Detecting Anomalous Network Traffic in Organizational Private Networks

Risto Vaarandi

This paper has been accepted for publication at the 2013 IEEE CogSIMA Conference, and the final version of the paper is included in *Proceedings of the 2013 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support* (DOI: 10.1109/CogSIMA.2013.6523859)

Detecting Anomalous Network Traffic in Organizational Private Networks

Risto Vaarandi, *NATO Cooperative Cyber Defence Centre of Excellence*

***Abstract*—During the last decade, network monitoring and intrusion detection have become essential techniques of cyber security. Nowadays, many institutions are using advanced solutions for detecting malicious network traffic, discovering network anomalies, and preventing cyber attacks. However, most research in this area has not been conducted specifically for organizational private networks, and their special properties have not been considered. In this paper, we first present a study of traffic patterns in a corporate private network, and then propose two novel algorithms for detecting anomalous network traffic and node behavior in such networks.**

***Index Terms*—cyber security, network anomaly detection, network monitoring, network forensics**

I. Introduction

DURING the last decade, network monitoring and intrusion detection have become essential techniques of cyber security. A number of vendors are offering advanced solutions for intrusion detection and prevention, for network anomaly detection, for network alarm correlation, and for other security monitoring purposes. Many larger institutions are using a dedicated intrusion detection system (IDS) for discovering cyber attacks and other malicious or abnormal traffic.

Today, most industrial-grade IDS solutions are employing signature-based analysis for identifying unwanted network activity (the signatures which describe bad traffic are written by human experts). Unfortunately, such IDSs are unable to discover previously unknown malicious network activity, such as zero-day attacks. For this reason, a number of network monitoring methods have been proposed which do not rely on an extensive human-written signature database, but rather use various algorithms for classifying network traffic as normal or anomalous. One widely used industrial protocol for collecting traffic information is Cisco NetFlow [1]. With NetFlow, a router, switch or dedicated network probe keeps track of network packets it has forwarded or observed. If a NetFlow-enabled device sees a flow of packets going from some source to some destination, it creates a memory-based record for this flow which is identified by source IP, source port, destination IP, destination port, transport protocol ID, and few other parameters (note that for the return traffic from destination to source, another flow record is created). The data stored to a flow record include the start and end timestamps of the flow in milliseconds, numbers of observed packets and bytes, and the union of all observed TCP flags. The device reports a flow record to the collector and drops it from memory when the corresponding network connection is terminated, or when the activity or inactivity timer expires for the flow. In order to increase processing speed for backbone networks, packet sampling may be configured – flow records are kept in memory only for a fraction of packets (e.g., one packet out of 10,000 is considered).

Due to the lightweight and efficient nature of NetFlow protocol, a number of widely used NetFlow-based network monitoring solutions have been developed, e.g., NfSen [2], Fprobe [3], Flow-tools [4], SiLK [5] and Nfsight [6]. Also, a number of algorithms have been proposed in recent papers for detecting malicious and abnormal network activity from NetFlow data [7]–[16]. Unfortunately, most of the recent research has not considered network anomaly detection for organizational private networks specifically, although such environments have several unique properties. First, in these networks clients can't exchange arbitrary data with potentially unlimited number of nodes in the Internet. Second, corporate policies often prohibit many activities which are common in public networks, such as the use of P2P protocols for exchanging large amounts of data. Therefore, in organizational private networks distinct network traffic patterns can be observed which deserve closer study. Also, private networks have specific monitoring requirements – apart from detecting well-known malicious traffic with IDS or other means, it is often highly desirable to track connections with seemingly harmless content between clients and services, in order to identify illegal actions by insiders (e.g., unauthorized access to confidential data), malware activity (e.g., zero-day attacks against services), illegitimate services, etc. Fortunately, in private networks NetFlow data can often be collected without sampling which allows for more precise security monitoring.

The first contribution of this paper is a study of typical network usage patterns in a corporate private network. We are not aware of any previous work done for similar environments. We will then present two novel anomaly detection algorithms which rely on these findings. The first algorithm addresses the problem of monitoring seemingly legitimate connections between clients and services in private networks, and discovers unusual TCP and UDP packet flows. The second algorithm applies a clustering method for detecting nodes with unexpected changes in their service usage patterns.

The remainder of this paper is organized as follows: section II reviews related work, section III presents the study of typical network traffic patterns in an organizational private network, section IV proposes algorithms for anomaly detection in organizational networks, section V describes the results of our experiments, and section VI discusses further work.

II. Related Work

In this section, we will review some of the recent related work in the field of NetFlow-based anomaly detection.

Wagner and Plattner have suggested an entropy-based worm and anomaly detection method which measures entropy contents of some network traffic features (IP addresses and port numbers) [7]. If changes in entropy contents are observed, the method raises an alarm. The authors have demonstrated that the method is able to detect worm outbreaks and massive scanning activities in near real time. Ranjan et al. have suggested another entropy-based worm detection algorithm which measures entropy ratios for traffic feature pairs, and issues an alarm on sudden changes [8].

Kind, Stoecklin and Dimitropoulos have proposed a histogram-based anomaly detection approach [9]. With this approach, histogram-based baselines are constructed from training data for some essential network traffic features (such as source IP address, destination IP address, source port number, etc.). If a deviation from a baseline is observed during network monitoring for some traffic feature, an alarm will be raised. Brauckhoff, Dimitropoulos, Wagner and Salamatian have augmented a histogram-based anomaly detection approach with association rule mining, in order to identify NetFlow records representing anomalous network traffic [10].

Weigert, Hiltunen and Fetzer have proposed a graph-based method for communities, where community members are institutions of the same type [11]. The method maintains graphs for IP addresses which communicate with members, and is able to identify similar attacks against several members. In their paper [12], Bartoš, Grill, Krmíček, Řehák and Čeleda describe a system which employs fast NetFlow probes. Flows collected from probes are processed by several agents with different anomaly detection algorithms. The results from agents are aggregated, in order to reduce the number of false positives. Wagner, François, State and Engel have proposed an approach based on support vector machines, in order to classify flow records [13].

Münz, Li and Carle have suggested a method which applies k-means clustering algorithm for NetFlow training data [14]. Detected cluster centroids are assumed to describe normal network traffic, and substantially different traffic can be classified as anomalous. Paredes-Oliva, Castell-Uroz, Barlet-Ros, Dimitropoulos and Solé-Pareta have proposed a method which first discovers frequent traffic patterns with a frequent itemset mining algorithm, and then applies decision trees for finding anomalous patterns [15]. The author of this paper has suggested another frequent itemset mining approach for near-real-time identification of strong anomalous network traffic patterns [16].

III. A Study of Traffic Patterns In an Organizational Private Network

In this section, we will present a study of traffic patterns in an organizational private network. For our study, we collected NetFlow data from a private backbone network of a large financial institution. The data set was collected during 150 days without packet sampling and contains 1,093,911,511 flow records. During the collection period, data for 1,780 nodes were recorded, with 436 nodes being workstations. Other nodes include several hundreds of servers, printers, network switches, but also a number of specialized nodes like ATMs, UPS devices, environment monitoring modules, etc.

We define the *service* as a tuple (IP address, port number, transport protocol ID), while the *client* of a service is defined as an IP address which is employed for communicating with the service. In order to distinguish services and clients from each other and analyze their behavioral patterns, we preprocessed the collected data set using heuristics described in subsection IVa. We found that during 150 days, 2,924 services were used by 1,609 clients. When we analyzed the number of clients the services have on a daily basis, we discovered that most services are used by only few clients per day. We calculated an average number of clients per day for each service, and divided services into non-overlapping groups by the calculated figure. Table I summarizes our findings.

According to Table I, less than 10% of services have an average of 5 or more clients per day, while 76.74% of services

TABLE I
Service Groups By The Average Number Of Clients Per Day

Service groups by the average number of clients per day[a]	Number of services in a group	Average number of days of service usage[b]
Services with an average of 100 or more clients per day	28 (≈0.96%)	137.21
Services with an average of at least 25 and less than 100 clients per day	87 (≈2.98%)	84.32
Services with an average of at least 10 and less than 25 clients per day	96 (≈3.28%)	48.11
Services with an average of at least 5 and less than 10 clients per day	65 (≈2.22%)	99.94
Services with an average of at least 2 and less than 5 clients per day	404 (≈13.82%)	39.72
Services with an average of more than 1 and less than 2 clients per day	648 (≈22.16%)	89.04
Services with an average of at most 1 client per day	1,596 (≈54.58%)	25.84

[a] *Clients per day* average for each service was calculated for days the service was active (e.g., if a service was used during 138 days, the average was found for 138 days)

[b] Calculated for services in a given group.

have an average of less than 2 clients per day. When we investigated services with few clients more closely, we discovered that a large part of them are specialized services which are not used by humans. For example, we found that about 20% of all services are SNMP agents which are accessed by few dedicated network management nodes. In addition to SNMP, a number of other protocols are extensively employed in the private network for system management, for communications between infrastructure components, etc. In all such cases the services typically have only 1-2 clients with well-known IP addresses.

In order to estimate how many network flows are associated with services which are consistently consumed by larger number of clients, we investigated the services which are used by at least 100 clients per day during at least 75 days (half of the timeframe covered by our NetFlow data set). We identified 28 such services, including corporate web proxies, mail servers, and name servers. We discovered that 417,771,347 flow records (38.19% of records in the data set) reflect network traffic from/to them. When we inspected the services which are used by at least 25 clients per day during at least 75 days, we found 67 services with 691,655,760 flow records (63.23% of records in the data set) associated with them.

To summarize our findings, majority of the services have very few clients (often with well-known IP addresses), while only a small fraction of services are consistently accessed by many clients. Nevertheless, a large part of collected NetFlow data describes the network traffic from/to such few popular services.

TABLE II
CLIENT GROUPS BY THE AVERAGE NUMBER OF USED SERVICES PER DAY

Client groups by the average number of used services per day[a]	Number of clients in a group	Average number of days of client activity[b]
Clients with an average of 100 or more used services per day	8 (≈0.50%)	107.25
Clients with an average of at least 50 and less than 100 used services per day	11 (≈0.68%)	91.82
Clients with an average of at least 25 and less than 50 used services per day	118 (≈7.33%)	119.96
Clients with an average of at least 10 and less than 25 used services per day	306 (≈19.02%)	138.11
Clients with an average of at least 5 and less than 10 used services per day	112 (≈6.96%)	128.25
Clients with an average of at least 2 and less than 5 used services per day	198 (≈12.31%)	99.77
Clients with an average of more than 1 and less than 2 used services per day	239 (≈14.85%)	99.26
Clients with an average of at most 1 used service per day	625 (≈38.84%)	75.70

[a]*Services per day* average for each client was calculated for days the client was active (e.g., if a client was active during 25 days, the average was found for 25 days)

[b]Calculated for clients in a given group.

We also analyzed the behavior of 1,609 client nodes. We found that 944 clients (58.67% of clients) consumed services during at least 120 days (80% of the timeframe covered by the data set), while 1,193 clients (74.15% of clients) consumed services during at least 30 days (20% of the timeframe covered by the data set). 83 clients were active during one day only. We also calculated an average number of used services per day for each client, and employed this figure for dividing clients into non-overlapping groups. Our findings are presented in Table II.

According to the table, almost 99% of clients are accessing an average of less than 50 services per day, while for 91.49% of clients the average number of consumed services remains below 25 per day. Furthermore, 66% of clients are contacting an average of less than 5 services per day. When we conducted a separate analysis for workstation client nodes, we found that the average number of used services per day ranges from 1 to 53, with the overall average 18.26 across 436 workstations. In other words, large majority of clients are consuming either few or moderate number of services each day. Moreover, this network behavior is also common for workstations, although human users are often not following the same service usage pattern on a daily basis. Other researchers have observed a similar phenomenon for university campus workstations [17].

Finally, we tried to estimate how often clients are accessing services they have used before. For this purpose, we extracted the list of all consumed services for each client from the entire 150 day NetFlow data set. We found that only 63 clients contacted more than 100 services, 336 clients contacted 50-99 services, 150 clients contacted 25-49 services, 88 clients contacted 10-24 services, and 972 clients contacted less than 10 services. We also found that 692 clients contacted at least 75% of services in their list during 75 days or more. Also, 922 and 1,330 clients used at least 50% and 25% of their services, respectively, during 75 days or more. These figures suggest that client nodes are using the same services over longer periods of time, and if a service has been accessed before, it is likely to be accessed in the future.

Another interesting phenomenon is that clients often access services which other clients have recently used, and seldom establish a connection to a rarely used service. When we investigated service sharing during 150 days, we discovered that 1,479-1,559 (91.92%-96.89%) clients contacted only these services which were also used by at least one other client during the same day. We have observed similar service sharing patterns during our previous research – a 2 week experiment revealed that in 1 hour time windows, 68-94% of workstations only interacted with services used by at least 4 other nodes within the same 1 hour window [18].

IV. Anomaly Detection For Organizational Private Networks

In this section, we will present two algorithms for anomaly detection in organizational private networks. The algorithms leverage typical traffic patterns and node behavior discussed in the previous section. Both algorithms employ a service detection method which discovers TCP and UDP based network services from NetFlow data sets of recent past (e.g., data from last 30 days). This information is used for creating behavior profiles for each client. Proposed anomaly detection algorithms use these profiles for near-real-time detection of anomalous network flows, and for daily detection of node behavior changes through data clustering. The following subsections provide a detailed discussion of our algorithms.

A. Service Detection From NetFlow Data

Service detection from NetFlow data sets is a non-trivial task, because NetFlow records do not contain information about the initiator of a network connection (see also [6] for a related discussion). For example, although each NetFlow record for TCP traffic contains a separate field for the union of TCP flags observed during the flow, the successful negotiation of a TCP connection sets SYN and ACK flags for both records describing the directions of the connection. Since throughout the rest of the connection both sides can use the same flag combinations in packet headers, the TCP flags field of the NetFlow record can't be employed for distinguishing the server from the client. Also, connectionless transport protocols like UDP do not involve any specific negotiations between the client and server before the actual data transfer. Each NetFlow record includes the timestamp of the start of the flow (measured in milliseconds), and it seems tempting to assume that when two records for a new connection are created, the flow record with an earlier timestamp reflects the traffic from client to server. However, quite often the first packet from the client and the server response are observed within the same millisecond. Furthermore, during our experiments we have seen buggy NetFlow implementations which sometimes set flow timestamps to incorrect values. For these reasons, comparing the timestamps of two records for a given connection will not always reliably distinguish a client from a server. Also, in the case of some UDP based protocols data travel from client to server only (e.g., SNMP traps and BSD *syslog* messages).

In order to address the problem of network service discovery from NetFlow data, we are using the following approximate method for finding TCP and UDP based services. During data set processing, flow records for UDP flows and for TCP flows with SYN and ACK flags are considered. First a pass over the data set is made, and a list of service candidates is created with several heuristic steps which are applied in the given order (some of these heuristics are also used by the Nfsight network monitoring tool [6]):

1. If a flow record reflects a communication between host ip_1 port p_1 and host ip_2 port p_2 over transport protocol *proto*, with p_1 being an unprivileged and p_2 a privileged port (i.e., $p_1 \geq 1024$ and $p_2 < 1024$), consider (ip_2, p_2, *proto*) as a candidate.
2. If a flow record reflects a communication between host ip_1 port p_1 and host ip_2 port p_2 over transport protocol *proto*, with p_1 being an unprivileged and p_2 a predefined unprivileged service port (during our experiments, we used 5 predefined TCP port numbers), consider (ip_2, p_2, *proto*) as a candidate.
3. If a flow record with timestamp t reflects a communication from host ip_1 port p_1 to host ip_2 port p_2 over transport protocol *proto*, and during past processing we have observed a record with an earlier timestamp t' ($t - d \leq t' < t$) for communication from host ip_2 port p_2 to host ip_1 port p_1 over *proto*, consider (ip_1, p_1, *proto*) as a candidate (during the experiments, we have set d to 120 seconds).
4. If a flow record with timestamp t reflects a communication from host ip_1 port p_1 to host ip_2 port p_2 over UDP, and during past processing we have not observed a record with an earlier timestamp t' ($t - d \leq t' < t$) for communication from host ip_2 port p_2 to host ip_1 port p_1 over UDP, consider (ip_2, p_2, *UDP*) as a candidate.

During the data pass, the number of flow records and the list of distinct clients associated with each candidate service are retained in memory (according to section III, most services have few clients). After this, the second data pass is made, and several other heuristics are applied to candidate services, in order to remove false positives and find nodes which run dynamic services. First, if during the data pass a record is observed where one endpoint is a service candidate with many associated flows and clients, and the other endpoint is a candidate with few associated flows, the latter will be dropped from the candidate list, since it is very likely to represent the client side of the connection. Second, if a well-known portmapper service port (e.g., 135/tcp) of some host has many associated flows and clients, and there are also many flow records describing traffic between unprivileged ports of this host and other nodes, consider unprivileged ports of the host as service ports. However, during our experiments we found only few such hosts from daily NetFlow data sets (1-14 hosts per day). After the second data pass, the refined list of service candidates will be reported as the list of discovered services. The service list will serve as input for anomaly detection algorithms described in the following subsections.

B. Near-Real-Time Detection of Anomalous Network Flows

For detecting anomalous network flows, we have developed a method which builds service usage profiles for each client from past NetFlow data sets, and then employs these profiles for distinguishing anomalous network activity from normal traffic in near-real-time.

```
Input: n – mine client profiles and widely used services from NetFlow
           data sets of last n days (D_1, …, D_n)
       minusage – relative threshold for service usage days
       mindays – absolute threshold for service usage days
       minclnt – threshold for number of clients of a widely used service
       mindays2 – threshold for activity days of a widely used service

Output:  {P_c1, …, P_cm} – profiles for all detected clients c_1, …, c_m
         W – the set of widely used services

foreach i ∈ (1, ..., n) do
    discover the list of services S_i from D_i, using the method from
    subsection IVa;

    C_i := ∅;
    foreach s ∈ S_i do L_s,i := ∅; done

    loop1:
    foreach r ∈ D_i do
      if (r_proto = UDP or
          (r_proto = TCP and {SYN, ACK} ⊆ r_flags)) then
        if ((r_dstip, r_dstport, r_proto) ∈ S_i) then
          c := r_srcip;  s := (r_dstip, r_dstport, r_proto);
        elsif ((r_srcip, r_srcport, r_proto) ∈ S_i) then
          c := r_dstip;  s := (r_srcip, r_srcport, r_proto);
        else continue to the next step of loop1; fi
        L_s,i := L_s,i ∪ {c};
        if (c ∉ C_i) then  C_i := C_i ∪ {c};  K_c,i := {s};
        else K_c,i := K_c,i ∪ {s}; fi
      fi
    done
done

foreach c ∈ ∪_{i=1}^{n} C_i do  P_c := ∅; M_c := ∅; done

foreach i ∈ (1, ..., n) do
  foreach c ∈ C_i do
    foreach s ∈ K_c,i do
      M_c := M_c ∪ {s};
      if (uninitialized(days_c,s)) then days_c,s := 0; fi
      days_c,s := days_c,s + 1;
    done
    if (uninitialized(days2_c)) then days2_c := 0; fi
    days2_c := days2_c + 1;
  done
done

foreach c ∈ ∪_{i=1}^{n} C_i do
  foreach s ∈ M_c do
    if (days_c,s / days2_c  ≥ minusage
        and days_c,s ≥ mindays) then P_c := P_c ∪ {s}; fi
  done
done

W := ∅;
foreach i ∈ (1, ..., n) do
  foreach s ∈ S_i do
    if (uninitialized(days3_s)) then days3_s := 0; fi
    if (|L_s,i | ≥ minclnt) then days3_s := days3_s + 1; fi
  done
done
foreach s ∈ ∪_{i=1}^{n} S_i do
  if (days3_s ≥ mindays2) then W := W ∪ {s}; fi
done
```

Fig. 1. An algorithm for mining client profiles and widely used services from NetFlow data sets

The method relies on the observation that if a client has used a service frequently in the past, the client is likely to use this service in the future (see section III). Our method mines service usage profiles from daily NetFlow data sets D_1, ..., D_n of the last n days. We assume that each D_i is a set of NetFlow records $D_i = \{r_1, ..., r_{ki}\}$, where each record has a number of attributes, including *proto*, *srcip*, *srcport*, *dstip*, *dstport*, and *flags* (the union of TCP flags observed during the lifetime of the flow). The mining is conducted once in every 24 hours, in order to update the profiles and adjust to environment changes. The profile mining algorithm is outlined in Figure 1.

Informally, after clients and services have been detected from D_1, ..., D_n, the profile for each client will be set up which is a list of frequently used services by this client. A service will be included in the profile if the client has accessed it at least during *(100*minusage)%* of days of client activity (e.g., if the client has been active during 20 days and *minusage* is 0.5, services accessed during 10 days or more are included in the profile). In addition to this relative threshold, the algorithm employs an absolute threshold *mindays* which is convenient for less active clients (e.g., if *mindays* is 5 and the client has been active for 5 days, only services used during all days can be included in the profile).

However, with this approach not all clients would have profiles – for example, clients which have been active for less than *mindays* days, and previously unseen clients which appear during near-real-time flow monitoring. Also, during our experiments we have observed that some clients access popular services with varying frequency which might leave these services out from their profiles, and thus trigger false positives. In order to address these issues, the algorithm presented in Figure 1 also builds the set of widely used services W which is employed as a default client profile, and also for filtering out false positives. A service is included in W if it is used during at least *mindays2* days by at least *minclnt* distinct clients per day. The employment of W is motivated by typical traffic patterns described in section III – clients mostly communicate with services also used by other nodes, and access to a popular service is thus likely a normal activity.

For faster execution of the mining algorithm, the sets W, P_{ci} (client profiles), $L_{s,i}$ (clients of given services for given days), $K_{c,i}$ (services for given clients for given days), and M_c (services for given clients for all days) have to be kept in memory. Fortunately, as discussed in section III, for most clients the number of consumed services remains either moderate or small. Furthermore, only few services have larger number of clients per day. Therefore, the aforementioned sets have a moderate size and fit into memory.

After client profiles and the set of widely used services have been discovered, this knowledge is employed for classifying flow records arriving from NetFlow-enabled network devices (see Figure 2). Since the device might report a flow record with a slight delay (typically at most few minutes after the network traffic was observed), the anomaly detection is accomplished in near-real-time. The algorithm classifies

records for UDP flows and for TCP flows with SYN and ACK flags (such TCP flows represent successfully established new TCP connections). Other flow records are regarded as out of scope. If the flow record describes communication with an unusual service for a client with the profile, or if neither peer is recognized as a widely used service, the algorithm reports the flow as anomalous. This allows for detecting unusual behavior of clients, but also the appearance of new and potentially illegal services. In order to avoid large numbers of false positives when a new widely used legitimate service is introduced, the administrators can add services to set *W* through an external white-list (however, during our experiments we have not used this measure). Since sets P_{c1}, …, P_{cm} and *W* mostly contain moderate number of elements, the algorithm can store them in memory for faster execution.

```
Input:   r – NetFlow record;
         {P_c1, …, P_cm} – client profiles;
         W – the set of widely used services

Output:  1 if flow record r is anomalous;
         0 if flow record r is normal;
         -1 if flow record is out of scope of the algorithm

if (r_proto ≠ UDP or r_proto ≠ TCP or
   (r_proto = TCP and {SYN, ACK} ⊄ r_flags)) then return -1; fi
k1 := r_srcip;
k2 := r_dstip;
if (exists(P_k1) and (r_dstip, r_dstport, r_proto) ∈ P_k1) then return 0; fi
if (exists(P_k2) and (r_srcip, r_srcport, r_proto) ∈ P_k2) then return 0; fi
if ((r_dstip, r_dstport, r_proto) ∈ W) then return 0; fi
if ((r_srcip, r_srcport, r_proto) ∈ W) then return 0; fi
return 1;
```

Fig. 2. Anomaly detection algorithm for NetFlow records.

Finally, since the algorithm focuses on anomaly detection for seemingly legitimate traffic, it does not monitor for well-known malicious network activities which leave specific traces in NetFlow data (e.g., TCP SYN-FIN scans). Nevertheless, such malicious traffic can be easily detected by writing simple filtering conditions for flow records.

C. Monitoring Node Behavior Changes

The method described in the previous subsection is capable of identifying anomalous data exchanges with services which are neither frequently accessed by the client nor commonly used by many other nodes. However, for some clients this monitoring might not be sufficient enough. For example, if the node is not a workstation but is rather running dedicated software which communicates with specific services only, deviations from normal service usage patterns should be monitored more closely. Unfortunately, it is quite hard to establish individual monitoring thresholds for each such node separately, since their number might be too large. However, quite often dedicated nodes form larger groups, with group members having (almost) identical network behavior. Therefore, the members can be monitored by first identifying a common behavioral pattern for the whole group, and then using this as a baseline for each member. In this subsection, we propose a data clustering algorithm for the detection of such groups, and a method for using detected clusters for finding behavior changes of individual nodes. The clustering algorithm is outlined in Figure 3.

```
Input:  D – NetFlow data set;
        m – minimum number of clients for a frequent service;
        e – distance threshold for the node to be included in a cluster

Output: Descr – the set of cluster descriptions;
        Clusters – the set of clusters

S := { services detected from D with method from subsection IVa };
C := { clients of services S detected from D };
foreach c ∈ C do K_c := { services used by client c }; done
foreach s ∈ S do L_s : = { clients which used service s }; done

S_F := { s | s ∈ S, |L_s| ≥ m };
Cand := ∅;

foreach c ∈ C do
   T := { s | s ∈ K_c, s ∈ S_F };
   if (|T| > 0) then
      if (uninitialized(counter_T)) then counter_T := 0; fi
      Cand := Cand ∪ { T }; counter_T := counter_T + 1;
   fi
done

Cand := { T | T ∈ Cand, counter_T ≥ m}
foreach T ∈ Cand do Members_T := ∅; done

foreach c ∈ C do
   maxd := 0;
   foreach T ∈ Cand do
      if (T ⊆ K_c and |T|/|K_c| > maxd and |T|/|K_c| ≥ e) then
         maxd := |T|/|K_c|; R := T; fi
   done
   if (maxd > 0) then Members_R := Members_R ∪ {c}; fi
done

Descr := { T | T ∈ Cand, |Members_T| > 0 }
Members_∅ := { c | c ∈ C, ∀T ∈ Descr c ∉ Members_T }
Descr := Descr ∪ { ∅ }
Clusters := { Members_T | T ∈ Descr }
```

Fig. 3. Node clustering algorithm for NetFlow data.

The algorithm first identifies services which are used by at least *m* clients (so called frequent services), and then finds sets of frequent services which are used by at least *m* clients. Each set is regarded as a description of a cluster candidate. After that, the closest candidate description is found for each client node, and the node is regarded as a member of this cluster candidate. The distance *d(c, A)* between the node *c* and the candidate description *A* is found as follows – if *B* is the set of all services node *c* has used, then $d(c, A) = |A|/|B|$ if $A \subseteq B$, and $d(c, A) = 0$ otherwise (note that $0 \le d(c, A) \le 1$). The algorithm also uses the distance threshold *e* during clustering – if the distance from the node to any candidate description is smaller than *e*, the node is assigned to the cluster of outliers. For the cluster of outliers, we define its description as an empty set of services. Finally, cluster candidates without any members are dropped, and remaining candidates are selected as clusters.

Due to the nature of the algorithm, nodes with similar

service usage patterns are assigned to the same cluster. Also, each cluster description naturally describes the common service usage pattern for all cluster members. The use of threshold *m* ensures that clusters are selected from candidates representing service usage patterns appearing frequently enough in the data set. With higher values for distance threshold *e*, only these nodes which match the common service pattern very closely are included in the cluster. If *Clusters* is a set of clusters, $Clusters_C$ denotes the cluster with the description *C*. In order to measure, how well the cluster description *C* captures the network behavior of all cluster members $m_1, \ldots, m_k$ ($m_i \in Clusters_C$, $|Clusters_C| = k$), we define the cluster significance $S(Clusters_C)$ as follows:

$S(Clusters_C) = \sum_{i=1}^{k} d(m_i, C) / k$.

In order to use this clustering algorithm for monitoring node behavior, we run the algorithm at the end of each day, and compare the clustering for the current day with clustering results for previous *n* days. Our method first checks if a given node has been a member of clusters with high significance during previous *n* days, and if the node's service usage patterns have been close enough to cluster descriptions. Also, the method verifies that the number of distinct cluster descriptions for previous *n* days is modest, and the average number of cluster members remains above the given threshold. Nodes not fulfilling these criteria are dropped from further consideration, since their network behavior has been either constantly changing or not similar enough to larger group(s) of other nodes.

For every remaining node the method verifies that its cluster description for the current day appears within the list of cluster descriptions for previous *n* days. In other words, the algorithm checks if the current service usage pattern of the node has already been observed in the past. If this is not the case and the cluster represents a new service usage pattern, the method checks if most other nodes in the cluster have consistently appeared in same clusters with the given node in the past (in other words, the behavior of the whole group of similar nodes has changed). If either of these conditions holds, the node behavior is regarded as normal, and abnormal otherwise.

Formally, for each client node *c* that is part of the clustering for the current day, the method applies the following steps (*Clusters* denotes the current clustering, and $Clusters^1, \ldots, Clusters^n$ the clusterings for previous *n* days). The method is called *NodeBehavior* and takes *n*, *minavgD*, *minavgM*, *minavgS*, *maxclust*, *thresh* and *thresh2* as input parameters:

1. Find the cluster description *C* for node *c* in *Clusters* (i.e., $c \in Clusters_C$).
2. If node *c* has not been active during the previous *n* days, terminate the processing and return "out of scope". If node *c* was active for *m* days $i^1,\ldots,i^m$ ($1 \le m \le n$), find cluster descriptions $C_1, \ldots, C_m$ and clusters $Clusters^{i^1}{}_{C_1}, \ldots, Clusters^{i^m}{}_{C_m}$ for all *m* days for *c* ($c \in Clusters^{i^j}{}_{C_j}$, $1 \le j \le m$).
3. Find the average cluster significance *AvgS(c)* for node *c*: $AvgS(c) := \sum_{j=1}^{m} S(Clusters^{i^j}{}_{C_j}) / m$. Also, find the average distance *AvgD(c)* from *c* to $C_1, \ldots, C_m$: $AvgD(c) := \sum_{j=1}^{m} d(c, C_j) / m$. Finally, find the average number of cluster members *AvgM(c)* for node *c*: $AvgM(c) := \sum_{j=1}^{m} |Clusters^{i^j}{}_{C_j}| / m$. If *AvgS(c)* is smaller than *minavgS*, or *AvgD(c)* is smaller than *minavgD*, or *AvgM(c)* is smaller than *minavgM*, terminate the processing and return "out of scope".
4. Find the number of unique elements *u* in the list $C_1, \ldots, C_m$: $u := |\cup_{j=1}^{m} \{ C_j \}|$. If *u* is larger than *maxclust*, terminate the processing and return "out of scope".
5. If the cluster description *C* for the current day appears within the list $C_1, \ldots, C_m$, terminate the processing and return "normal".
6. If the cluster description *C* for the current day does not belong to the list $C_1, \ldots, C_m$, extract other nodes from cluster $Clusters_C$. If at least *(100*thresh)%* of these nodes appear in at least *(100*thresh2)%* of clusters $Clusters^{i^1}{}_{C1}, \ldots, Clusters^{i^m}{}_{C_m}$, terminate the processing and return "normal"; otherwise return "anomalous".

V. EXPERIMENTS

In order to evaluate the performance of algorithms proposed in the previous section, we have run prototype implementations of them in a private backbone network of a large financial institution (the data for traffic pattern study presented in section III were obtained from the same environment). The experiment lasted for 123 days from June 2012 to October 2012, and involved the processing of 801,477,584 NetFlow records.

For near-real-time anomaly detection of network flows, we used the following input parameter values for daily mining of client profiles $P_{c1}, \ldots, P_{cm}$ and the set of widely used services *W* (see Figure 1): *n*=30, *minusage*=0.25, *mindays*=5, *minclnt*=50, *mindays2*=15. In other words, NetFlow data sets of the last 30 days were mined, and a service was included in the client profile if it was used during at least 25% of days of client activity, but no less than 5 days. Services consumed by at least 50 clients during at least 15 days were regarded as widely used services. The flow anomaly detector (see Figure 2) was implemented in Perl and was running as a daemon on a Linux server. As discussed in subsection IVb, the memory requirements of the detector are modest, and during our experiments its memory consumption remained under 10MB. During the experiment of 123 days, the anomaly detector classified 679,402,649 flow records as normal and 28,976 records as anomalous (remaining records were regarded as being out of scope). However, almost half of the anomalous records (13,744) were detected during a single day, while during other two days 1,805 and 1,752 flows were tagged as anomalous. For the remaining 120 days the number of anomalous flows ranged from 2 to 521 (an average of 97.29

flows per day). Since typically few hundred anomalous flows are reported per day, the human administrator can easily inspect them. The anomalous flows reported during our experiment represent a variety of unusual network traffic. For example, we detected abnormal packets triggered by faults in device configurations (e.g., SNMP traps sent to wrong destinations), anomalous end user actions (e.g., print jobs submitted from a workstation to a wrong printer in another city), but also unusual system management activity (e.g., SSH connections to rarely managed devices).

We also implemented the *NodeBehavior* method for automated discovery of node groups with common service usage patterns, and finding deviations from common node group behavior. For the *NodeBehavior* method, input parameter values were n=30, $minavgD$=0.9, $minavgM$=5, $minavgS$=0.9, $maxclust$=10, $thresh$=0.9, and $thresh2$=0.9. For daily node clustering (see Figure 3), input parameters were set as m=10 and e=0.75. During the whole experiment, 1,557 clients were observed, and the algorithm was able to report normal or anomalous behavior for 485-603 nodes per day (for remaining nodes, "out of scope" was returned). The number of detected clusters per day ranged from 57 to 72, and the number of anomalous nodes from 0 to 21 (an average of 2.08 nodes per day). We discovered that in many cases node behavior was regarded as anomalous because of an access to rarely used legitimate service (e.g., an SNMP trap alert was sent to a network management server, while other similar nodes did not issue any alerts during the same day). However, in a number of cases node behavior was regarded anomalous for not consuming a service used by other nodes in the same group. For example, this helped to discover nodes which had stopped sending *syslog* messages to a central collection point, or had stopped synchronizing their clocks with a central NTP server. In other words, the clustering based anomaly detection is quite useful for finding unusual inactivity of a node.

VI. Future Work

In this paper, we have presented a study of traffic patterns in an organizational private network, and have presented two anomaly detection methods for identifying unusual TCP and UDP traffic. The methods suggested in this paper can be extended in several ways. First, other clustering algorithms with different distance functions could be harnessed for finding nodes with similar service usage patterns. Second, if dynamic addressing is used for workstations, IP addresses can't be reliably associated with end users who are often following distinct network usage patterns. For tackling this issue, a mapping between successfully authenticated users and workstation IP addresses could be stored in a database, and this information could be employed for identifying workstations by username, not by IP address. Third, while algorithms proposed in this paper focus on client network behavior, we plan to extend this work with service anomaly detection (e.g., finding illegitimate clients). Finally, we would like to investigate the use of other flow attributes (e.g., flow duration) for anomaly detection purposes.

Acknowledgment

The author wishes to thank SEB Estonia for supporting this work, and, in particular, Mr. Kaido Raiend, Mr. Ants Leitmäe, Mr. Andrus Tamm, Dr. Paul Leis, and Mr. Ain Rasva.

References

[1] http://www.cisco.com/go/netflow
[2] http://nfsen.sourceforge.net
[3] http://fprobe.sourceforge.net
[4] M. Fullmer and S. Romig, "The OSU Flow-tools Package and Cisco NetFlow Logs," *Proceedings of 14th USENIX Large Installation Systems Administration Conference*, 2000, pp. 291-303.
[5] C. Gates, M. Collins, M. Duggan, A. Kompanek, M. Thomas, "More NetFlow Tools: For Performance and Security," *Proceedings of 18th USENIX Large Installation Systems Administration Conference*, 2004, pp. 121-132.
[6] R. Berthier, M. Cukier, M. Hiltunen, D. Kormann, G. Vesonder, and D. Sheleheda, "Nfsight: NetFlow-based Network Awareness Tool," *Proceedings of 24th USENIX Large Installation Systems Administration Conference*, 2010, pp. 119-134.
[7] A. Wagner and B. Plattner, "Entropy Based Worm and Anomaly Detection in Fast IP Networks," *Proceedings of 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, 2005, pp. 172-177.
[8] S. Ranjan, S. Shah, A. Nucci, M. Munafò, R. Cruz, and S. Muthukrishnan, "DoWitcher: Effective Worm Detection and Containment in the Internet Core," *Proceedings of 26th IEEE International Conference on Computer Communications*, 2007, pp. 2541-2545
[9] A. Kind, M. Ph. Stoecklin, X. Dimitropoulos, "Histogram-Based Traffic Anomaly Detection," *IEEE Transactions on Network and Service Management*, vol. 6 (2), 2009, pp. 110-121.
[10] D. Brauckhoff, X. Dimitropoulos, A. Wagner, and K. Salamatian, "Anomaly Extraction in Backbone Networks using Association Rules," *Proceedings of 9th ACM SIGCOMM Internet Measurement Conference*, 2009, pp. 28-34.
[11] S. Weigert, M. Hiltunen, and C. Fetzer, "Community-based Analysis of Netflow for Early Detection of Security Incidents," *Proceedings of 25th USENIX Large Installation Systems Administration Conference*, 2011, pp. 241-252.
[12] K. Bartoš, M. Grill, V. Krmíček, M. Řehák, and P. Čeleda, "Flow Based Network Intrusion Detection System using Hardware-Accelerated NetFlow Probes," *Proceedings of CESNET Conference*, 2008, pp. 49-56.
[13] C. Wagner, J. François, R. State, and T. Engel, "Machine Learning Approach for IP-Flow Record Anomaly Detection," *Proceedings of 10th International IFIP TC 6 Conference on Networking*, 2011, pp. 28-39.
[14] G. Münz, S. Li, and G. Carle, "Traffic Anomaly Detection Using K-Means Clustering," *Proceedings of Leistungs-, Zuverlässigkeits- und Verlässlichkeitsbewertung von Kommunikationsnetzen und Verteilten Systemen, 4. GI/ITG-Workshop MMBnet 2007*.
[15] I. Paredes-Oliva, I. Castell-Uroz, P. Barlet-Ros, X. Dimitropoulos, and J. Solé-Pareta, "Practical Anomaly Detection Based On Classifying Frequent Traffic Patterns," *Proceedings of Workshops of 31th IEEE International Conference on Computer Communications*, 2012, pp. 49-54.
[16] R. Vaarandi, "Mining Event Logs with SLCT and LogHound," *Proceedings of 11th IEEE/IFIP Network Operations and Management Symposium*, 2008, pp. 1071-1074.
[17] J. McHugh and C. Gates, "Locality: A New Paradigm for Thinking About Normal Behavior and Outsider Threat," *Proceedings of 2003 New Security Paradigms Workshop*, 2003, pp. 3-10.
[18] R. Vaarandi, "Methods for Detecting Important Events and Knowledge from Data Security Logs," *Proceedings of 10th European Conference on Information Warfare and Security*, 2011, pp. 261-267.