters in CUF. We implement many filters on our own to increase flexibility and customization of algorithms for flow-based data. For example, we implemented the *k-Prototypes* algorithm to be able to use adjusted distances measures. When no such adjustments are necessary, we prefer the use of publicly available implementations. For instance, we include several classifiers from the WEKA[1] toolkit like *J48* or *Support Vector Machines*.

We distinguish five subcategories of filters, namely *Input and Output*, *Preprocessing*, *Clustering*, *Classification* and *Evaluation*.

2.2.1 Input and Output

As the name suggests, input and output filters are responsible for reading and writing data. CUF offers various input filters which can read data from different sources like text files, binary files, or databases. For each input filter, a corresponding output filter is available for storing data in a particular format. Binary files are primarily used to read and write temporary results from clusterings. For text files, CUF offers two formats: *CSV* and *HTML*. The *CSV* format is most widely used. Yet, analysing *CSV* files with many columns and lines can quickly become confusing. Therefore, CUF may also write data in a formatted table and store it in *HTML* format.

We are primarily interested in analysing flow-based data streams. In experimental settings, however, network data streams are often not available. Instead, flow-based data streams are recorded from network devices and stored in *CSV* files. Each recorded flow (data point) contains an attribute named *Date first seen*. This attribute depicts the timestamp at which the corresponding network device created the flow. For simulating live data streams from these stored *CSV* files, CUF offers an additional stream simulation filter. The stream simulation filter emulates a data stream by extracting the attribute *Date first seen* from each flow and calculates the time difference between two successive flows. The filter forwards the flows with respect to the calculated time differences in the processing chain to simulate real traffic.

[1] http://www.cs.waikato.ac.nz/ml/index.html

2.2.2 Preprocessing

CUF contains a wide range of preprocessing filters which, in most cases, are not limited to network data. Preprocessing filters clean data (e.g. by removing inconsistent data), transform attribute values, or add further information. Some preprocessing filters use explicit domain knowledge to add further information to data points.

An important aspect in network data analysis is heterogeneous data, namely data points that are composed of continuous and categorical attributes. Continuous attributes take real numbers as values and thus they have a natural order in their value range. This allows simple calculations of similarities between different values. Examples for continuous attributes are the *bytes* or the *timestamp* of a network flow. The values of categorical attributes are discrete and have no natural order in their value range. This makes it hard to calculate similarities between categorical values, since simple distance measures like the *Euclidian distance* can not be applied. Examples for categorical attributes are the *Source IP Address* or *Source Port* of a flow. The mixture of continuous and categorical attributes complicates matters since most data mining algorithms can only handle either continuous or categorical attributes. To transform the problem of heterogeneous data into a well-known problem, CUF offers various preprocessing filters to discretize continuous attributes (e.g. the *Hellinger Discretization* [28] or the *Multi-Interval Discretziation using Minimal Description Length* [15]) or to transform categorical attributes to continuous attributes (e.g. a transformation to binary attributes for each categorical value).

Further, CUF contains a novel distance measure *ConDist* [45]. *ConDist* [45] is able to calculate distances between data points which contain both, continuous and categorical attributes. *ConDist* utilizes the *Minkowski distance* to calculate the distance for continuous attributes. For categorical attributes, *ConDist* uses correlated context attributes to calculate distances. It also considers the quality of information that can be extracted from the data set and automatically weights the attributes in the overall distance calculation.

2.2.3 Clustering

In general, the first goal of any type of data analysis is a better understanding of the data [27]. Clustering is an unsupervised technique, meaning it uses no a priori knowledge about the data. Consequently, appropriate clustering techniques can only be determined experimentally. In order to avoid restricting the range of techniques, CUF integrates clustering algorithms from all categories, namely partitioning (k-means and variants), hierarchical (Lance-Williams [33], ROCK [20] and extensions), grid-based and density-based algorithms (CLIQUE [2] and extensions) [27]. Any clustering algorithm may choose an appropriate distance measure, ranging from *Minkowski distances* for continuous data over the *Jaccard Index* for categorical attributes to *ConDist* [45] for heterogeneous data. The latter is even capable of self-adapting to the underlying data set.

Further, CUF integrates the stream clustering algorithms *CluStream* [1] and *DenStream* [6] both of which are based on the principle of micro and macro clusters. An online component clusters incoming data points into micro clusters. The offline component is triggered by the user, uses micro clusters as input data points, and presents a clustering result for a certain timeframe to the user. *CluStream* [1] and *DenStream* [6] can only process continuous attributes in their standard form. Therefore, CUF also integrates *HCluStream* [64] and *HDenStream* [29] which have been proposed as extensions in order to handle categorical attributes in addition to numerical ones.

2.2.4 Classification

Classification is a basic task in data mining and used to automatically assign (classify) data points into predefined groups (classes). Such a data mining model takes as input a set of labelled examples. CUF integrates different classifiers like *Neural Networks*, *Decision Trees*, *k-Nearest-Neighbour* or *Support Vector Machine*. Classification proceeds in two steps. In the first step, the classifier is trained using a labelled set of training data. The training data set contains a fixed number of class information. Each training data point X_i is composed of a constant set of attributes and a label indicating the class to which the data point belongs. The classifier learns the characteristics of the different classes and builds a model or function for the assignment (classification) of unlabelled data points. In the second step, the classifier uses this model or function to classify new (unseen) data points [56].

2.2.5 Evaluation

CUF provides a wide range of filters for result evaluation. First of all, filters are integrated to calculate default evaluation measures like *Accuracy, F1-Score, Recall* or *Precision*. However, these measures can only be calculated when a ground truth is available, like the labels in the classification setting. If there is no ground truth available, CUF offers filters which calculate intrinsic validation measures. Regarding the evaluation of data stream clustering, Hassani and Seidl examine the performance and properties of eleven internal clustering measures in [21]. Since Calinski-Harabasz [5] emerges as best internal evaluation measure it is also implemented in CUF. Besides Calinski-Harabasz, other promising evaluation methods are implemented as well (e.g. CPCQ-Index [30], CDbw-Index [9], or Davies-Bouldin-Index [13]). These intrinsic cluster validation measures evaluate the clustering results by their compactness and separability. However, these key measurements can only provide an overall assessment of the results and give no further insights.

Therefore, CUF provides various visualization filters for deeper investigations. *Data plots* (see Figure 2) may give a first impression of the data. *Bar diagrams* give an overview of the number of generated clusters and their number of data points. Further, CUF may use parallel coordinates, pixel-based visualization, or radial vi-

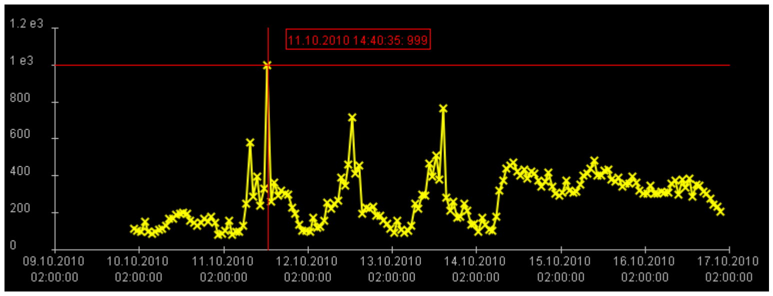


Fig. 2 Data plot which represents the number of flows per time.

sualizations to display large amounts of multi-dimensional data to a human security expert. Figure 3 shows the visualization of a network data set in parallel coordinates

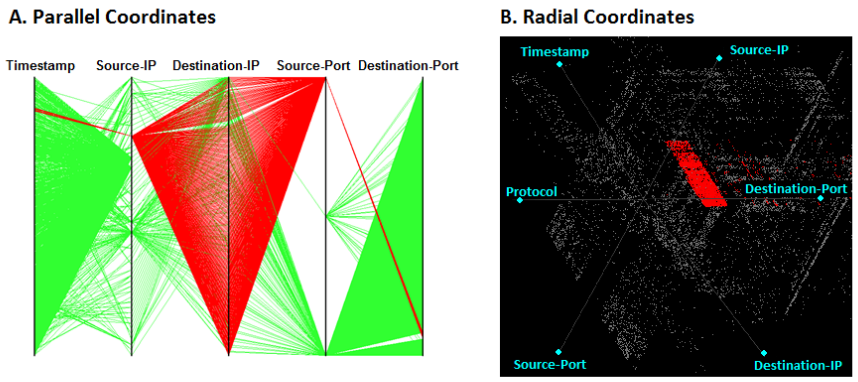


Fig. 3 Representation of a network data set with a *horizontal Port Scan* in parallel coordinates (A) and a *vertical Port Scan* in radial coordinates (B). The flows which belong to the *Port Scan* are highlighted in red.

as well as in radial coordinates. In parallel coordinates, each attribute is displayed on an axis. All axes are displayed parallel to each other on the screen. Each data point is represented as a line from left to right. The situation is slightly different for radial coordinates where each attribute is displayed on an axis as well, but data are arranged in a circular layout. However, it has to be taken into account that opposing axis are influencing each other. Other visualizations may be used to get an overview of or detailed insights into the overall result.

This way, we tackle the fundamental challenge of evaluation of anomaly based intrusion detection systems by using a broad range of evaluation technologies.

3 Online Analysis of Network Data Streams

In this section, we introduce our approach of analysing flow-based data streams with CUF. First, we investigate the general problem setting and deduce some implications for our approach in Section 3.1. Section 3.2 gives an overview of the proposed approach and provides the underlying ideas. Then, the integration of additional domain knowledge is described in Section 3.3. The Sections 3.5, 3.6 and 3.7 describe the three different views in which the incoming flow-based data stream is analysed.

3.1 General Problem Setting

The objective of our research is to identify malicious network traffic in flow-based data streams using data mining methods. Hence, we now discuss typical attack scenarios, available data and necessary data preparation steps.

3.1.1 Attack Scenarios

Attack scenarios often follow predefined patterns. Literature usually distinguishes various phases of attacks. A popular definition by Skoudis and Liston [51] is depicted in Figure 4 and contains the following phases:

1. Reconnaissance
2. Scanning
3. Gaining Access
4. Maintaining Access
5. Covering Tracks

In each phase, different methods are used to reach the corresponding goals.

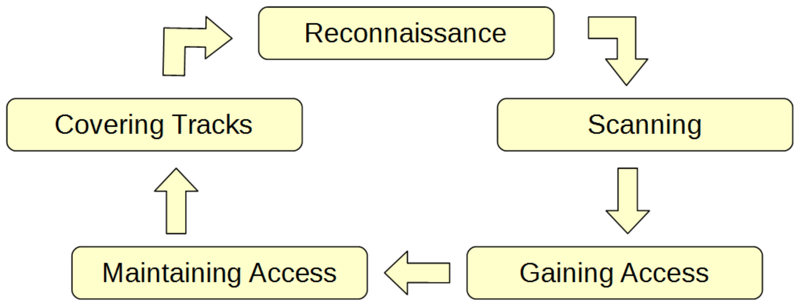


Fig. 4 Overview of the five attack phases.

In the first phase (*Reconnaissance*), the attacker tries to gather as much information as possible about the target system. Skoudis and Liston [51] identify various *Reconnaissance* techniques, e.g. *web searches*, *whois database analysis*, *domain name systems*, *social engineering* or *dumpster diving*. A popular starting point in this phase would be to search the companies website. In case of *social engineering*, names of the current CEOs or other employees (e.g. server administrators) would be targeted. Attackers commonly use such information by acting as an administrator via email to get critical data from a trustful employee. *Domain name systems* can be used to identify concrete IP ranges. Alternatively, it is possible to retrieve usernames, (inital) passwords and internal *IP Addresses* via *dumpster diving*.

Scanning is the second phase and uses information which was gathered during *Reconnaissance* to examine the target system. In essence, the *Scanning* phase is an extension of *Reconnaissance*. Directly accessible hosts can be identified by *IP Range Scans*. Then, *Port Scans* are used to identify open ports and thereby potential attacking points. A variety of powerful scanning tools such as *nmap*[2] exists. Further, these scanning tools contain features for deeper analysis like operation system detection.

In the third phase (*Gaining Access*), the actual attack takes place. Typical attacks in this phase are *Denial of Service* (*DoS*), *SQL injections*, *PHP exploits*, or *Brute Force attacks*. Exemplary, *DoS* attacks try to make the target system unavailable by flooding the target with illegitimate packets [8], while *SSH Brute Force* attacks are used to gain access to a remote host by repeatedly trying to guess the correct login information.

After gaining access, attackers usually want to maintain it. This can be accomplished in phase four (*Maintaining Access*) via typical techniques like *trojan horses*, *bots*, *backdoors* or *rootkits* [51]. The compromised system can be used as starting base for deeper exploration of the target network and to collect information for further attacks.

The *Covering Tracks* phase completes the cycle of the attack scenario. Attackers typically prefer to stay hidden, so they can maintain control of the systems as mentioned in phase four. Staying hidden buys time for the attackers and enables them to steal data, consume CPU cycles, launch other attacks or just maintain the control for future actions [51].

Attack scenarios do not necessarily include all phases. For example, an insider (e.g. an employee) already holds information that an outsider would acquire in phase one. Consequently, insiders would start at phase two or even three. In addition, not all attackers are interested in keeping access and wiping out their traces. In these cases the phases four and five are optional.

[2] https://nmap.org/

3.1.2 Flow-based Data

As already mentioned above, we focus on flow-based data for intrusion and insider threat detection. To reiterate, flows contain meta information about connections between two network components. A flow is identified by the default five tuple: *Source IP Address, Source Port, Destination IP Address, Destination Port* and *Transport Protocol*. We capture flows in unidirectional *NetFlow* format [10] which typically contains the attributes shown in Table 1. These attributes are typical in flow-based

Table 1 Overview of the used *NetFlow* attributes in our approach. The third column gives a short description of the attributes.

Nr.	Name	Description
1	Src IP	Source IP Address
2	Src Port	Source Port
3	Dest IP	Destination IP Address
4	Dest Port	Destination Port
5	Proto	Transport Protocol (e.g. ICMP, TCP, or UDP)
6	Date first seen	Start time flow first seen
7	Duration	Duration of the flow
8	Bytes	Number of transmitted bytes
9	Packets	Number of transmitted packets
10	Flags	*OR* concatenation of all TCP Flags

data and also available in other flow standards like *IPFIX* [11] or *sFlow* [40]. *NetFlow* terminates a flow record in two cases: (1) a flow receives no data within α seconds after the last packet arrived (inactive timeout) or (2) a flow has been open for β seconds (active timeout). By default, *NetFlow* uses the values $\alpha = 15$ and $\beta = 1800$.

NIDS usually operate either on flow-based data or on packet-based data. We analyse flow-based data for several reasons. In contrast to packet-based data, the amount of data can be reduced, fewer privacy concerns are raised, and the problem of encrypted payloads is bypassed. Further, problems associated with the variability of input data [52] are avoided. Flows have a standard definition and can be easily retrieved by central network components. Another advantage of flows is that huge parts of the company network can be observed. For example, firewall log files would limit the analysed data to the traffic which passes the firewall. Consequently, insider attacks do not appear in these log files as they usually do not pass the firewall. In contrast to that, traffic between internal network components always passes switches or backbones. In conclusion, the use of flows allows to analyse the whole company network traffic independently of its origin.

Flows come as either unidirectional or bidirectional flows. Unidirectional flows aggregate those packets from host A to host B into one flow that have identical five tuples. The packets from host B to host A are merged to another unidirectional flow. In contrast, bidirectional flows contain the traffic from host A to host B as well as vice versa. Consequently, bidirectional flows contain more information. However, we decided to use unidirectional flows since company backbones often contain asymmetric routing [23] which would distort the information in bidirectional flows.

The individual phases of an attack (see Section 3.1.1) have different effects on flow-based data. The *Reconnaissance* phase has no influence on the data since methods like dumpster diving or social engineering generate no observable network traffic within the company. In comparison, the other four phases generate observable network traffic.

It should be noted that the detection of attacks using host-based log files could sometimes be easier than the analysis of flows, e.g. failed *ssh logins* are stored in the ssh-log-file. However, regarding phase five (*Covering Tracks*), attackers usually manipulate the log files on the host to wipe their traces. Since we use flow-based data of network components and no host-based log files, the covering of tracks fails in our anomaly based intrusion detection system. This would only be possible if the attacker hacks the network components and manipulates the flows.

3.1.3 Data Preparation

We follow the typical phases of the CRISP (Cross-Industry Standard Process for Data Mining) model [49] which defines a standard model for Data Mining workflows. Business understanding and data preparation are necessary prerequisites for the application of Data Mining methods. In our setting, business understanding corresponds to the analysis of hacking phases (Section 3.1.1) while data understanding complies with the analysis of flow-based data (Section 3.1.2).

The next step in the workflow is the data preparation before data mining methods can be applied. Reasons are the need for attributes which represents hidden information or the requirement of some data mining methods, as they are only able to handle some types of attributes. For example, when using *Neuronal Networks* the input data should consist of exclusively continuous attributes. Hence, categorical attributes like *IP Addresses* have to be transformed. Several aspects have to be considered when preprocessing flow-based data. Clustering (Section 2.2.3), classification (Section 2.2.4) and outlier detection algorithms are usually based on a definition of distance or similarity. Applying these algorithms directly on the attributes of flow-based data (see Table 1) could lead to fatal problems. Therefore, it is necessary to abstract certain attributes of the flow in a preprocessing step. For example, when facing the same attack twice, but from a different *Source IP Address* and on a different *Destination IP Address* to a later time, at least three of the ten attributes of a flow will differ. As a consequence, the flows of the two attacks are dissimilar and categorized to different groups.

3.1.4 Implications

Considering our above analysis of the general problem setting, where we investigated attack scenarios, the underlying flow-based data and necessary data preparation steps, we can draw several conclusions:

(1) A basic assumption is that due to the big number of various applications and services it is nearly impossible to decide if a single flow is normal or malicious traffic based on the available attributes (see Table 1). This assumption is supported by the fact that normal user behaviour and malicious user behaviour are characterized by sequences of flows. Lets illustrate this by an example. Assume a *Vertical Port Scan* attack. Here, the attacker scans some or all open ports on a target systems [55]. Since *Source Port* and *Destination Port* are keys of the default five tuple for creating flows, each scanned port generates a new flow. Another example is the loading of a web page. Often different pictures are reloaded or other web pages are included. In such cases, the client opens various *Source Ports* or sends request to different web servers (different *Destination IP Addresses*). For these reasons, it makes more sense to collect multiple flows for each host rather than to analyse each flow separately.

(2) Different attack phases have different effects on flow-based data. In the second phase (*Scanning*) different services and or hosts are targeted, whereas in the third phase (*Gaining Access*), a concrete service of a host is attacked. In the fourth phase (*Maintaining Access*), flow characteristics like transmitted *Bytes* or *Packets* seem to be normal, only the origin of the connections are suspicious. Consequently, it makes more sense to analyse the flow-based data from different views to detect different attack phases.

(3) The information within flow-based data is limited. Therefore, flows should be enriched with as much information about the network as possible using domain knowledge.

3.2 Outline of the Proposed Approach

This section provides an overview of the proposed approach and discusses the underlying ideas. Figure 5 shows the essential components of our approach.

The *IP Address Info Filter* is the first filter in the processing chain. It receives the flow-based data stream X_S from central network components and incorporates domain knowledge about the network (see Section 3.3). Data is passed through the *Service Detection Filter* (Section 3.4) to identify the services of each flow (e.g. *SSH*, *DNS* or *HTTP*). The *Collecting Filter* is the central component of our approach. It also incorporates domain knowledge about the network and receives the enriched flow-based data stream X_S from the *Service Detection Filter*. Based on the observations above, in its first step, the *Collecting Filter* collects all incoming flows for each user separately. User identification is based on the *Source IP Address* of the flow. A parameter δ controls the windows-size (in seconds) of flows which are collected for each user. The larger the parameter δ, the more memory is necessary but in con-

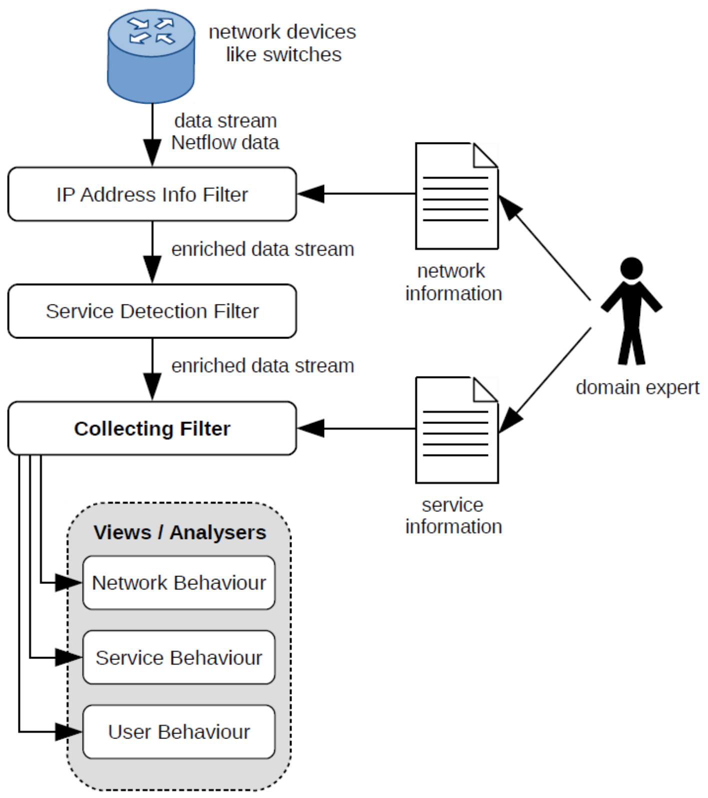


Fig. 5 Overview of the proposed approach.

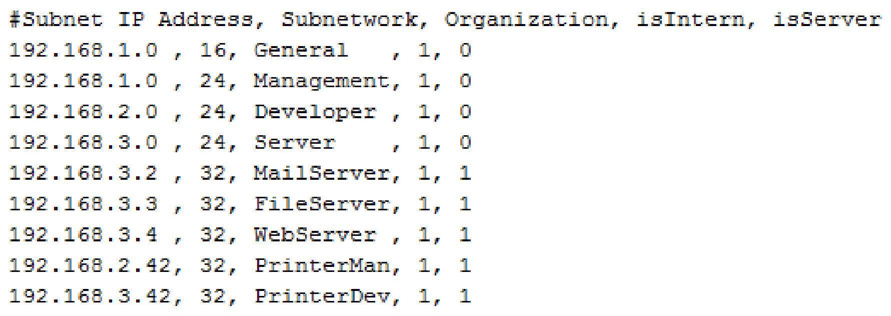
sequence, the quality of the calculated summary of flows increases. The *Collecting Filter* creates one *Network data point* for each user and each time window. For each user and identified service within the time window a *Service data point* and *User data point* is created. Each of these data points is created for investigating the user flows from a specific view, namely *Network Behaviour Analyser*, *Service Behaviour Analyser* and *User Behaviour Analyser* in Figure 5. The *Network data point* contains specific information about the users' network behaviour and is described in Section 3.5. The *Service data point* contains specific information about the usage of the concrete service and is described in Section 3.6. The *User data point* contains specific information, e.g. if the behaviour is typical for a user or not. It is described in 3.7. We argue that incorporating domain knowledge from IT security experts and other flows allow us to create more meaningful attributes for downstream analysis methods.

The *IP Address Info Filter*, *Service Detection Filter*, *Collecting Filter*, *Network Behaviour*, *Service Behaviour* and *User Behaviour* are implemented as separate filters in CUF which can be run independently and in parallel.

3.3 Integration of Domain Knowledge

Arguably, the performance of a NIDS increases with the amount of domain specific information about the network. Therefore, we integrate more detailed information about *Source IP Address*, *Destination IP Address*, *Source Port* and *Destination Port* in our system.

First, we integrate additional information about the *Source IP Address* and *Destination IP Address* through the *IP Address Info Filter*. Therefore, the network administrator has to set up a *csv* file with all network addresses of the company. Further, the administrator has to add each *IP Address* of internal servers. Figure 6 shows such a sample *csv* file.



```
#Subnet IP Address, Subnetwork, Organization, isIntern, isServer
192.168.1.0 , 16, General    , 1, 0
192.168.1.0 , 24, Management, 1, 0
192.168.2.0 , 24, Developer , 1, 0
192.168.3.0 , 24, Server     , 1, 0
192.168.3.2 , 32, MailServer, 1, 1
192.168.3.3 , 32, FileServer, 1, 1
192.168.3.4 , 32, WebServer , 1, 1
192.168.2.42, 32, PrinterMan, 1, 1
192.168.3.42, 32, PrinterDev, 1, 1
```

Fig. 6 Exemplary configuration file for integrating domain knowledge.

The first column in Figure 6 is a specific *IP Address* or the address of an internal subnet. The second column defines the size of the subnet in column one, where 32 means that this entry refers to a single *IP Address*. The third column describes the organization of the subnet or the function of the particular *IP Address*. The last two columns indicate if the *IP Address/Range* is internal and if the *IP Address/Range* is a server. Normally, all servers are part of the company network and marked as internal. However, the configuration file allows the administrator to add special roles to extern but known servers (e.g. servers of other companies for joint projects). Any *IP Address/Range* that can not be matched to a given entry within the configuration file is treated as an external address. Based on this information, the *IP Address Info Filter* adds the corresponding organization and the two flags *isIntern* and *isServer* to each *Source IP Address* and *Destination IP Address*.

The second configuration file contains information about used ports in the company. Here, the network administrator has to flag open ports of internal servers, e.g. *port* 20 and 21 for *FTP* or *port* 22 for *SSH*. By default, all *ports* between 0 and 1023 (the standardized ports) are considered as open ports. Requested connections to non open ports increase the level of suspiciousness. This configuration file is read by the *Collecting filter* for identifying client and servers within a flow.

3.4 Service Detection Filter

The *Service Detection Filter* classifies each flow with respect to their services (e.g. *HTTP*, *SSH*, *DNS* or *FTP*). Right now, this filter uses a common identification method which is based on evaluating known port numbers assigned by the *Internet Assigned Numbers Authority* (*IANA*)[3].

Unfortunately, this approach is no longer viable because many applications do not use fixed port numbers [65]. Another problem when evaluating known port numbers is that many applications tunnel their traffic through port 80 (e.g. Skype).

Therefore, we intend to integrate more sophisticated service detection algorithms in the future. Nguyen et al. [37] and Valenti et al. [59] broadly review traffic classification using data mining approaches which is not limited to flow-based data. Several approaches for flow-based service classification have been published ([32] and [65]). Moore and Zuev [32] show the effectiveness of a Naive Bayes estimator for flow-based traffic classification. Zander et al. [65] use *autoclass*, an unsupervised Bayesian classifier which learns classes inherent in a training data set with unclassified objects. A more recent approach of service classification using *NetFlow* data is given by Rossi and Valenti [46].

3.5 Network Behaviour Analyser

The *Network Behaviour Analyser* evaluates hosts with respect to their general network behaviour. Consequently, this analyser primarily checks if the number and the kind of connections are normal or suspicious for a specific host. The primary goal is to identify activities in the *Scanning* phase (see Section 3.1.1). Therefore, the main attacks in this scenario are *IP Range Scans* or *Port Scans*. For the detection of *Port Scans*, a more detailed analysis is required. *Port Scans* can be grouped into *horizontal scans* and *vertical scans*. In case of more common *horizontal scans* the attacker exploits a specific service and scans numerous hosts for the corresponding port [55]. In contrast, *vertical scans* target some or all ports of a single host.

Since the scanning behaviour of attacker and victim hosts are different for *TCP* and *UDP*, they need to be treated separately. The most common *TCP* scan is the *SYN-scan*. In this case, the attacker sends the initialization request of the 3-Way-Handshake. If the port is open, a *SYN-ACK*-response is sent by the victim. Otherwise, the victim host responds with a *RST-Flag*. It should be noted that there are different approaches of *TCP* scans, e.g. sending a *FIN* flag instead of the initialization *SYN* flag for bypassing firewall rules. These approaches are described in more detail in the documentation of the popular *nmap*[4] tool.

[3] http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

[4] https://nmap.org/

Scanning *UDP* ports differs fundamentally from scanning *TCP* ports. Successful addressing *UDP* ports does not necessarily render a response. However the same behaviour can be observed if a firewall or other security mechanisms blocks the request. If the attacker addresses a closed *UDP* port, the victim sends an *ICMP* unreachable message in return. However, most operating systems limit the number of *ICMP* unreachable messages to one per second.

In consequence, it is easier to detect the targeted victim instead of the attacker, since victims follow rules predefined by protocols. In contrast, attackers may vary their behaviour regarding the protocol to trick the systems. Also, it is more likely to detect the scanning of closed ports due to the atypical behaviour of request-response. Due to these observations, data points are calculated in the *Collecting Filter* with corresponding attributes for this view. The calculated attributes contain values like the number of flows within a time window, the number of sent or received *RST*-Flags, the number of sent or received *ICMP* unreachable messages as well as the number of requested ports per *Destination IP Address*. Since we consider both, the view of the victim and the attacker, it is easier to detect distributed *Port Scans* too.

In preliminary experiments, we applied various classifiers (*J48 Decision Tree*, *k-Nearest-Neighbour*, *Naive Bayes* or *SVM*) for the detection of *IP Range Scans* and *Port Scans*. The proposed approach seems to work on our emulated data sets. We will describe the process of data emulation in Section 4.

Several other methods which use similar approaches for *Port Scan* detection are discussed in literature. Methods like Time-based Access Pattern, and Sequential hypothesis testing (TAPS) [54] use the ratio of *Destination IP Addresses* and *Destination Ports* to identify scanners. If the ratio exceeds a threshold, the host is marked as scanner. Threshold Random Walk (TRW) [24] assumes that a scanner has more failed connections than a legitimate client. For identification of failed connections, TRW also evaluates the *TCP-Flags*.

3.6 Service Behaviour Analyser

The *Service Behaviour Analyser* evaluates the hosts from the view of their correct usage of services. Consequently, the main goal of this analyser is to check if the use of the current service is normal or malicious for a host. Primary target is to recognize the *Gaining Access* phase mentioned in Section 3.1.1. *DoS* or *SSH Brute Force* attacks are typical representatives.

For the detection of misused services, it is necessary to collect all flows of this service within the time window. Therefore, we use the service attribute added by the *Service Detection Filter* (Section 3.4). All flows of the host within a time window which share the same *service* and the same *Destination IP Address* are collected. Based on these collected flows, the *Collecting Filter* calculates data points with adjusted attributes for this view. The calculated attributes contain values like the sum of transmitted *Bytes* and *Packets*, the duration of the flows, or the number of flows. More useful attributes like the number of open connections or the number of successfully closed connections can be derived from *TCP Flags* (if available). The *Collecting Filter* also builts attributes which give additional information about the source and destination using the domain knowledge of Section 3.3.

Again, we applied various classifiers (*J48 Decision Tree*, *k-Nearest-Neighbour*, *Naive Bayes* or *SVM*) for the detection of misused services. Preliminary experiments regarding the detection of *SSH Brute Force* attacks on our emulated data sets (Section 4) show promising results.

Quite a few similar approaches have already been published. For instance, Wagner et al. [60] proposed a kernel for anomaly detection, especially for *DDoS* attacks. Their kernel includes all recorded flows within a time window and considers the *IP Addresses* and *Bytes* of the flows. Najafabadi et al. [34] propose an approach for detecting *SSH Brute Force* attacks using aggregated *NetFlow* data. The authors incorporate domain knowledge about *SSH Brute Force* attacks, extract discriminating attributes from *NetFlows* and use classifiers for the detection. Hellemons et al. [22] propose *SSHCure*, a flow based SSH intrusion detection system. The authors analyse the model of *SSH Brute Force* attacks and derive attributes for their detection in *NetFlow* data. However, unlike these approaches, our approach is more general and not limited to one specific source.

3.7 User Behaviour Analyser

The *User Behaviour Analyser* filter evaluates the hosts with respect to their used services being typical. The main goal of this analyser is to recognize if the current connection is normal or malicious for this user and to identify already infected and misused hosts. Primary target is to recognize the *Maintaining Access* and *Covering Tracks* phases mentioned in Section 3.1.1.

Once a server is infected, the attacker knows a valid combination of username and password and is able to start a *SSH* session to that server. In this case, the traffic characteristics like transmitted *bytes*, *packets* or *duration* seem to be legitimate. Therefore, the *Collecting Filter* calculates data points for this view which strongly consider the source, destination and services of the flows. Here, the domain knowledge of Section 3.3 and Section 3.4 is used. For example, the calculated attributes describe if the *Source (Destination) IP Address* is internal or external, if *Source (Destination) IP Address* is a server or a client, and which organizations (see Figure 6) the *Source (Destination) IP Address* belongs to. Further, the identified service by the *Service Detection Filter* is added as an additional attribute.

Right now, we use a simple rule learner which generates rules regarding normal and malicious behaviour. If an unknown combination occurs, the corresponding connection information is sent to the domain experts for further investigation. Rules generated in this case would look like the following:

$organization_{source} = extern \land organization_{destination} = server \land service = SSH \rightarrow Malicious$

This example would imply malicious behaviour, since we would not expect a valid SSH connection to an internal server from outside the company network.

4 Data Generation

Labelled publicly available data sets are necessary for proper comparison and evaluation of network based intrusion detection systems. However, evaluation of our system proves to be difficult due to the lack of up-to-date flow-based data sets. Many existing data sets are not publicly available due to privacy concerns. Those which are publicly available often do not reflect current trends or lack certain statistical characteristics [50]. Furthermore, correct labelling of real data proves to be difficult due to the massive and in-transparent generation of traffic in networks. In order to overcome these problems, we create labelled flow-based data sets through emulation of user activities in a virtual environment using OpenStack [39].

In this section, some prominent existing data sets are presented as well as our own approach to generate data sets for IDS evaluation.

4.1 Existing Data Sets

DARPA98 and DARPA99 from the MIT Lincoln Laboratory were among the first standard packet-based data sets published for evaluation purposes. Those data sets were created by capturing simulated traffic of a small US Air Force base with limited personnel via tcpdump [26]. The MIT Lincoln Laboratory also provided the KDD CUP 99 data set, which is a modified version of the DARPA98 data set [17]. Each data point of the KDD data set consists of 41 attributes. The KDD data set, however, has a few problems, one being the huge number of redundant records. To overcome these problems, the NSL-KDD data set was generated by Tavallaee et al. [57]. Since publicly available data sets are sparse due to privacy concerns, a lot of today's work is based on the DARPA ([41], [47]) and KDD ([7], [12] and [42]) data sets. As DARPA data sets were created more than 17 years ago, it is questionable if they still reflect relevant up-to-date scenarios appropriate for IDS evaluation ([19], [62] and [66]).

Besides the data sets being outdated, conversion of packet-based to flow-based data sets turns out to be tricky, if the data set is not available in a standard packet-based format like *pcap*. Since we prefer to analyse network flows naturally flow-based data sets would be best. Sperotto et al. [53] created one of the first publicly available flow-based labelled datasets by monitoring a single honeypot. Due to the traffic being recorded by monitoring a honeypot, the data set mainly consists of malicious data. Thus, the detection of false positives could not be determined during the evaluation [53]. For a more comprehensive IDS evaluation, a more balanced data set would be preferable.

In 2014, Wheelus et al. [62] presented the SANTA dataset which consists of real traffic as well as penetration testing attack data. The attack data was labelled via manual analysis [62]. In 2015, Zuech et al. [66] introduced a two part dataset named IRSC. The IRSC set comprises a netflow data set as well as full packet capture set. The data sets were labelled manually for uncontrolled attacks and via an IP filter for controlled attacks [66]. Although the two flow-based data sets ([62], [66]) are up-to-date, they are not publicly available as of now.

Another labelled flow-based data set is CTU-13. It was especially created for training botnet detection algorithms. CTU-13 contains a variety of botnet traffic mixed with background traffic coming from a real network [16].

Shiravi et al. [50] introduce a dynamic data set approach based on profiles containing abstract representations of events and network behaviour. This allows to generate reproducible, modified and extended data sets for better comparison of different IDS. We use unidirectional flows whereas [50] contains bidirectional ones. Converting flows might constitute a viable approach, but would require some effort for re-labelling.

In conclusion, the validation of IDS through data sets in general seems to be difficult due to few publicly available data sets. Some of the most widely used data sets are outdated ([17], [48] and [57]) or only contain malicious data which complicates the attempt for comprehensive validation. Since we use unidirectional flows some of the presented data sets would only be applicable after conversion and re-labelling of the data [50]. Some data sets are only for specific attack scenarios [16] and other promising approaches are not publicly available ([62], [66]).

4.2 Data Set Emulation

Małowidzki et al. [31] define a list of characteristics for a good data set. A good data set should contain recent, realistic and labelled data. It should be rich containing all the typical attacks met in the wild as well as be correct regarding operating cycles in enterprises, e.g. working hours. We try to meet these the requirements listed in [31] in our approach using *OpenStack*.

OpenStack is an open source software platform which allows the creation of virtual networks and virtual machines. This platform provides certain advantages when generating flow-based data sets. A test environment can be easily scaled by using virtual machines and therefore allows to generate data sets of any size. Furthermore, one has full control over the environment including the control of the network traffic. This ensures the correct capturing and labelling of truly clean flow-based data sets which do not contain any harmful scenarios. Conversely, it is also possible to include a host as an attacker and clearly label the data generated from the attacker as malicious. A data set acquired from a real enterprise network can never be labelled with the same quality. However, it is of upmost importance to emulate the network activity as authentic as possible to generate viable data sets which is difficult for synthetic data. To reach that goal, we emulate a sample small company network with different subnets containing various servers and simulated clients and record generated network traffic in unidirectional *NetFlow* format.

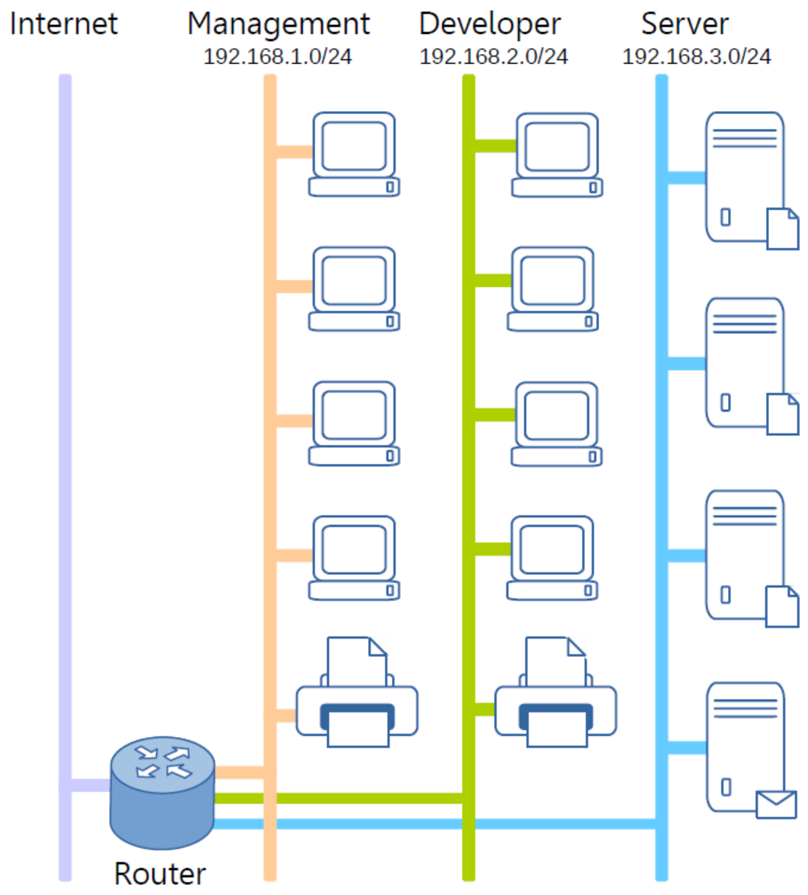


Fig. 7 An overview of a sample network in our *OpenStack* environment. The *Router* separates the internal network from the internet and acts as firewall. The internal network structure with three subnets containing several clients and servers is shown on the right.

4.2.1 Test Environment

Our sample network currently comprises three subnets, namely a management, a developer and a server subnet. Each subnet contains several servers or clients. A schematic overview of the network is shown in Figure 7.

The servers provide different services like printing on a network printer, email, file sharing and hosting websites. The clients, which are windows and linux machines, emulate user activity through various Python scripts. All three subnets are connected through a virtual router. *Open vSwitch* is used to monitor the traffic in the virtual network and to capture it in *NetFlow* format.

As already mentioned, *OpenStack* easily enables scaling of the desired test environment. Our setup scripts contain the automatic update of the newly created hosts as well as the installation of tools required to emulate normal user activity and setting up an automatic startup of the said activities. Using this approach, new hosts can easily be added to the test environment if needed.

4.2.2 Generation of Normal Data

Several aspects have to be considered when emulating normal user behaviour through scripts. Ideally, scripts should:

1. run on all established operating systems
2. include all computerized activities of typical employees
3. be free of periodic activities
4. consider different working methods
5. consider working hours and idling of the employees

to be as close to real user behaviour as possible. Using this list, we have setup a kind of user model within the scripts which reflects the real behaviour of the users. The fact that we use the platform-independent language Python resolves the first issue above.

To be realistic, emulated clients have to simulate a broad list of activities. Those activities include all quintessential computerized activities of a typical employee like browsing the web for work-related matters (e.g. companies websites or doing research via search engines), private browsing (e.g. facebook, certain newsgroups or blogs), file sharing, writing emails or printing on a network printer. Also, it is important to ensure a varying length of files regarding the latter two activities and varying types and numbers of attachments in the activity of sending emails.

To emulate user behaviour as realistically as possible it is vital to not simply repeat given activities periodically but rather involve randomness to some extend. Conversely, the activities should not be totally random but reflect different working methods of the employees. It should be noted that some employees are more likely to write a great amount of emails but rarely use a file sharing service. Configuration files can be altered to model the different characteristica of each type of employee.

During a typical work day, employees are not permanently using “their” assigned computer. Meetings, offline work or coffee breaks should also be taken into account. The last point of the list is similar. The activities typically focus on the employees working hours and drop during lunch break or in the evening.

It is to be noted that while we do our best to satisfy the requirements for generating as realistic data as possible, we do not claim to capture emulating viable behaviour. In essence, one needs to ensure making the simulation as realistic as possible by modelling time and size distributions. While those distributions are not implemented yet, in this setup new ideas for improving the emulation can be easily integrated and tested.

4.2.3 Generation of Malicious Data

In addition to clean non-threatening flow-based data, malicious data behaviour is needed in a fully labelled data set. This problem is solved by inserting one or more hosts deployed as attacker. Subsequently, it is easy to label malicious flows using the structure of the *NetFlow* data as well as additional knowledge like the *Source IP Address* of the attacker and the exact timestamp of the attack. Again, Python scripts are used to create malicious behaviour like *DoS* attacks, *Port Scans*, or *SSH brute force* attacks. If malicious data is inserted via scripts, it is always possible to include new attacks by simply writing new scripts. Thus, up-to-date data sets can be generated at any time.

5 Related Work

Work on network based anomaly detection methods for intrusion and insider threat detection can be separated into packet-based and flow-based anomaly detection. A comprehensive review of both methods is given in Bhuyan et al. [3]. A recent survey of data mining and machine learning methods for cyber security intrusion detection is published by [4]. Further, Weller-Fahy et al. [61] published an overview of similarity measures which are used for anomaly based network intrusion detection.

Since the proposed approach is based on flow-based data, the following review does not consider packet-based methods. We categorize flow-based anomaly detection methods into (I) treating each flow separately, (II) aggregating all flows over time windows and (III) aggregating flows of single hosts over time windows.

Category I

Winter et al. [63] propose a flow-based anomaly detection method of category (I). The authors use an One-Class SVM and train their system with malicious flows instead of benign flows since data mining methods are better at finding similarities than outliers. For learning the One-Class SVM, the honeypot data set of [53] is used. During the evaluation phase, each flow within the class is considered as malicious and each outlier is considered as normal behaviour. Another approach of this category is proposed by Tran et al. [58]. The basis of their system is a block-based neural network (BBNN) integrated within an FPGA. They extract four attributes (*Packets*, *Bytes*, *Duration* and *Flags*) from each flow as input for their IDS. The authors compared their system against SVM and Naive Bayes classifier and outperformed them in an experimental evaluation. Najafabadi et al. [35] use four different classification algorithms for *SSH Brute Force* detection. The authors selected 8 attributes from the flow-based data and were able to detect the attacks. The detection of *RUDY* attacks using classification algorithms is studied by Najafabadi et al. [36]. *RUDY* is an *application layer DoS attack* which generates much less traffic than traditional *DoS* attacks. The authors use the enriched flow-based SANTA dataset [62] for evaluation. The flows in this data set contain additional attributes which are calculated based on full packet captures.

Category II

Approaches from category (II) aggregate all flows within a certain time window. Wagner et al. [60] developed a special kernel function for anomaly detection. The authors divide the data stream in equally sized time windows and consider each time window as a data point for their kernel. The kernel function takes information about the *Source (Destination) IP Address* and the transferred *Bytes* of all flows within the time window. The authors integrate their kernel function in an One-Class SVM and evaluate their approach in the context of an internet service provider (ISP). An entropy based anomaly detection approach is presented in [38]. Here, the authors divide the data stream in five minute intervals and calculate for each interval seven different distributions considering flow-header attributes and behavioural attributes. Based on these distributions, entropy values are calculated which are used for anomaly detection.

Category III

Approaches from category (III) use more preprocessing algorithms in the data mining workflow and do not work directly on flow-based data. These approaches aggregate for each host the flows over a time window and calculate new attributes based on these aggregations. BClus [16] uses this approach for behavioural-based botnet detection. At first, they divide the flow data stream in time windows. Then, flows are aggregated by *Source IP Address* for each time window. For each aggregation, new attributes (e.g. amount of unique destination IP addresses contacted by this *Source IP Address*) are calculated and used for further analysis. The authors evaluate their botnet detection approach using the CTU-13 Malware data set. Another representative of this category is proposed by Najafabadi et al. [34]. The authors aggregate all *NetFlows* with the same *Source IP Address*, *Destination IP Address* and *Destination Port* in 5 minute intervals. Based on these aggregations, new attributes are calculated like the *average transmitted bytes* or the *standard deviation of the transmitted bytes*. Then, Najafabadi et al. [34] train different classifiers and use them for the detection of *SSH Brute Force* attacks.

Besides these three categories we want to also mention the Apache Spot[5] framework. Apache Spot is an open source framework for analysing packet- and flow based network traffic on Hadoop. This framework allows to use machine learning algorithms for identifying malicious network traffic. Another network based anomaly

[5] http://open-network-insight.org/

detection system is proposed by Rehak et al. [43, 44]. Their system *Camnep* uses various anomaly detectors and combine their results to decide if the network traffic is normal or malicious. The individual filters use direct attributes from *NetFlow* and additional context attributes. For calculating these attributes, the anomaly detectors can access all flows of a 5 minute window.

Our proposed approach neither handles each flow separately (category I) nor simply aggregates all flows within a time window (category II). Instead, it follows the approach of the third category and collects all flows for each host within a time window. However, in contrast to the third category, the proposed approach generates more than one data point for each collection. The generation of multiple data points for each collection allows us to calculate more adapted data points which describe the network, service and user behaviour of the hosts. Further, we do not try to recognize all attack types with a single classifier like [60] or [63]. Instead, we analyse the calculated data points from different views and develop for each view a separate detection engine.

6 Summary and Future Work

Company data needs to be protected against unauthorized access. Attempts to gain such unauthorized access may originate from outside the company network, but may also be traced back to insiders. In this contribution, we report on ongoing work that aims to assist domain experts by highlighting significant incidents. To that end, we develop the Coburg Utility Framework (CUF), a toolset to support the analysis using data mining methods. CUF is based on a pipes-and-filter architecture and contains various machine learning algorithms, visualization tools, preprocessing algorithms and evaluation algorithms. Due to its architecture, CUF is very flexible and can be easily extended or adapted.

While our earlier work focused on offline analysis of data from company networks, we currently work on extending our approach to online analysis of data streams. In particular, our approach builds upon flow-based data since this allows to reduce the volume of data, bypasses the problem of encrypted payloads and leads to less privacy concerns compared to packet-based data. Further, we include additional domain knowledge to support otherwise uniformed analysis methods. In particular, our approach includes analyses from three different points of view: the first perspective is concerned with the general network behaviour of hosts, the second perspective focuses on the usage of services, and the third perspective concentrates on user behaviour.

Evaluation of anomaly-based intrusion and insider threat detection approaches presupposes test data that are labelled in terms of normal or malicious behaviour. As it turns out, such labelled test data are hard to obtain since such data sets are rarely available for public use. Likewise, real data from company networks cannot be used easily due to the lack of reliable labels. Therefore, we devised and set up a virtual environment for generating flow-based network data. To that end, an *OpenStack* environment may emulate a flexible configuration of servers and clients. Client behaviour within that virtual network is generated through randomized Python scripts. This allows to record real flow-based traffic with typical user activities and further to simulate attacks like *Port Scans*, *DoS* or *SSH Brute Force*.

Future activities of our research are directed towards refining the flow-based analysis approach and provide appropriate visualization tools for data streams, e.g. by extending well-known visualization approaches such as parallel coordinates to data streams. In addition, the simulation environment needs to be expanded allowing to generate even more realistic data sets as a basis to validate and refine our anomaly-based intrusion and insider threat detection approach more thoroughly.

Acknowledgements This work is funded by the Bavarian Ministry for Economic affairs through the WISENT project (grant no. IUK 452/002).

References

1. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: International Conference on very large data bases (VLDB), pp. 81–92. Morgan Kaufmann (2003)
2. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data for data mining applications. In: International Conference on Management of Data, pp. 94–105. ACM Press (1998)
3. Bhuyan, M.H., Bhattacharyya, D.K., Kalita, J.K.: Network anomaly detection: Methods, systems and tools. IEEE Communications Surveys & Tutorials **16**(1), 303–336 (2014)
4. Buczak, A.L., Guven, E.: A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys & Tutorials **18**(2), 1153–1176 (2016)
5. Caliński, T., Harabasz, J.: A dendrite method for cluster analysis. Communications in Statistics-theory and Methods **3**(1), 1–27 (1974)
6. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: SIAM International Conference on Data Minning (SDM), vol. 6, pp. 328–339. Society for Industrial and Applied Mathematics (2006)
7. Chae, H.s., Jo, B.o., Choi, S.H., Park, T.: Feature selection for intrusion detection using NSL-KDD. Recent Advances in Computer Science pp. 978–960 (2015)
8. Chen, E.Y.: Detecting DoS attacks on SIP systems. In: IEEE Workshop on VoIP Management and Security, 2006., pp. 53–58. IEEE (2006)
9. Chou, C.H., Su, M.C., Lai, E.: A new cluster validity measure and its application to image compression. Pattern Analysis and Applications **7**(2), 205–220 (2004)
10. Claise, B.: Cisco systems netflow services export version 9. RFC 3954 (2004)
11. Claise, B.: Specification of the ip flow information export (IPFIX) protocol for the exchange of ip traffic flow information. RFC 5101 (2008)
12. Datti, R., Verma, B.: B.: Feature reduction for intrusion detection using linear discriminant analysis. International Journal on Engineering Science and Technology **1**(2) (2010)
13. Davies, D.L., Bouldin, D.W.: A cluster separation measure. IEEE transactions on pattern analysis and machine intelligence **1**(2), 224–227 (1979)
14. Depren, O., Topallar, M., Anarim, E., Ciliz, M.K.: An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. Expert systems with Applications **29**(4), 713–722 (2005)

15. Fayyad, U.M., Irani, K.B.: Multi-interval discretization of continuous-valued attributes for classification learning. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 1022–1029. Morgan Kaufmann (1993)
16. Garcia, S., Grill, M., Stiborek, J., Zunino, A.: An empirical comparison of botnet detection methods. Computers & Security **45**, 100–123 (2014)
17. Gharibian, F., Ghorbani, A.A.: Comparative study of supervised machine learning techniques for intrusion detection. In: Annual Conference on Communication Networks and Services Research (CNSR'07), pp. 350–358. IEEE (2007)
18. Giacinto, G., Perdisci, R., Del Rio, M., Roli, F.: Intrusion detection in computer networks by a modular ensemble of one-class classifiers. Information Fusion **9**(1), 69–82 (2008)
19. Goseva-Popstojanova, K., Anastasovski, G., Pantev, R.: Using multiclass machine learning methods to classify malicious behaviors aimed at web systems. In: International Symposium on Software Reliability Engineering, pp. 81–90. IEEE (2012)
20. Guha, S., Rastogi, R., Shim, K.: Rock: A robust clustering algorithm for categorical attributes. In: International Conference on Data Engineering, pp. 512–521. IEEE (1999)
21. Hassani, M., Seidl, T.: Internal clustering evaluation of data streams. In: Trends and Applications in Knowledge Discovery and Data Mining, pp. 198–209. Springer (2015)
22. Hellemons, L., Hendriks, L., Hofstede, R., Sperotto, A., Sadre, R., Pras, A.: SSHCure: a flow-based SSH intrusion detection system. In: IFIP International Conference on Autonomous Infrastructure, Management and Security, pp. 86–97. Springer (2012)
23. John, W., Dusi, M., Claffy, K.C.: Estimating routing symmetry on single links by passive flow measurements. In: International Wireless Communications and Mobile Computing Conference, pp. 473–478. ACM (2010)
24. Jung, J., Paxson, V., Berger, A.W., Balakrishnan, H.: Fast portscan detection using sequential hypothesis testing. In: IEEE Symposium on Security and Privacy, pp. 211–225. IEEE (2004)
25. Kang, D.K., Fuller, D., Honavar, V.: Learning classifiers for misuse and anomaly detection using a bag of system calls representation. In: Annual IEEE SMC Information Assurance Workshop, pp. 118–125. IEEE (2005)
26. Kendall, K.: A database of computer attacks for the evaluation of intrusion detection systems. Tech. rep., DTIC Document (1999)
27. Landes, D., Otto, F., Schumann, S., Schlottke, F.: Identifying suspicious activities in company networks through data mining and visualization. In: P. Rausch, A.F. Sheta, A. Ayesh (eds.) Business Intelligence and Performance Management, pp. 75–90. Springer (2013)
28. Lee, C.H.: A hellinger-based discretization method for numeric attributes in classification learning. Knowledge-Based Systems **20**(4), 419–425 (2007)
29. Lin, J., Lin, H.: A density-based clustering over evolving heterogeneous data stream. In: ISECS International Colloquium on Computing, Communication, Control, and Management, vol. 4, pp. 275–277. IEEE (2009)
30. Liu, Q., Dong, G.: CPCQ: Contrast pattern based clustering quality index for categorical data. Pattern Recognition **45**(4), 1739–1748 (2012)
31. Małowidzki, M., Berezinski, P., Mazur, M.: Network intrusion detection: Half a kingdom for a good dataset. In: NATO STO SAS-139 Workshop, Portugal (2015)
32. Moore, A.W., Zuev, D.: Internet traffic classification using bayesian analysis techniques. In: ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 50–60. ACM, New York, USA (2005)
33. Murtagh, F., Contreras, P.: Algorithms for hierarchical clustering: an overview. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **2**(1), 86–97 (2012)
34. Najafabadi, M.M., Khoshgoftaar, T.M., Calvert, C., Kemp, C.: Detection of SSH brute force attacks using aggregated netflow data. In: International Conference on Machine Learning and Applications (ICMLA), pp. 283–288. IEEE (2015)
35. Najafabadi, M.M., Khoshgoftaar, T.M., Kemp, C., Seliya, N., Zuech, R.: Machine learning for detecting brute force attacks at the network level. In: International Conference on Bioinformatics and Bioengineering (BIBE), pp. 379–385. IEEE (2014)

36. Najafabadi, M.M., Khoshgoftaar, T.M., Napolitano, A., Wheelus, C.: Rudy attack: Detection at the network level and its important features. In: International Florida Artificial Intelligence Research Society Conference (FLAIRS), pp. 288–293 (2016)
37. Nguyen, T.T., Armitage, G.: A survey of techniques for internet traffic classification using machine learning. IEEE Communications Surveys & Tutorials **10**(4), 56–76 (2008)
38. Nychis, G., Sekar, V., Andersen, D.G., Kim, H., Zhang, H.: An empirical evaluation of entropy-based traffic anomaly detection. In: ACM SIGCOMM Conference on Internet measurement, pp. 151–156. ACM (2008)
39. Otto, F., Ring, M., Landes, D., Hotho, A.: Creation of specific flow-based training data sets for usage behaviour classification. In: European Conference on Cyber Warfare and Security (ECCWS), pp. 437–440 (2016)
40. Phaal, P., Panchen, S., McKee, N.: InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. RFC 3176 (2001)
41. Pramana, M.I.W., Purwanto, Y., Suratman, F.Y.: DDoS detection using modified k-means clustering with chain initialization over landmark window. In: International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC), pp. 7–11 (2015)
42. Rampure, V., Tiwari, A.: A rough set based feature selection on KDD CUP 99 data set. International Journal of Database Theory and Application **8**(1), 149–156 (2015)
43. Rehák, M., Pechoucek, M., Bartos, K., Grill, M., Celeda, P., Krmicek, V.: Camnep: An intrusion detection system for high-speed networks. Progress in Informatics **5**(5), 65–74 (2008)
44. Rehák, M., Pechoucek, M., Grill, M., Stiborek, J., Bartoš, K., Celeda, P.: Adaptive multiagent system for network traffic monitoring. IEEE Intelligent Systems **24**(3), 16–25 (2009)
45. Ring, M., Otto, F., Becker, M., Niebler, T., Landes, D., Hotho, A.: Condist: A context-driven categorical distance measure. In: European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 251–266. Springer (2015)
46. Rossi, D., Valenti, S.: Fine-grained traffic classification with netflow data. In: International wireless communications and mobile computing conference, pp. 479–483. ACM (2010)
47. Rostamipour, M., Sadeghiyan, B.: An architecture for host-based intrusion detection systems using fuzzy logic. Journal of Network and Information Security **2**(2) (2015)
48. Shah, V.M., Agarwal, A.: Reliable alert fusion of multiple intrusion detection systems. International Journal of Network Security **19**(2), 182–192 (2017)
49. Shearer, C.: The CRISP-DM model: the new blueprint for data mining. Journal of data warehousing **5**(4), 13–22 (2000)
50. Shiravi, A., Shiravi, H., Tavallaee, M., Ghorbani, A.A.: Toward developing a systematic approach to generate benchmark datasets for intrusion detection. Computers & Security **31**(3), 357–374 (2012)
51. Skoudis, E., Liston, T.: Counter Hack Reloaded: A Step-by-step Guide to Computer Attacks and Effective Defenses. Prentice Hall Series in Computer Networking and Distributed Systems. Prentice Hall Professional Technical Reference (2006)
52. Sommer, R., Paxson, V.: Outside the closed world: On using machine learning for network intrusion detection. In: IEEE Symposium on Security and Privacy, pp. 305–316. IEEE (2010)
53. Sperotto, A., Sadre, R., Van Vliet, F., Pras, A.: A labeled data set for flow-based intrusion detection. In: IP Operations and Management, pp. 39–50. Springer (2009)
54. Sridharan, A., Ye, T., Bhattacharyya, S.: Connectionless port scan detection on the backbone. In: IEEE International Performance Computing and Communications Conference, pp. 10–pp. IEEE (2006)
55. Staniford, S., Hoagland, J.A., McAlerney, J.M.: Practical automated detection of stealthy portscans. Journal of Computer Security **10**(1-2), 105–136 (2002)
56. Tan, P.N., Steinbach, M., Kumar, V.: Introduction to Data Mining, (First Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2005)
57. Tavallaee, M., Bagheri, E., Lu, W., Ghorbani, A.A.: A detailed analysis of the KDD CUP 99 data set. In: IEEE Symposium on Computational Intelligence for Security and Defense Applications, pp. 1–6 (2009)

58. Tran, Q.A., Jiang, F., Hu, J.: A real-time netflow-based intrusion detection system with improved BBNN and high-frequency field programmable gate arrays. In: International Conference on Trust, Security and Privacy in Computing and Communications, pp. 201–208. IEEE (2012)
59. Valenti, S., Rossi, D., Dainotti, A., Pescapè, A., Finamore, A., Mellia, M.: Reviewing traffic classification. In: Data Traffic Monitoring and Analysis, pp. 123–147. Springer (2013)
60. Wagner, C., François, J., Engel, T., et al.: Machine learning approach for ip-flow record anomaly detection. In: International Conference on Research in Networking, pp. 28–39. Springer (2011)
61. Weller-Fahy, D.J., Borghetti, B.J., Sodemann, A.A.: A survey of distance and similarity measures used within network intrusion anomaly detection. IEEE Communications Surveys & Tutorials **17**(1), 70–91 (2015)
62. Wheelus, C., Khoshgoftaar, T.M., Zuech, R., Najafabadi, M.M.: A session based approach for aggregating network traffic data - the santa dataset. In: International Conference on Bioinformatics and Bioengineering (BIBE), pp. 369–378. IEEE (2014)
63. Winter, P., Hermann, E., Zeilinger, M.: Inductive intrusion detection in flow-based network data using one-class support vector machines. In: International Conference on New Technologies, Mobility and Security (NTMS), pp. 1–5. IEEE (2011)
64. Yang, C., Zhou, J.: Hclustream: A novel approach for clustering evolving heterogeneous data stream. In: International Conference on Data Mining-Workshops (ICDMW'06), pp. 682–688. IEEE (2006)
65. Zander, S., Nguyen, T., Armitage, G.: Automated traffic classification and application identification using machine learning. In: The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05) l, pp. 250–257. IEEE (2005)
66. Zuech, R., Khoshgoftaar, T.M., Seliya, N., Najafabadi, M.M., Kemp, C.: A new intrusion detection benchmarking system. In: International Florida Artificial Intelligence Research Society Conference (FLAIRS), pp. 252–256. AAAI Press (2015)