Universidad del País Vasco
Euskal Herriko Unibertsitatea

Departamento de Arquitectura y Tecnología de Computadores
Konputagailuen Arkitektura eta Teknologia Saila

Universidad del País Vasco
Euskal Herriko Unibertsitatea
KONPUTAGAILUEN ARKITEKTURA ETA TEKNOLOGIA SAILA
DEPARTAMENTO DE ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

Generation of the database *gurekddcup*

EHU-KAT-IK-02-16

Informe de Investigación

Iñigo Perona
Olatz Arbelaitz
Ibai Gurrutxaga
José I. Martín
Javier Muguerza
Jesús M. Pérez

Ekaina 2016

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Departamento de Arquitectura y Tecnología de Computadores
Konputagailuen Arkitektura eta Teknologia Saila

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
KONPUTAGAILUEN ARKITEKTURA ETA TEKNOLOGIA SAILA
DEPARTAMENTO DE ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

Generation of the database *gurekddcup*

EHU-KAT-IK-02-16

Informe de Investigación

Iñigo Perona
Olatz Arbelaitz
Ibai Gurrutxaga
José I. Martín
Javier Muguerza
Jesús M. Pérez

Ekaina 2016

Acknowledgements: This work was funded by the University of the Basque Country UPV/EHU (BAILab, grant UFI11/45); by the Department of Education, Universities and Research of the Basque Government (grant IT-395-10); and by the Ministry of Economy and Competitiveness of the Spanish Government and by the European Regional Development Fund - ERDF (eGovernAbility, grant TIN2014-52665-C2-1-R).

Generation of the database *gurekddcup*

Iñigo Perona, Olatz Arbelaitz, Ibai Gurrutxaga,
José I. Martín, Javier Muguerza, Jesús M. Pérez

May 2008 - Translated during 2015

Contents

List of Tables

Listings

1 Introduction

The database gureKDDCup has been generated within the UADI project (Unsupervised Anomaly Detection for Intrusion detection system) [3, 2, 12] in which a classifier that detects intrusions or attacks in network based systems was developed. To develop this classifier we are going to use unsupervised classification techniques. The main distinctive feature of this project is that it uses the payload (body part of network packages) to detect attacks in network connections. The analysis of the payload to classify the connections is not a deeply analysed field, however, it seems that it is essential to detect attacks such as R2L (Remote to Local, its goal is to use resources without permission) and U2R (User to Root, its goal is to get root or administrative privileges without having them).

In the classification process we have to handle with a huge amount of connections and discover useful patterns among them. Therefore, this leads us to the Data Mining field. Moreover, we want our UADI system to be able to discover patterns or generate the model of network traffic automatically, that is, we want the learning process to be automatic, and to do it possible, we are going to use Machine Learning techniques [1].

The working environment of this project is the KDD (Knowledge Discovery in Databases), and as a consequence, to develop this project we are going to follow the steps established in KDD:

1. Collect data.
2. Evaluate the data.
3. Clean & transform data.
4. Select the cases or examples.
5. Learn a classifier.
6. Validation phase.
7. Set up the classifier.

The aim of this report is to explain how the database was generated to research upon it, that is, the first three steps.

2 The objective

The aim of this report is to explain the process we have followed to generate the database we used in the UADI project. The objective is to generate a database with similar characteristics to KDDCup99 [7] which is broadly used database in the scientific environment, taking as starting point the Darpa98 (DARPA Intrusion Detection Data Sets) [10, 6]. The generated database is called gureKDDCup and it has similar features to the ones in KDDCup99, but we added to it payload information and other features related to the connection such as IP address and port numbers. Next lines explains the steps followed to generate the KDDCup99 database because our aim is to repeat those steps as accurately as possible, to create KDDCup99 the database we need in UADI project, in other words, a new extension of the (KDDCup99+payload) that we called it gureKDDCup.

The source data of the KDDCup99 database was the Darpa98 dataset. To create Darpa98 the Cyber Systems and Technology Group (formerly the DARPA Intrusion Detection Evaluation Group) of MIT Lincoln Laboratory, under the Defense Advanced Research Projects Agency (DARPA ITO) and Air Force Research Laboratory (AFRL/SNHS) sponsorship, created a closed military network environment to simulate real traffic, normal traffic, attacks and intrusions. They recorded all network movements using tcpdump the sniffer program. All the simulated traffic was captured in tcpdump files. Thereafter from these files they extracted information such as outcome of executed commands in the simulation or network packages information. Therefore, they created a standard corpora for evaluation of computer network intrusion detection systems and to provide researchers with extensive examples of attacks and background traffic. This dataset is called Darpa98 and it was carried out in 1998.

Afterwards, they generated from Darpa98 a database in the same format the UCI Machine Learning Repository uses (to use it in the Machine Learning field), that is, a comma separated data matrix and they called it KDDCup99. This database was used for The Third International Knowledge Discovery and Data Mining Tools Competition (KDD99). The competition task was to build a network intrusion detector, a predictive model capable of distinguishing between 'bad' connections, called intrusions or attacks, and 'good' or normal connections.

To generate the KDDCup99 they had to identify the connections and their values for each attribute. These attributes were computed analysing the files generated by tcpdump command in the simulated network environment. They computed three types of attributes: intrinsic attributes (the attributes obtained by analysing the headers of network packages, such as, service-name and protocol), content attributes (the attributes obtained by analysing the payloads of network packages, such as, if the root user has got the shell or not and number of logins) and traffic attributes (the attributes that are computed by analysing the previous connections, such as, the number of connections accessing to the same IP address in the last two seconds or between the last 100 connections).

The database KDDCup99 is available in the UCI Machine Learning Repository with the name 'KDD Cup 1999 Data Set'. In KDDCup99, each example is a connection represented by many attributes.

Our objective is to extend KDDCup99, that is, to compute all attributes KDDCup99 has for each connection and to add the basic information (IP address, port numbers...) and the payload for each connection. To achieve this

objective, as the starting point were the files generated by the tcpdump sniffer program the sniffed files contained all the network packages transmitted in the simulated network, so, we could identify a connection and compute its attributes but we could not know the class of each connection, that is, if it belonged to an attack or to a normal connection.

To identify the connection class Darpa98 provides tcpdump.list file with the class of each connection for each sniffed tcpdump file, i.e., if it is an attack (and type of the attack) or not. This is possible because all the connections, normal traffic and attacks, were generated in a controlled environment. Although not every connection appearing in tcpdump files appear in tcpdump.list files they ensure that all attacks are included.

To create our database, gureKddcup, we need to repeat the same attributes computation carried out for KDDCup99. Hence, we used the explanations that Lee gives in his Ph.D. thesis [11].

In the database generation process, we used different tools for different tasks. First of all, we used Bro (The Bro Network Security Monitor) [9] to extract basic attributes (IP addresses, port names, service names...), intrinsic attributes, content attributes and payloads from tcpdump files, writing scripts that suits us. Secondly, we used bash-unix shell [4] and awk scripting language [5] to extract traffic attributes. Finally, to label the connections obtained with bro we matched them with the connections in tcpdump.list file and validate them comparing with KDDCup99. As a first approach, to carry out the latter task we used bash and awk but it took to long, and consequently we implement the matching and labelling process in C programming language [8].

3 gureKDDCup generation steps

The steps followed to create gureKDDCup database can be summarized as follows:

1. Get the connections: to ensure that the connections were identified correctly, at first we only extracted the basic information of the connection such as starting time of the connections, source's & destination's IP addresses, and source's & destination's port numbers.

2. Match the connections: to ensure that the connections were identified correctly, we matched them with the tcpdump.list file and checked the obtained matching degree. Further steps were only executed when the matching degree was acceptable.

3. Intrinsic attributes: Compute the attributes for each connection, using the networks' packages headers.

4. Content attributes: Compute the attributes for each connection, using the networks' packages payloads.

5. Traffic attributes: Compute the attributes for each connection, using the previous connections. Two options were computed: a fixed time-window and a fixed number of connections.

6. Add the payload: Add to each connection its payload, that is, the byte sequence sent by the specific application.

7. Label the connections using tcpdump.list: Each connection must be labelled as attack or normal traffic. This was done by matching each connection with one connection in the tcpdump.list file and taking its class to label it.

4 Obtain the connections from tcpdump

The script language provided by bro was used to define the policies used to extract the connections from the tcpdump files. Although, bro defines itself as an Intrusion Detection System (IDS), provides a powerful tool for traffic analysis.

It is worth mentioning that bro uses signatures, that is, it identifies known attacks by finding in its inner attack databases the sniffed sequences of characters. Unfortunately, it is easy to think that signature based methods are not able to detect new attacks. Hence, the UADI project becomes necessary: the generation of a system that does not use signatures, at least that does not use only signatures and models the payload part of networks packages. In this context, the necessity of a database to research and compare different algorithms appears. The aim of this report, is to explain how the database for evaluation was generated.

Beforehand consider necessary the explanation of some concepts: functions, events and policies.

- Policy: They are executable files of the bro IDS and its name's format is "xxx.bro". Although they are called policies in bro they are similar to scripts. See Listing 1.

Listing 1: Policy execution of previously sniffed tcpdump file.

```
$ bro -r tcpdump xxx.bro
```

- Events: They are the key of bro's policies. When bro is analysing the traffic from the network or from the stored tcpdump files, it follows the guidelines of the policy in execution (xxx.bro). These policies have events inside and when bro detects them, triggers the code written inside them, that is, the body of the event. There are many events for example: initialisation of the TCP connection or the end of the TCP connection. See Listing 2.

Listing 2: Checking all events of bro.

```
[installDirectory]/bro/policy/event.bif.bro
```

- Functions: Bro has them defined to make the analysis of the network traffic easier. We use them in the body of events' code. There are many functions, such as functions that return the protocol of the connection (TCP: Transmission Control Protocol, UDP: User Datagram Protocol, ICMP: Internet Control Message Protocol) or the connections' service name (http, ssh...). See Listing 3.

Listing 3: Checking all functions of bro.

```
[installDirectory]/bro/policy/bro.bif.bro
```

4.1 The detection of connections

Bro is based on events were we will write the desired code. For example, the event new_connection will trigger up when a new connection is detected within

the analysed traffic. In our case, all information of every connection was stored in a table, then, the needed variables were added and finally the connections with their attributes were written in the output file. The following events are the one used to detect connections in bro's policies.

- event new_connection(c: connection) {...}
- event connection_state_remove(c: connection) {...}

The first event, new_connection will be triggered when a new connection is detected in the network traffic being analysed and this event is in charge of saving the connection's characteristics in a global table to have them accessible in the rest of the policy. The second event, connection_state_remove will be triggered when the end of the connection is detected. Therefore, this event is in charge of finishing the computation and verification of all connection's features.

Although according to the definition of the event new_connection, it detects TCP, UDP and ICMP connections, it seems that it does detect all TCP connections but does not detect all UDP and ICMP connections. Therefore events listed on following lines were used.

- event udp_request(u: connection) {...}
- event udp_reply(u: connection) {...}
- event udp_contents(u: connection, is_orig: bool, contents: string) {...}

To detect connections there is not need to use the three UDP events: it is enough to use first two events or only the latter one. Initially we used the first two events because there were enough to detect connections but the need of adding the payload obliged us to use the last event: udp_contents. This event does provide the byte sequence belonging to the package's payload, thus, at last we used this one event for identifying the type (request or reply) of the package by its is_orig parameter.

All the ICMP events listed bellow are not necessary, that is, we can take away the first one (icmp_sent) because with the remaining events we can detect all ICMP packages with more information. All the information obtained from the execution of these events will be saved in a global table.

- event icmp_sent(c: connection, icmp: icmp_conn) {...}
- event icmp_echo_request(c: connection, icmp: icmp_conn, id: count, seq: count, payload: string) {...}
- event icmp_echo_reply(c: connection, icmp: count, id: count, seq: count, payload: string) {...}
- event icmp_unreachable (c: connection, icmp: icmp_conn, code: count, context: icmp_context) {...}
- event icmp_time_exceeded (c: connection, icmp: icmp_conn, code: count, context: icmp_context) {...}

In ICMP and UDP protocols the concept of the connection is not clear because they are not developed to keep a stable connection and as a consequence they do not have formal beginning and ending since these protocols admit to lose of packages in the communication. Nevertheless, it seems that bro takes as a connection any package exchanges between two machines (IP addresses) and pair of port numbers that defines the used service in a small window of time. Initially, we decided to take a connection as bro defines it. However, we changed this decision over the time because we did not obtain acceptable matching rates between the connections we generated and the connections in tcpdump.list. The matching is compulsory to label the connections as normal or attack. Therefore, we assumed each ICMP and UDP package to be an independent connection to improve the matching rates. This theoretically can be done because in ICMP and UDP protocols do not include the formal concept of the connection.

There exist some connections or packages in the tcpdump files that do not trigger any of the previous events. This could be because the presented events dismiss all the damaged packages, for example, the packages with "bad_TCP_checksum", "bad_UDP_checksum" and "bad_ICMP_checksum", known as wrong fragment packages. Consequently we had to make use of some more defined events to detect all connections and packages. Therefore, we used some lower level bro events (listed below) that they were triggered with all packages.

- event tcp_packet (c: connection, is_orig: bool, flags: string, seq: count, ack: count, len: count, payload: string) {...}
- event new_packet (c: connection, p: pkt_hdr) {...}
- event packet_contents(c: connection, contents: string) {...}

We are going to analyse those events more deeply in the section dedicated to payload extraction.

To finish with this subsection we would like to underline that to identify TCP connections uni-vocally as a first approach we used the starting time of the connection, pair of IP addresses as a machines identification (client and server sides) and pair of port numbers as service identification (client and server sides). Likewise, to identify the UDP and ICMP connections or packages, we took into account these features: the time stamp of the packages, pair of IP addresses and pair of port numbers.

At last, when matching our connections with the tcpdump.list connections we took into account only the starting time of the connections or packages that bro give us for all protocols (TCP, UDP and ICMP) the pair of IP addresses and port numbers. On the contrary, we did not take into account the duration attribute of the connections to do the matching for labelling the connections. Taking into account the starting time instead of the starting time with the duration, the matching rate increased and as consequence, also the amount of labelled connections.

4.2 Obtaining the features of connections

The information of each connection was retrieved from the parameters of the used events. Next lines present the attributes and sub-fields required to identify the connections, being 'c' a connection type record.

- Connection's start time: c$start_time-6*60min. We executed bro's policy in Central European Time (CET) time zone, that is, in UTC+1. On the contrary, the network simulation was carried out in an east USA (United States of America) laboratory, i.e, in UTC-5. Therefore the timestamps we extracted with bro from tcpdump files were 6 hours ahead. This made necessary to subtract 6*60 minutes to the timestamps obtained with bro, in order to match the connections extracted from the tcpdump file with the labelled connections of tcpdump.list.
- Origin's IP address: cidorig_h.
- Responders IP address: cidresp_h.
- Origin's port number: cidorig_p.
- Responders port number: cidresp_p.
- Get protocol: To obtain the protocol (TCP, UDP and ICMP) of each connection different options must be used. In events dedicated to TCP, UDP or ICMP the protocol is evident. But in other events we used the function offered in bro: get_port_transport_proto(id$resp_p).

At this point, I would recommend to check the following bro policies because they are very interesting and give us a good idea and hints about how bro's policies work.

- [installDirectory]/bro/policy/bro.init: When bro stars up, it shows the data structures and registers loaded by bro. Afterwards, it helps us finding out what features of a connection are available.
- [installDirectory]/bro/policy/bro.bif.bro: It lists all available functions in bro.
- [installDirectory]/bro/policy/event.bif.bro: It lists all available events in bro.
- [installDirectory]/bro/policy/conn.bro: It writes each detected connection line-by-line with some attributes. In this work, we generated a policy similar to this one, so it is very interesting to analyse it.

5 Comparison with tcpdump.list

To execute any implemented bro policy, for example, the file policy.bro the command shown in Listing 4 needs to be executed.

Listing 4: Policy execution to obtain connections.

```
$ bro -r tcpdump policy.bro > policy.list
```

The consequence of this command is to obtain a file containing lines for each connection with corresponding attributes. Thus, this file follows the format of the labelled tcpdump.list file.

The following step was to measure the matching rates between policy.list and the labelled tcpdump.list files. The matching was initially done using scripting languages such as, AWK and bash commands. However, we later realized that it was more effective and less time consuming to use C program language. Therefore, the explanations in this report are focused on C.

5.1 tcpdump.list VS policy.list

To compare the labelled tcpdump.list and our policy.list we need to have clear the format each of them has. Therefore, first of all, we are going to explain the format, characteristics and differences of the files:

- The fields appears in different order: the order of fields in tcpdump.list is the following: connection identifier, date, start time of the connection, duration, service name, protocol, origin machine's port number, destination machine's port number, origin machine's IP address, destination machine's IP address, if it is an attack or not (0 or 1, normal or attack respectively) and what kind of attack it is (if it is an attack then the name of the attack an otherwise the symbol "-"). Table 1 and 2 show some lines of tcpdump.list as an example. However, in policy.list the order of the fields changes to: connection identifier, the start time of the connection (in timestamp format), origin machine's port number, destination machine's port number, origin machine's IP address, destination machine's IP address, duration, protocol, service name, etc. Table 3 shows some lines of policy.list as an example.

- The formats of the time fields are different: the file tcpdump.list needs two fields (date and time) and the measurement precision is the second. On the other hand in policy.list, the time is represented as a timestamp and it needs as a consequence, a single field and moreover, it has a higher precision.

- The name of many services appears differently in both files: in tcpdump.list the service and protocol are given in the same field and moreover the service names are not identical. For example, what in tcpdump.list is called "domain", in policy.list is called "dns".

- The way to represent the protocols: in tcpdump.list the service name and the protocol appear in the same field: xxx (TCP), xxx/u (UDP) and xxx/i (ICMP). So, if only the service name appears the protocol is TCP and if the service name is followed by "/u" or "/i" the protocol is UDP or ICMP

1	07/13/1998	07:57:27	00:00:01	ntp/u	123	123	...
2	07/13/1998	07:57:27	00:00:01	ntp/u	123	123	...
3	07/13/1998	07:58:11	00:00:01	eco/i	-	-	...
4	07/13/1998	07:58:11	00:00:01	ecr/i	-	-	...
5	07/13/1998	07:58:35	00:00:01	eco/i	-	-	...
6	07/13/1998	08:00:19	00:00:01	domain/u	53	53	...
7	07/13/1998	08:00:19	00:00:01	domain/u	53	53	...
8	07/13/1998	08:00:19	00:00:01	domain/u	53	53	...
33	07/13/1998	08:04:19	00:00:01	smtp	1115	25	...
34	07/13/1998	08:04:25	00:00:01	eco/i	-	-	...
35	07/13/1998	08:04:27	00:00:01	eco/i	-	-	...
36	07/13/1998	08:04:30	00:00:01	eco/i	-	-	...
37	07/13/1998	08:04:32	00:00:01	eco/i	-	-	...

Table 1: Example of the first fields of tcpdump.list.

respectively. For example: http is TCP, ntp/u is UDP, eco/i is ICMP. In the format of the field can also find the following: "eco/i:ref8888". This belongs to an ICMP protocol but it includes more information. In the same way, when the service name is unknown the tcpdump.list shows "8888" and "8888/u" but it seems that in this kind of expressions the assignment of the protocol is not always accurate. This affirmation can be done after analysing the matching connections in tcpdump.list and policy.list and observing that "8888" field can be UDP even though it does not have "/u" at the end of the expression. Anyway, the protocol field is not mandatory to match connections. On the other hand, policy.list includes a field for the protocols and the names it can take are: tcp, udp, icmp.

- The port numbers are not always the same in both files: in tcpdump.list file all the connections have a port number assigned. However, the port number appears empty or unspecified "-" in most connections of ICMP type. In the ICMP protocol instead of using the port number concept there are two numbers specifying the type and the code of the ICMP message. Therefore, in some cases those codes or numbers appear in tcpdump.list. Thus, when tcpdump.list does not specify the port number we are going to try to obtain it from the service name field. On the other side, in policy.list bro always assigns the port number.

- The format of the IP addresses is not exactly the same: in tcpdump.list each sub-number of the IP addresses has three digits, for example: 172.-016.112.020. However, policy.list does not include left zeros to the sub-numbers, for example: 172.16.112.20.

Summing up, two connections are matched according to the starting time of the connection and the pair of IP addresses and port numbers. Note that in ICMP packages the starting time and the IP addresses are enough because ICMP does not have port abstraction (it has type and code numbers), so, tcpdump.list does not specify those fields. If we analyse the headers of the datagrams of ICMP network packages we will notice that there are not port numbers there. However, ICMP packages have a field to define the type of service called "type" and the "code" field to define the parameters or options chosen within the service.

1	...	172.016.112.020	192.168.001.010	0	-	
2	...	172.016.112.020	192.168.001.010	0	-	
3	...	192.168.001.005	192.168.001.001	0	-	
4	...	192.168.001.001	192.168.001.005	0	-	
5	...	192.168.001.005	192.168.001.001	0	-	
6	...	172.016.112.020	192.168.001.020	0	-	
7	...	172.016.112.020	192.168.001.020	0	-	
8	...	172.016.112.020	192.168.001.010	0	-	
33	...	172.016.114.169	196.227.033.189	0	-	
34	...	207.230.054.203	172.016.114.050	1	satan	
35	...	207.230.054.203	172.016.114.050	1	satan	
36	...	207.230.054.203	172.016.114.050	1	satan	
37	...	207.230.054.203	172.016.114.050	1	satan	

Table 2: Example of last fields of tcpdump.list.

1	900309447.573664	123	123	172.16.112.20	192.168.1.10	0.000000	udp	ntp_u	...
2	900309447.573664	123	123	192.168.1.10	172.16.112.20	0.001156	udp	ntp_u	...
3	900309491.415114	8	0	192.168.1.5	192.168.1.1	0.000000	icmp	echo_reply_i	...
4	900309511.572783	123	123	172.16.112.20	192.168.1.10	0.000000	udp	ntp_u	...
5	900309511.572783	123	123	192.168.1.10	172.16.112.20	0.001155	udp	ntp_u	...
6	900309491.415114	0	8	192.168.1.1	192.168.1.5	24.522417	icmp	echo_req_i	...
7	900309575.571715	123	123	172.16.112.20	192.168.1.10	0.000000	udp	ntp_u	...
8	900309575.571715	123	123	192.168.1.10	172.16.112.20	0.001157	udp	ntp_u	...
9	900309619.206720	53	53	172.16.112.20	192.168.1.20	0.000000	udp	domain_u	...
10	900309619.206720	53	53	192.168.1.20	172.16.112.20	0.003666	udp	domain_u	...

Table 3: Example of policy.list.

5.2 Exact match of the connections

Initially we have compared the connections splitting them up into protocols, to analyse them separately and make the suitable adaptations in our decisions to match as many of them as possible. During this matching process we found in tcpdump.list files some irregularities. For example, service name “ntp” and port number “8888” in the cases that the connections were UDP, that is, it should appear “ntp/u” and “8888/u” respectively. The hypothesis is confirmed because the connections extracted with bro (policy.list) suggest that those connections should belong to the UDP protocol and moreover, by definition, the ntp service normally works upon the UDP protocol. Those irregularities made very difficult to split up the tcpdump.list by protocols.

Therefore the final decision was not to divide the connections by protocol, and so, we do not take the protocols in consideration to match the connections.

5.3 Match interchanging connections’ origin and destination

Once we have matched the tcpdump.list and policy.list connections we analysed the unmatched connections and we noticed that some of them would have been matched if the IP addresses and port numbers would have been interchanged. So, we matched these connections interchanging the origin IP address and port number with destination ones in one of the “.list” files. In this way we were able to increase the matching rate.

5.4 Match connections within a time window

After matching the connections in policy.list as described in the previous sections we have analysed, if the unmatched connections have a pair within a time window. We have relaxed the matching criteria due to the following reasons: Firstly, we have noticed that many unmatched connections would have been matched if we had taken into account the connections within a time window. After all, the exact procedure followed to generate KDDCup99 is not known, so, we supposed that some of the used tools had introduce this time difference. Secondly, bro gives us the timestamp of the connections in milliseconds of preciseness, however tcpdump.list date field only fells to seconds of preciseness. Therefore, we found justifiable to match the connections that were generated within the same time window and between the same origin-destinations machines.

Hence, we have fixed the time window to plus-minus 3 seconds, that is, we took 3 seconds before and after each connection in tcpdump.list and analysed if any of the connections in policy.list was generated within these 6 seconds and between the same origin-destinations machines. We accepted the connections fulfilling these conditions as having the time field matched and we continued comparing the IP addresses and port numbers.

5.5 Match connections in a time window and interchanging connections' origin and destination

To match the last few unmatched connections we used the time window and interchanged connections' origin and destination machines, that is, we swap IP addresses and port numbers to match the connections more easily. As happened in the previous sections possible little misinterpretations, format dis-concordance and the lack of preciseness in time in tcpdump.list, have permitted us to make this decision.

5.6 Implementing the matching process: etiketatu.c

The next step was the labelling of the connections, after all this is the objective of matching. Later on, in the Appendix A the developed C programs and scripts will be explained.

5.7 Results of the matching process

Tables 4, 5, 6, 7, 8, 9, 10 and 11 show in detail the matching rates we obtained after each processing phases and in each week. The addition of the connections of all days of the week appears in SUM column whereas the number of labelled connections we will have in gureKddcup appears in bold. The tables also include in the last row the number of connections there are in tcpdump.list.

WEEK1	Mon.	Tues.	Wednes.	Thurs.	Fri.	SUM
directly	33715	31868	45532	36378	29905	177398
interchanging	33827	31976	45638	36488	29980	177909
time-window directly	33827	31976	45638	36489	29980	177910
time-window interchanging	33827	31976	45638	36489	29980	**177910**
gureKddcup.list	76958	75559	118184	92124	72928	435753
tcpdump.list	33856	32011	46250	36981	30017	179115

Table 4: Matching rates between tcpdump.list and policy.list in Week1.

WEEK2	Mon.	Tues.	Wednes.	Thurs.	Fri.	SUM
directly	36717	52157	34596	47539	10369	181378
interchanging	38030	53620	36235	49219	11683	188787
time-window directly	38030	53621	36235	49220	11683	188789
time-window interchanging	38030	53621	36235	49221	11683	**188790**
gureKddcup.list	86327	123709	78230	105635	30786	424687
tcpdump.list	38291	54368	36475	49476	11935	190545

Table 5: Matching rates between tcpdump.list and policy.list in Week2.

WEEK3	Mon.	Tues.	Wednes.	Thurs.	Fri.	SUM
directly	12506	12640	20331	218586	19440	283503
interchanging	12648	12806	20483	218730	20673	285340
time-window directly	12650	12806	23322	218733	20848	288359
time-window interchanging	12654	12806	23326	218735	20848	**1288369**
gureKddcup.list	40579	27801	53151	240167	84118	445816
tcpdump.list	13068	12825	96923	218899	21281	362996

Table 6: Matching rates between tcpdump.list and policy.list in Week3.

WEEK4	Mon.	Tues.	Wednes.	Thurs.	Fri.	SUM
directly	17731	23150	11637	28760	12286	93564
interchanging	18727	23751	12044	42116	15144	111782
time-window directly	18737	23846	12295	42987	15554	113419
time-window interchanging	18737	23863	12296	43403	15647	**113946**
gureKddcup.list	55947	47123	26305	63386	40407	233168
tcpdump.list	131998	25131	13211	47570	16280	234190

Table 7: Matching rates between tcpdump.list and policy.list in Week4.

WEEK5	Mon.	Tues.	Wednes.	Thurs.	Fri.	SUM
directly	25728	18523	26229	221218	268290	559988
interchanging	30606	22102	38900	223202	282650	597460
time-window directly	31252	22401	40451	223700	284199	602003
time-window interchanging	31575	22543	41196	223851	285138	**604303**
gureKddcup.list	218221	83412	63355	408883	320321	1094192
tcpdump.list	32992	24135	44698	227166	286992	615983

Table 8: Matching rates between tcpdump.list and policy.list in Week5.

WEEK6	Mon.	Tues.	Wednes.	Thurs.	Fri.	SUM
directly	27617	28768	245515	530651	42036	874587
interchanging	41461	41565	257640	535469	55009	931144
time-window directly	43293	43386	260313	536751	55613	939356
time-window interchanging	46743	47750	263545	537241	56082	**951361**
gureKddcup.list	87308	78111	308214	718663	274927	1467223
tcpdump.list	48960	50886	267081	544622	83193	994742

Table 9: Matching rates between tcpdump.list and policy.list in Week6.

WEEK7	Mon.	Tues.	Wednes.	Thurs.	Fri.	SUM
directly	44190	33778	48946	55440	249420	431774
interchanging	44331	33940	49098	56781	250166	434316
time-window directly	44332	33941	49425	56932	250184	434814
time-window interchanging	44332	33941	49425	56933	250184	**434815**
gureKddcup.list	98483	72876	109287	141503	325539	747688
tcpdump.list	44364	34642	49572	57385	250328	436291

Table 10: Matching rates between tcpdump.list and policy.list in Week7.

WEEK[1-7]	Mon.	Tues.	Wednes.	Thurs.	Fri.	SUM
directly	198204	200884	432786	1138572	631746	2602192
interchanging	219630	219760	460038	1162005	665305	2726738
time-window directly	222121	221977	467679	1164812	668061	2744650
time-window interchanging	225898	226500	471661	1165873	669562	**2759494**
gureKddcup.list	663823	508591	756726	1770361	1149026	4848527
tcpdump.list	343529	233998	554210	1182099	700026	3013862

Table 11: Matching rates between tcpdump.list and policy.list in all Weeks.

6 Feature calculation

The previous sections explained the matching of connections obtained with bro with ones in tcpdump.list getting acceptable matching-rates. At this point we considered adequate the matching results and next step was to compute the attributes that appear in KDDCup99 database for the connections we have obtained.

The attributes we have already obtained are essential to identify the connections: the pair of IP addresses, the pair of port numbers and the start-time of the connection. However they are not included in KDDCup99 because they are protocol required attributes which do not add important information for the machine learning process. Moreover, being the source of data an experiment in a simulated network which leads to a closed network makes these attributes dispensable. This could be an arguable decision in a real context. However in this work we have included them since there had to be calculated.

The KDDCup99 database has 41 predictor attributes for each connection and the class attribute defining the type of the connection is attack or normal. Those attributes are divided in 3 groups: intrinsic attributes, content attributes and traffic attributes.

1. Intrinsic attributes: There are 9 attributes and they are computed using the fields in the headers' area of the network packages. Those connection attributes are the following ones: duration, protocol type, service name, flag or status, source bytes or amount of sent bytes, destination bytes or amount of received bytes, if there are landed packages or if we are sending packages to ourselves, number of wrong fragments and lastly, number of urgent packages. All previous attributes are directly available in the header fields without doing any processing or doing little preprocessing as in variables such as in land variable or duration. For example in the land attribute we need a pair of IP addresses and port numbers to compute its value. Summing up, intrinsic attributes are available in package headers.

2. Content attributes: There are 13 attributes and they are computed finding signatures in the content area of the network packages. The payload or content area of the packages is service dependant, so those critical signatures are also service dependant and moreover they are proposed or created by experts' using their knowledge in network security. Some examples of content attributes are: number of failed logins or if it the root shell has been obtained.

3. Traffic attributes: There are 19 attributes and they are computed using the connections preceding the current connection. There are two criteria to determine the number of previous connections that are needed to compute these attributes: (1) time traffic features (9 features) (2) machine traffic features (10 features). The difference between the former group and the latter is the mode to select the previous connections. To calculate time traffic attributes the connections that occurred in the past 2 seconds were considered, while to calculate machine traffic attributes the previous 100 connections were taken into account. Therefore, to calculate these attributes unlike the previous ones, we need the information about a group of connections to be able to calculate them.

For the moment, our extended KDDCup99 (gureKDDCup) has the following attributes defined: the identifier of the connection, the start time of the connection, origin's port number, destination's port number, origin's IP address and destination's IP address and afterwards all 41 attributes of KDDCup99 database.

Next subsection explains all the assumptions made to determine the exact computation of KDDCup99 variables.

6.1 Intrinsic attributes

These attributes are calculated using the header area of the network packages.

1. duration: Duration of the connection in seconds, therefore, the attribute is of type integer. It can be obtained using the 'c: connection' data structure the events offer, in the following way: c$duration. The format can be modified according to the requirements as in this example: 'fmt("%.6f", c$duration);'

2. protocol_type: Data transmission protocol: TCP, UDP or ICMP, therefore this attribute's type is nominal. In the events which are directly related with UDP and ICMP connections, the protocol is defined by the event name. These events are the following ones: 'udp_request', 'udp_reply', 'udp_contents', 'icmp_sent', 'icmp_echo_request', 'icmp_echo_reply', 'icmp_unreachable', 'icmp_time_exceeded'. However, there are more general events too, such as 'new_connection' and 'connection_state_remove' that catch all type of connections, so, to determine the protocol we have to use the functions bro offers us. Here we list them:

   - is_tcp_port: function(p: port): bool;
   - is_udp_port: function(p: port): bool;
   - is_icmp_port: function(p: port): bool;
   - get_conn_transport_proto: function (cid: conn_id): transport_proto;
   - get_port_transport_proto: function(p: port): transport_proto;

   The TCP connections are detected by general events, so it is essential to use at least one of these functions to identify the connection's protocol. In our work we used the latter one: 'get_port_transport_proto' so, given a port number it returns the data transmission protocol used.

3. service: Application protocol, i.e., service name, such as, http, ftp, smtp, telnet... and other (if not very used service). This attribute is nominal. The service names have been defined based on the port number to service name table offered by bro in the policy 'port-name.bro'. Therefore to be able to use this conversion table in our policy, we have loaded this policy at the beginning of our policy by writing '@load port-name.bro' (the '@load' command is called which is equivalent to 'insert' or 'include' in other scripting or programming languages). Unfortunately, we found more service-names in KDDCup99 than in bro's table, consequently, we had to define all of them. As a consequence, we redefined the table to add more port numbers and service name rows and, we modified many service

names to be the same as in KDDCup99. Hence, we rebuilt the table in the beginning of our policy.bro.

In a nutshell, to obtain the port number, we asked it to the conversion table as shown in Listing 5, being the variable 'port_names' the conversion table. Being cidresp_p a transport type attribute, for example, 80/tcp.

Listing 5: Accessing to the port service conversion table.

```
local service = c$id$resp_p in port_names ?
port_names[c$id$resp_p] : 'other';
```

To identify the type of ICMP message, we created another table specifying the message type for each ICMP port number. See the Listing 6, being resp_p a port number of type string, for example, 8.

Listing 6: Accessing to the ICMP type conversion table.

```
local service = resp_p in icmp_port_names ?
icmp_port_names[resp_p] : 'other';
```

Even though we did the analysis of service names, at last we decided to use the server side port number to define the connection's used service. This decisions was made because the service names used in KDDCup99 do not follow any clear criteria. Therefore, we used the server's port number instead of the service name.

4. flag: The state of the connection. The different states or situations the connection can reach are the following ones:

   - SF: The connection has started and finished correctly.
   - S0: The request to establish the connection has not received any response.
   - S1: The connection has started (or established) but it has not finished.
   - S2: The connection has started (or established) and the source-side machine has requested to close it but the destination-side machine does not respond.
   - S3: The connection has started (or established) and the destination--side machine has requested to close it but the sourceside machine does not respond.
   - OTH: There have been no steps to establish the connection but traffic packages have been detected.
   - REJ: The request to establish the connection has been rejected.
   - RSTO: The connection has started (or established) correctly but the source-side machine has aborted using the RST (reset flag) field of the datagram.
   - RSTR: The connection has started (or established) correctly but the destination-side machine has aborted using the RST (reset flag) field of the datagram.

- RSTOS0: The source machine has sent a SYN signal and then a RST signal. However the destination machine has not responded with SYNACK signal.

conn.bro policy of bro has a function that gets the listed states, thus, this function was copied to our policy.bro. In the same way since conn.bro did not specify what to do when an analysed package was of type ICMP, we used the equivalent function found in icmp.bro policy of bro. All these functions return 13 possible states while KDDCup99 uses 11. As they add information we decided to keep the 13 values in our policy. The states that do not appear in KDDCup99 are those listed bellow.

- RSTRH: The destination machine sends a SYNACK signal and then a RST signal. In this case the origin machine did not send the SYN signal.
- SHR: The destination machine sends a SYNACK signal and then a FIN signal. In this case the origin machine did not send the SYN signal.

In Listing 7 the function "function conn_state (c: connection, trans: string): string" is shown to make the explanation clearer.

Listing 7: Function conn_state of conn.bro.

```
const conn_closed = {TCP_CLOSED, TCP_RESET};
function conn_state(c: connection,
                    trans: string): string{
  local os = c$orig$state;
  local rs = c$resp$state;
  local o_inactive = os==TCP_INACTIVE ||
                      os==TCP_PARTIAL;
  local r_inactive = rs==TCP_INACTIVE ||
                      rs==TCP_PARTIAL;

  if(trans=="tcp"){
    if(rs==TCP_RESET){
      if(os==TCP_SYN_SENT ||
         os==TCP_SYN_ACK_SENT ||
         (os==TCP_RESET &&
          c$orig$size==0 && c$resp$size==0))
        return "REJ";
      else if(o_inactive)
        return "RSTRH";
      else
        return "RSTR";
    }

    else if(os==TCP_RESET)
      return r_inactive ? "RSTOS0" : "RSTO";

    else if(rs==TCP_CLOSED && os==TCP_CLOSED)
      return "SF";

    else if(os==TCP_CLOSED)
      return r_inactive ? "SH" : "S2";

    else if(rs==TCP_CLOSED)
      return o_inactive ? "SHR" : "S3";

    else if(os==TCP_SYN_SENT && rs==TCP_INACTIVE)
      return "S0";

    else if (os==TCP_ESTABLISHED &&
             rs==TCP_ESTABLISHED)
```

```
      return "S1";

    else
      return "OTH";
  }

  else if(trans=="udp"){
    if(os==UDP_ACTIVE)
      return rs == UDP_ACTIVE ? "SF" : "S0";
    else
      return rs == UDP_ACTIVE ? "SHR" : "OTH";
  }

  else if(trans=="icmp")
    if(c$orig$size>0){
      if(c$resp$size>0)
        return "SF";
      else
        return "SH";
    }
    else if(c$resp$size>0)
      return "SHR";
    else
      return "OTH";
  }

  else
    return "OTH";
}
```

The UDP and ICMP packages do not have an strict control of transmitted packages, so, they differ from TCP, thus, as the algorithm indicates they trigger up 4 states: SF, S0, SHR and OTH.

5. src_bytes: the variables src_bytes and dst_bytes will be explained together in the next point.

6. dst_bytes: src_bytes indicates the number of bytes sent from the origin to the destination machine while dst_bytes denotes the bytes sent from the destination to the origin machine. Those attributes are offered by c connection structure which appears in many events as a parameter: 'fmt("%d", c$orig$size);' and 'fmt("%d", c$resp$size);'.

   The connection structure facilitates the accumulated transmitted data sizes. In the case of TCP connection we used it directly. However in UDP and the ICMP cases, if bro can tie up related nearby packages in a pseudo-connection, it accumulates the size of those package. So, as we defined each package as one connection in this case, we need to obtain the size of each package and we did that subtracting the previous packages' connection data.

7. land: '1' if the source and destination port and IP addresses are the same in one connection, otherwise '0'. This attribute was generated precisely to detect the land attack which is a DoS type attack. This attack sends a lot of request packages to the service provider machine with the same origin and destination addresses. The needed information to compute this attribute is offered by the connection structure, in this case the IP addresses and port numbers are extracted. In Listing 8 the code is presented:

Listing 8: Code to compute land attribute.

```
local land = (c$id$orig_p==c$id$resp_p &&
              c$id$orig_h==c$id$resp_h) ? 1 : 0;
```

8. wrong_fragment: number of wrong fragments in a connection. According to Lee's thesis [11] if the length of the package (number of bytes) is not multiple of 8 this package will have a wrong fragment. Therefore, a TCP connection can have many wrong fragments while UDP an ICMP connections at most 1.

   To compute this attribute the following event was utilised: "event new_packet (c:connection, p: pkt_hdr)...". Concretely the p registry was used, its piplen field to be more precise. This field stores the length of the IP datagram. Computing the variable as defined previously the values were very high comparing with KDDCup99. Therefore, we extracted a package from a higher layer of abstraction, so we used the TCP, UDP and ICMP packages' length which are encapsulated in the IP datagram. In this way, we used the following code: $p\$tcp\$dl\%8! = 0$ (dl: DataLength), $p\$udp\$ulen\%8! = 0$ and $p\$ip\$len\%8! = 0$. Even though the values had decreased, the value of this variable was very high comparing with the 10% sample of the KDDCup99. Hence, we researched in new options.

   Bro has a event that triggered up when it noticed a weird things are happening in the connection, which is "event conn_weird(name: string, c:connection)...". These weird happenings are of many types and more information can be found in the bro policy "weird.bro". For our aim we labelled as a weird the connections triggered by this event which in the 'name' attribute had the following strings: "bad_ICMP_checksum", "bad_TCP_checksum" and "bad_UDP_checksum". Therefore, when bad checksum was found the number of weird packages in the connection was incremented. The checksum or hash sum is a small-size datum, in this case of the IP datagram, to detect errors which may have been introduced during its transmission. For example, a very simple checksum function is to add one bit that it makes the amount of 1 bit even in the IP datagram, however there are more complex and effective CRC functions to detect almost all kind of changes could happen in the datagram.

9. urgent: The number of urgent packages transmitted, that is to say, the packages which have the urgent bit activated. For this aim the "event new_packet (c:connection, p:pkt_hdr)" event was used. To know if the package of the event has the urgent bit activated we evaluated this expression: ptcpflags ≥ 32. As the flag field is a counter, we do not know with how many bits it represented, but from the definition of the datagram fields that embeds the urgent bit, we think that it has 6 bits. So, evaluating the expression we ensured that the last bit, the urgent bit, was activated.

6.2 Content variables

The content variables are calculated applying functions to the payload, i.e., the area of the packages area where application data is transmitted.

10. hot: number of hot actions in a connection. According to Lee's thesis is defined as a hot action entering into the system's directories, creating a program and running it.

Initially we used the hot attribute provided by the c connection structure: c$hot. The value of this attribute was always 0, i.e., it was never activated with the tcpdump files we were working with, so we decided to explore other ways. In addition, the manual of bro stated that the implementation of bro was not reliable. Analysing the policy conn.bro which bro offers as an example, the hot attribute is incremented when the actions related to interruption of connections occur. As a consequence, we concluded that the meaning of bro's hot attribute and the Lee's hot variable are not the same so we computed it using our intuition.

To compute the hot variable, the definition given by Lee and the event "event login_input_line(c: connection, line: string)" were used. This event is triggered up when bro detects the payload that a command is entered. We analysed the commands found in line parameter to decide to increment the hot attribute or not.

- To enter to system directories: in the given string a command is detected meaning access to a system directory. Analysing the given string we realized that users start their remote shell sessions in "/home/user" directory and to change from directory to directory they use absolute paths like "cd /usr" and not relative paths "cd ../../usr". On one hand, we detected fixed commands such as "cd /". On the other hand, as the traffic was generated in a controlled network with controlled commands, we analysed accessed folders and we saw that in the sniffed traffic appear directories containing "dir" into the name to create simulated activity. Those folders are created in each user's workspace, so they do not have system files, thus they are not "hot" folders or to move there it is not a "hot" action. Therefore, all movements to folders that do not contain "dir" were labelled as hot. See Listing 9.

  Listing 9: Detecting access to system folders.

```
if (/cd \// in line && !(/dir/ in line))
  ++myconns[c$id]$hot;
```

- To create programs and run them: users use the compiler gcc to create programs, thus the Listing 10 shows an example of creation of a program.

  Listing 10: Creation of one program.

```
$ /bin/gcc probe.c -o probe.out
```

  Therefore, the condition to find creation of programs will be shown in the Listing 11.

Listing 11: Condition which detects the creation of programs.

```
if (/\/bin\/gcc/ in line || /\/bin\/g++/ in line)
  ++myconns[c$id]$hot;
```

Besides, the analysis of how the created programs are executed was done and the pattern shown in the conditions of Listing 12 were extracted.

Listing 12: Condition which detects the executions of programs.

```
if(/^\.\// in line)
  ++myconns[c$id]$hot;
if(/^\/tmp\/[0-9]+/ in line)
  ++myconns[c$id]$hot;
```

That is to say, created programs are executed using absolute paths which are situated in the folder tmp ("/tmp/12345") and programs executed using relative paths start the path indicating the current folder ("./"). Of course, programs created by gcc follow this guidelines.

- To execute programs: comparing the statistic values obtained for the variable hot in KDDCup99 and gureKDDCup, using only the hot patterns presented above, it can be observed that gureKDDCup values are smaller. Therefore, we decided to take into account whichever program execution, however the application of this hot pattern boost the variable value, thence we decided to count the executions of most frequent programs. To select these programs we did use of top command's output where the most resource consuming at execution programs are listed. Thus we took into account the programs appeared in this output.

  In fact, analysing the line attribute of the event "event login_input_line(c: connection, line: string)" it can be seen that the execution of the top commands is frequent and thus makes sense because in darpa98 the use of top's output was analysed thoroughly. Hence our intention is to detect programs which are long time in the processor by analysing the output of top command in the event "event login_output_line(c:connection, line: string)" and then to find those programs in the input commands analysing the event login_input_line. In Listing 13 shows the exact programs we found interesting.

Listing 13: Detecting the most used programs.

```
if(/a.out/ in line || /auditd/ in line ||
   /automountd/ in line || /cron/ in line ||
   /find/ in line || /fsck/ in line || /ftp/ in line ||
   /in.comsat/ in line || /inetd/ in line ||
   /in.ftpd/ in line || /init/ in line ||
   /in.telnetd/ in line || /kerbd/ in line ||
   /keyserv/ in line || /lockd/ in line ||
   /login/ in line || /lp.cat/ in line ||
   /lpNet/ in line || /lpsched/ in line ||
   /lp.tell/ in line || /lynx/ in line ||
   /mail/ in line || /man/ in line || /mlp/ in line ||
   /more/ in line || /netscape/ in line ||
   /nscd/ in line || /primes/ in line ||
   /sendmail/ in line || /sh/ in line ||
   /sleep/ in line || /sshd/ in line ||
   /statd/ in line || /syslogd/ in line ||
   /tcsh/ in line || /telnet/ in line || /tex/ in line ||
   /top/ in line || /ttymon/ in line || /vi/ in line ||
   /vold/ in line || /xntpd/ in line ||
   /ps_monitor/ in line)
     ++myconns[c$id]$hot;
```

It seems that some programs are not dangerous such as some daemons which are waiting for a request to process or a command-line web browser and so on. Still we included them in this hot list to maintain coherence although many could be removed because they

were not executed or activated from the command-line. Nevertheless gureKDDcup's hot values are still lower than in KDDCup99.

11. num_failed_logins: number of failed logins in a connection. On the one hand we used the event "event login_failure(c: connection, user: string, client_user: string, password: string, line: string)" to compute this attribute and on the other hand in the event "event login_output_line(c: connection, line: string)" we counted the apparition of the string as shown in Listing 14.

Listing 14: Detecting the failed logins.

```
if (/Login incorrect/ in line)
  num_failed_logins(c);
```

12. logged_in: it takes two values, 1 if the loggings were correct in a connection, 0 otherwise. In this case bro's event "event login_success(c: connection, user: string, client_user: string, password: string, line: string)" was used. Still the number of obtained 1 values was lower than in KDDCup99, so we thought in using signatures because they are somehow related with services that require logging process. For example the event "event telnet_signature_found(c: connection, is_orig: bool, len: count)" finds the action of connecting to a machine trough telnet. Bro can find many actions like this comparing payloads with its inner signature base which can be found in the folder /usr/local/bro/policy/sigs. Therefore, to create gureKDDCup we used the following events:

    - event login_success(c: connection, user: string, client_user: string, password: string, line: string) {...}
    - event login_terminal(c: connection, terminal: string) {...}
    - event login_display(c: connection, display: string) {...}
    - event login_prompt(c: connection, prompt: string) {...}
    - event ssh_signature_found(c: connection, is_orig: bool) {...}
    - event telnet_signature_found(c: connection, is_orig: bool, len: count) {...}
    - event rlogin_signature_found(c: connection, is_orig: bool, num_null: count, len: count) {...}
    - event root_backdoor_signature_found(c: connection) {...}
    - event ftp_signature_found(c: connection) {...}
    - event napster_signature_found(c: connection) {...}
    - event gnutella_signature_found(c: connection) {...}
    - event kazaa_signature_found(c: connection) {...}
    - event http_signature_found(c: connection) {...}
    - event http_proxy_signature_found(c: connection) {...}
    - event smtp_signature_found(c: connection) {...}
    - event irc_signature_found(c: connection) {...}

- event gaobot_signature_found (c: connection) {...}

13. num_compromised: Lee's thesis defines this attribute as the number of "file path not found" error in a connection.

    On the one hand, from the event "event login_output_line(c: connection, line: string)" the output of the commands executed can be obtained. In this output the number of apparitions of the string "File Not found" were counted. Even though the number of detected strings were lower than in KDDCup99, thus we decided to generalize the "File Not found" string to "not found". Likewise, instead of focusing only in not found files we counted all not found requests, such as, not found commands. The condition used is shown in Listing 15.

Listing 15: Detecting the failed logins.

```
if (/[nN]ot [fF]ound/ in line)
  ++myconns[c$id]$num_compromised;
```

14. root_shell: it takes two values, 1 if the user obtained the root shell in a connection and 0 otherwise. Analysing the input and output of the command-line activity we noticed that to obtain the root shell always the command "$ su - root" was executed. Afterwards the password was provided and if it was correct the following line was printed: "Sun Microsystems Inc. SunOS 5.5 Generic November 1995". The detection of this pattern was interpreted as logging as a root. Therefore, in the event "event login_output_line(c: connection, line: string)" the code presented in the Listing 16 was added.

Listing 16: Detecting the root shell access.

```
if (/su - root/ in line){
  myconns[c$id]$rootDa = T;
}
if (/^Sun Microsystems Inc. SunOS 5.5 November 1995/ in line){
  if (myconns[c$id]$rootDa){
    ++myconns[c$id]$root_shell_num;
    myconns[c$id]$rootDa = F;
  }
}
```

    Listing 16 shows the sequence of commands that counts the number of times the user got the root shell, thus, to convert it to a binary value the line shown in Listing 17 was included.

Listing 17: Converting the times the root shell is obtained to a binary value.

```
local root_shell = (root_shell_num > 0) ? 1 : 0;
```

15. su_attempted: it takes two values, 1 if the user used the "su" command in a connection and 0 otherwise. However, analysing this attribute in KDDCup99 we noticed that it takes also the value 2, so we decided to redefine the attribute as the times the user tries the command "su". To know if in the command-line the command "su" was tried, in the event "event login_input_line(c: connection, line: string)" the code shown in Listing 18 was written.

Listing 18: Detecting the su attempt in the command-line.

```
if ( /su -/ in line)
  ++myconns[c$id]$su_attempted;
```

Now, we want to implement the original definition, it is straightforward to compute it: if the "su" is used zero times the attribute value remains 0 and 1 otherwise.

16. num_root: the number of operations done in root mode in a connection. To compute this attribute the output of the event "event login_output_line(c: connection, line: string)" is taken and the number of prompts that start with "root@" are counted. In other words, the number of lines executed as a root are counted and in Listing 19 the implementation of the computation is shown.

Listing 19: Counting the number of operations done as a root.

```
if (/root@/ in line)
  ++myconns[c$id]$num_root;
```

On the other hand, the events that provide user information have been taken and if the user triggering the event was root the counter of num_root was incremented.

- event finger_request(c: connection, full: bool, username: string, hostname: string) {...}
- event ident_reply(c: connection, lport: port, rport: port, user_id: string, system: string) {...}
- event rsh_request(c: connection, client_user: string, server_user: string, line: string, new: bool) {...}
- event rsh_reply(c: connection, client_user: string, server_user: string, line: string) {...}
- event pop3_login_success(c: connection, is_orig: bool, user: string, password: string) {...}
- event pop3_login_failure(c: connection, is_orig: bool, user: string, password: string) {...}
- event irc_who_line(c: connection, target_nick: string, channel: string, user: string, host: string, server: string, nick: string, params: string, hops: count, real_name: string) {...}
- event irc_whois_message(c: connection, server: string, users: string) {...}
- event irc_whois_user_line(c: connection, nick: user: string, host: string, real_name: string) {...}
- event irc_oper_message(c: connection, user: string, password: string) {...}
- event irc_kick_message(c: connection, prefix: string, chans: string, users: string, comment: string) {...}
- event irc_names_info(c: connection, c_type: string, channel: string, users: string_set) {...}

17. num_file_creations: number of times a file is created in a connection. To its computation in the event "event login_output_line(c: connection, line: string)" the string "New file" was found because it is the output given by the text editor used when a file is created. Listing 20 shows the code used to count it.

Listing 20: Counting the New file string for attribute num_file_creations.

```
if(/\[New file\]/ in line)
  ++myconns[c$id]$num_file_creations;
```

On the other hand the event "event login_input_line(c: connection, line: string)" was analysed to find this operations that create files in the system, such as, copy, move or redirection command or operations. Listing 21 shows the code used.

Listing 21: Counting operations related to the attribute num_file_creations.

```
if(/cp[ \t]/ in line || /mv[ \t]/ in line || /cat >/ in line)
  ++myconns[c$id]$num_file_creations;
```

18. num_shells: the number of prompt lines used in a connection. This was interpreted as follows: number of logins of normal users. When the user inserts the user-name and the password the system confirms the login by writing a line that starts as follows: "Last login: ". Therefore, in the event "event login_output_line(c: connection, line: string)" the condition shown in the Listing 22 was added to count the number of no root shells the user obtains, in this way the root shells are not counted because they had already been counted in the attribute root_shell.

Listing 22: Counting operations related to the attribute num_file_creations.

```
if (/^Last login:/ in line)
  ++myconns[c$id]$num_shells;
```

19. num_access_files: number of operations done in control files in a connection. It was not clear how to implement this definition because the control files can be all human-readable files that are not in /home folder, however some files hanging from homeuser are also control files such as .bashrc. Therefore, we needed to define a list of control files appeared and check if they were accessed or not.

    This way, first the programs used to edit files were analysed: vi, rm and cat. We analysed the line parameter of the event login_input_line, thus, we considered the operations write, remove and read. With this information, from all accessed files control-files were detected. Analysing the control file list and the operations done to them we noticed that they are rarely accessed and mostly only to read with the cat command. Accordingly we added the condition shown in Listing 23 in the event "event login_input_line(c: connection, line: string)".

Listing 23: Counting operations in control files.

```
if (/(vi|rm|cat) \/etc\/passwd/ in line ||
    /(vi|rm|cat) \/usr\/include\/stdlib.h/ in line ||
    /(vi|rm|cat) \/usr\/include\/stdio.h/ in line ||
```

```
    /(vi|rm|cat) \/usr\/include\/wait.h/ in line ||
    /(vi|rm|cat) \/usr\/include\/archives.h/ in line ||
    /(vi|rm|cat) \/usr\/include\/wctype.h/ in line ||
    /(vi|rm|cat) \/usr\/include\/errno.h/ in line)
      ++myconns[c$id]$num_access_files;
```

The values of this attribute were higher in KDDCup99, thus we though about including a more general condition. Analysing more deeply the commands used in the simulated network, due to the high apparition of ".tex" extension, we deduced that the files created as common files, were generated in LaTeX markup language. Therefore, assuming the common files were those of extension ".tex" we considered all other extensions as a control files. See Listing 24.

Listing 24: Counting operations in control files.

```
if(/cat \// in line ||
   (/vi / in line && !(/.tex/ in line)) ||
   (/rm / in line && !(/.tex/ in line)) )
     ++myconns[c$id]$num_access_files;
```

20. num_outbound_cmds: number of outbound commands in a FTP session. The outbound option of ftp is one of many methods FTP has to send files to FTP servers and is based in RCP (remote copy). This option does not encrypt the user-name and the password. And as a consequence, it is not recommended to use it. Being dangerous to work with outbound, in KDDCup99 was included via this attribute. To compute this attribute in the event "event ftp_request(c: connection, command: string, arg: string)" the parameter command was analysed to find if it was outbound. In Listing 25 appears the code used in the event.

Listing 25: Counting ftp-outbound executions.

```
if (command == "outbound")
  ++myconns[c$id]$num_outbound_cmds;
```

21. is_hot_login: it takes two values, 1 if the user is logged in with superuser privileges, i.e., as a root or adm; 0 otherwise. The computation of this attribute is explained in the next attribute, is_guest_login. Listing 26 shows the code included in the events discussed in the next attribute, is_guest_login.

Listing 26: Counting hot logins.

```
if (user == "root" || user == "adm")
  ++myconns[c$id]$is_hot_login;
```

And to convert the frequency to a binary value the code in Listing 27 was used.

Listing 27: Convert frequency to binary.

```
local is_hot_login =
  (c$id in myconns && myconns[c$id]$is_hot_login>0) ? 1 : 0;
```

22. is_guest_login: it takes two values, 1 if the user is logged in with basic permissions, with no privileges, i.e., as a guest, anonymous or visitor; and 0 otherwise.

    The event used to compute this attribute was "event login_output_line(c: connection, line: string)" and we searched for "login guest" and "login visitor" its line parameter. The number of guest logins was very low comparing with KDDCup99, thus we added all events which contain user in its parameters and we found the string "guest", "visitor" and so on.

    - event finger_request(c: connection, full: bool, username: string, hostname: string) {...}
    - event rsh_request(c: connection, client_user: string, server_user: string, line: string, new: bool) {...}
    - event rsh_reply(c: connection, client_user: string, server_user: string, line: string) {...}
    - event pop3_login_success(c: connection, is_orig: bool, user: string, password: string) {...}
    - event pop3_login_failure(c: connection, is_orig: bool, user: string, password: string) {...}
    - event irc_who_line(c: connection, target_nick: string, channel: string, user: string, host: string, server: string, nick: string, params: string, hops: count, real_name: string) {...}
    - event irc_whois_message(c: connection, server: string, user: string) {...}
    - event irc_whois_user_line(c: connection, nick: string, user: string, host: string, real_name: string) {...}
    - event irc_open_message(c: connection, user: string, password: string) {...}
    - event irc_kick_message(c: connection, prefix: string, chans: string, users: string, comment: string) {...}
    - event irc_names_info(c: connection, c_type: string, channel: string, users: string_set) {...}

    And inside these events the code presented in the Listing 28 was included.

Listing 28: Counting guest logins.

```
if (user=="guest" || user=="anonymous" || user=="visitor")
  ++myconns[c$id]$is_guest_login;
```

Since the attribute's definition requires a binary value, the conversion was done using the code in Listing 29.

Listing 29: Convert frequency to binary.

```
local is_guest_login =
  (c$id in myconns && myconns[c$id]$is_guest_login>0) ? 1 : 0;
```

6.3 Traffic variables

The traffic variables are computed using a set of connections occurred before the current one.

To compute the intrinsic and content attributes bro's policies were used. However, to compute the traffic attributes the information of the current connection and the previous ones are needed, precisely the connections established in the 2 previous seconds (time traffic variables) and the last 100 connections established (machine traffic variables).

There are 19 traffic variables: 9 variables for time traffic variables and 10 machine traffic variables. The two groups of variables are almost the same in their computation, the only difference is the window of connections used and that in machine traffic variables there is one more.

6.3.1 Time traffic variables

Time traffic attributes are computed using the features of the connections happened in the last 2 seconds. In the following lines the exact definition of these attributes are listed.

23. count: number of connections to the same destination IP address as the current connection's destination IP address (in the last 2 seconds).

24. srv_count: number of connections to the same destination port number as the current connection (in the last 2 seconds).

25. serror_rate: percentage of connections that satisfied the condition of count attribute and also the "SYN" error occurred (in the last 2 seconds). We did two interpretation of the meaning of the SYN error. The first one was to see if the flag variable had S0, S1, S2 or S3 values because those values happen when there is a problem with the starting or ending process of the connection. The second option was to use the event "event conn_weird(name: string, c: connection) {...}" found analysing the weird.bro example policy of bro. In this event the parameter name was analysed to find one of the following strings related with synchronization problems:

    - "bad_SYN_ack"
    - "baroque_SYN"
    - "connection_originator_SYN_ack"
    - "data_without_SYN_ACK"
    - "repeated_SYN_reply_wo_ack"
    - "repeated_SYN_with_ack"
    - "SYN_after_close"
    - "SYN_after_partial"
    - "SYN_after_reset"
    - "SYN_inside_connection"
    - "SYN_seq_jump"

- "SYN_with_data"
- "unsolicited_SYN_response"

The first strategy obtained proper values according to KDDCup99 database, thus, we used it in gureKDDCup.

26. srv_serror_rate: percentage of connections that satisfied the condition of srv_count attribute and also the "SYN" error occurred (in the last 2 seconds). The SYN error was defined the same way it was done for attribute serror_rate.

27. rerror_rate: percentage of connections that satisfied the condition of count attribute and were also "REJ" connections (in the last 2 seconds). To be a REJ connection is to have the value REJ in the flag attribute.

28. srv_error_rate: percentage of connections that satisfied the condition of srv_count attribute and are also "REJ" connections (in the last 2 seconds).

29. same_srv_rate: percentage of connections that satisfied the condition of count attribute and were also using the same service as the current one (in the last 2 seconds).

30. diff_srv_rate: percentage of connections that satisfied the condition of count attribute and were NOT using the same service as the current one (in the last 2 seconds). It seems that the value of this attribute should be computed by doing $1 - same_srv_rate$, in this case the information would be repeated or 100% correlated in the database. However, this is not the case because we assumed that the service-name "other" corresponds to different service-names.

31. srv_diff_host_rate: percentage of connections that satisfied the condition of srv_count attribute added to the connections that have different destination IP addresses to the current one (in last 2 seconds).

    Listing 30 presents the algorithm used to compute traffic variables. The number "(1)" makes reference to the current connection for which we are computing the variables and the number "(2)" to the connections in the last 2 seconds.

Listing 30: Algorithm of time traffic variables.

```
while connections_of_policy.list (1) loop
  lerroa= $0
  while connections_of_last_two_seconds (2) loop
    if(resp_h1==resp_h2){
      count= count + 1
      if(flag2=="S0"||flag2=="S1"||
         flag2=="S2"||flag2=="S3")
        serror= serror + 1
      if (flag2=="REJ")
        rerror= rerror + 1
      if (zerb2!=  other  && zerb1==zerb2)
        same_srv= same_srv + 1
      if (zerb1!=zerb2)
        diff_srv= diff_srv + 1
    }
    if(resp_p1==resp_p2){
      srv_count= srv_count + 1
      if(flag2=="S0"||flag2=="S1"||
```

```
          flag2=="S2" || flag2=="S3")
        srv_serror= srv_serror + 1
        if(flag2=="REJ")
          srv_error= srv_error + 1
        if(resp_h1!=resp_h2)
          srv_diff_host= srv_diff_host + 1
        if(orig_p1==orig_p2)
          same_src_port= same_src_port + 1
      }
  endloop
  if (count!=0){
    serror_rate= serror/count
    rerror_rate= rerror/count
    same_srv_rate= same_srv/count
    diff_srv_rate= diff_srv/count
  }
  else{
    serror_rate= 0
    rerror_rate= 0
    same_srv_rate= 0
    diff_srv_rate= 0
  }
  if (srv_count!=0){
    srv_serror_rate= srv_serror/srv_count
    srv_error_rate= srv_error/srv_count
    srv_diff_host_rate= srv_diff_host/srv_count
  }
  else{
    srv_serror_rate= 0
    srv_error_rate= 0
    srv_diff_host_rate= 0
  }
  same_src_port_rate= same_src_port/100
  print lerroa" " count" " srv_count" "
  serror_rate" " srv_serror_rate" "
  rerror_rate" "srv_error_rate" "
  same_srv_rate" "diff_srv_rate" "
  srv_diff_host_rate
endloop
```

6.3.2 Machine traffic variables

Machine traffic variables are computed taking into account the features of the last 100 connections. Each variable's definition is almost the same as in Section 6.3.1 (time traffic variables), the only difference is that to compute these machine traffic variables, the connections' window is changed to the previous 100 connections (if there are). Therefore the definitions in this section are not as detailed as in Section 6.3.1 (time traffic variables). However one extra attribute is defined in this group of variables: dst_host_same_src_port_rate

32. dst_host_count: number of connections to the same destination IP address as the current connection's destination IP address (in the last 100 connections).

33. dst_host_srv_count: number of connections to the same destination port number as the current connection's destination port number (in the last 100 connections).

34. dst_host_same_srv_rate: percentage of connections that satisfied the condition of dst_host_count attribute and are also using the same service as the current one (in the last 100 connections).

35. dst_host_diff_srv_rate: percentage of connections that satisfied the condition of dst_host_count attribute and were NOT using the same service as the current one (in the last 100 connections).

36. dst_host_same_src_port_rate: percentage of connections that satisfied the condition of dst_host_count and were also connected to the same source port number as the current connection (in the last 100 connections).

37. dst_host_srv_diff_host_rate: percentage of connections that satisfied the condition of dst_host_srv_count attribute and also the connections that have different destination IP addresses to the current one (in the last 100 connections).

38. dst_host_serror_rate: percentage of connections that satisfied the condition of dst_host_count attribute and also the "SYN" error occurred (in the last 100 connections).

39. dst_host_srv_serror_rate: percentage of connections that satisfied the condition of dst_host_srv_count attribute and also the "SYN" error occurred (in the last 100 connections).

40. dst_host_rerror_rate: percentage of connections that satisfied the condition of dst_host_count attribute and were also "REJ" connections (in the last 100 connections).

41. dst_host_srv_error_rate: percentage of connections that satisfied the condition of dst_host_srv_count attribute and were also "REJ" connections (in the last 100 connections).

The 41 attributes defined in Section 6.3, 6.2 and 6.1 constitute the KDD-Cup99 (or its variation).

6.4 The programs used to compute the variables

The intrinsic and content variables were computed by creating bro's policies as shown in Listing 31.

Listing 31: Bro's execution to compute intrinsic and content variables.

```
$ bro -r tcpdump policy.bro > policy.list
```

The traffic variables were computed by taking policy.list as input and coding in C to compute them. After compiling the C file trafAld.out was generated and executed as in Listing 32. In our case we provided as programs parameters the sub-folders the policy.list file is situated in and the program will write the trafAld.list file with the traffic variables in the same folder.

Listing 32: C program's execution to compute traffic variables.

```
$ ./trafAld.out policy.list Week7 Monday
```

6.5 gureKDDCup.list vs KDDCup10percent.data

Once KDDCup99 variables were computed (or our variation of them) we compared them to the 10% sample of KDDCup99 and verified if the frequency and range of values were similar in gureKDDCup and in KDDCup99 (in both senses). In this process we tried to ensure that the database created was reliable. To extract statistics the following scripts were prepared:

- The script policyStats.sh (see the execution in Listing 33).

Listing 33: C program's execution to compute statistics for gureKDDCup.

```
$ ./policyStats.sh trafAld.list > trafAld.est
```

- The script kddcupStats.sh (see the execution in Listing 34).

Listing 34: C program's execution to compute statistics for gureKDDCup.

```
$ ./kddcupStats.sh kddcup10percent.data > kddcup.est
```

Afterwards we compared the two files obtained in Listing 33 and in Listing 34. The results are shown in Appendix B.

7 The Payload

Payload is the name taken by the data area of the traffic packets and this section will explain how it has been added to our previously computed KDDCup99 attributes. For each connection 3 files were stored with the number of connection as file name and differentiated by their extensions which are ".a", ".b" and ".c", i.e., "connectionNumber.a", "connectionNumber.b" and "connectionNumber.c". In "connectionNumber.a" files we accumulated payloads sent by the client side while in "connectionNumber.b" we accumulated payloads sent by the server side, finally in "connectionNumber.c" we accumulated all payloads in sequential order according to their timestamp.

The event used to obtain the payloads are the following ones:

- event packet_contents (c: connection, contents: string) {...}
- event udp_contents (u: connection, is_orig: bool, contents: string) {...}
- event tcp_packet (c: connection, is_orig: bool, flags: string, seq: count, ack: count, len: count, payload: string) {...}
- event icmp_echo_request (c: connection, icmp: icmp_conn, id: count, seq: count, payload: string) {...}
- event icmp_echo_reply (c: connection, icmp: icmp_conn, id: count, seq: count, payload: string) {...}
- event icmp_unreachable (c: connection, icmp: icmp_conn, code: count, context: icmp_context) {...}
- event icmp_time_exceeded (c: connection, code: count, context: icmp_-context) {...}

The first event is triggered up with all the traffic sniffed from the net, however this event does not specify if a packet is generated in the sender side or in the recipient side, thus this information was obtained from the next events. Therefore, the first event will provide the payload and the next ones the side of the communication. In case one packet did not have the communication side we considered it to belong to the sender because is the most logic value.

Firstly, to know the direction of the packets the parameter is_orig in the events tcp_packet and udp_contents was used, if is_orig was true, the packet was sent by the sender side while if it is false was sent by the recipient side. Secondly, the packages captured by the event icmp_echo_request (request) were considered of the sender side and icmp_echo_reply (reply) of the recipient side. Finally, the events icmp_unreachable and icmp_time_exceeded were considered sender side packages, the default value, because they do not specify the packages direction.

An noteworthy problem we had with the event udp_contents that detects UDP packages was that it was never activated. Afterwards we found that the udp_contents event needed two variables to be activated. In the Listing 35 the two variables and the udp_contents event are shown.

Listing 35: The variables to define to use the event udp contents.

```
redef udp_content_deliver_all_orig = T;
redef udp_content_deliver_all_resp = T;
event udp_contents (u: connection, is_orig: bool,
                    contents: string) {...}
```

Each UDP or ICMP package has been considered to be a connection As a consequence, the payload was dumped in one direction file letting the opposite direction file empty. Therefore, 'connectionNumber.a' or (exclusive) 'connectionNumber.b', will be always equal to 'connectionNumber.c'.

The policy policy.bro saves all payload files in a folder called 'payload'. To verify that this script has generated correctly all payload files the number of connections and the number of payload files generated were compared. The number of payload needs to be 3 times more than number of connections.

Finally, all payload files' names were renamed to make them of the same length. In first place the assigned name was the connection number with its extensions were saved, for example, '1.a', '1.c', '1.c', '2.1', '2.b' and '2.c'; and then, adding week and day information they were converted to the following format: '71000001.a', '71000001.b', '71000001.c', '71000002.a', '71000002.b', and '71000002.c'. Where the first digit represents the number of the week (from 1 to 7), the second digit represent the day inside the week (from 1 to 5, from Monday to Friday), the 6 remainder digits refer to the connection numbers of the day and the last characters refer to the extension or direction.

In the renaming phase a problem arose, when the number of files inside the folder 'payload' was too big, the execution was cut due to some memory overload problem, that is to say, because the machine needed to keep in buffers too many names. To overcome this problem, instead of renaming all names in one run we did many runs of smaller packages of names dividing our whole name space by the extensions (*.a, *.b and *.c). As this was not enough, we also divided the space by the last digit (*0.*, *1.*, *2.*, *3.*, *4.*, *5.*, *6.*, *7.*, *8.*, *9.*) dividing the whole space in $3x10 = 30$ groups of names.

8 Labelling the file trafAld.list

In this section, the last step for generating gureKDDCup will be described, the labelling of connections (labelling.sh). In this step the labelled connections of the file tcpdump.list were matched with the unlabelled connections of the file trafAld.list to label them. The program that does this task is shown in Listing 36. The execution generates the file gureKDDCup.list which contains basic attributes, all attributes of KDDCup99 plus the connection type (class) and attached the payload information.

Listing 36: Labelling the file trafAld.list.

```
$ ./labelling.out tcpdump.list trafAld.list Week1 Monday
```

When two connections were matched after exchanging ports (in TCP and UDP connections) or exchanging IP addresses (in ICMP connections) we indicated it in a new attribute with value 1. We considered this information interesting for the future analysis.

If two connections were matched the label assigned to the connection gureKDDCup was the one indicated in tcpdump.list: 'normal' or the corresponding attack name. However, not matched connections were labelled as 'normal2' because the tcpdump.list's documentation says that all attacks are identified but no all normal connections. Therefore, not matched connections had to be normal, but as they were not matched we marked them with the digit 2: 'normal2'.

To validate the criteria used to generate the attributes and matching the connection we repeated all steps again and again. Mainly each program shown in Listing 37 were tweaked and executed again and again (in bro's case payload generation lines commented) until acceptable values were obtained and so the gureKDDCup was generated. Being the most critical script the policy.bro.

Listing 37: Labelling the file trafAld.list.

```
$ bro -r tcpdump policy.bro > policy.list
$ ./trafAld.out policy.list Week1 Monday
$ ./labelling.out tcpdump.list trafAld.list Week1 Monday
```

9 The execution of all parts

The database gureKDDCup needs to run all the steps continuously for all data, thus a script was written (kddcupEguna.sh). The main steps are the ones listed in the Listing 38.

Listing 38: The main lines of the script kddcupEguna.sh.

```
$ bro -r tcpdump policy.bro > policy.list
$ ./trafAld.out policy.list Week1 Monday
$ ./labelling.out tcpdump.list trafAld.list Week1 Monday
```

The input files are ordered by time to make the execution faster. In the following subsection some details about each step will be explained.

9.1 The field date and daytime of the file tcpdump.list

Analysing matched and unmatched connections we realized that some tcpdump.list connections were not in order according to the start-time. Therefore these connections were ordered. To that end these connections' start-time, which was provided by date and daytime, was converted to compact timestamp format. Ordering the connections, the labelling process could be optimized and thus reduce execution time. To order the connections of tcpdump.list a C program was generated, time.c that is shown in the Listing 39, to order all connections before starting the matching process.

Listing 39: The time.c program to convert time from date-daytime format to timestamp.

```
#include <stdio.h>
#include <time.h>

#define MaxToken 20
#define MaxLine 1000

int main(int argc, char **argv){
  FILE *finTcpdump, *foutTcpdump;
  char line[MaxLine];
  int tcpKop;
  //line info
  int konZenb, erasoaDa;
  char data[MaxToken], ordua[MaxToken],
       duration[MaxToken], service[MaxToken];
  char orig_p[MaxToken], resp_p[MaxToken], orig_h[MaxToken],
       resp_h[MaxToken], erasoa[MaxToken];
  int urt, hil, egu, ord, min, seg;
  int hasUnea;
  struct tm t;
  time_t time;

  if (argc != 2){
    printf("Deia: %s inTcpdump.list\n", argv[0]);
    return (1);
  }

  //read tcpdump.list, format it and save
  finTcpdump = fopen(argv[1], "r");
  tcpKop = 0;
  while(!feof(finTcpdump)){
    fgets(line, MaxLine, finTcpdump);
    sscanf(line, "%d %s %s %s %s %s %s %s %d %s\n",
           &konZenb,&data,&ordua,&duration,
           &service,&orig_p,&resp_p,
           &orig_h,&resp_h,&erasoaDa,&erasoa);
    orig_p[strlen(orig_p)] = '\0';
```

```
    resp_p[strlen(resp_p)] = '\0';
    erasoa[strlen(erasoa)] = '\0';
    sscanf(data, "%d/%d/%d",&hil,&egu,&urt);
    sscanf(ordua, "%d:%d:%d",&ord,&min,&seg);
    t.tm_year = urt - 1900;
    t.tm_mon = hil - 1;
    t.tm_mday = egu;
    t.tm_hour = ord;
    t.tm_min = min;
    t.tm_sec = seg;
    t.tm_isdst = 1;
    time = mktime(&t);
    hasUnea = (int)time;
    printf("%d %d %s %s %s %s %s %d %s\n",
           konZenb,hasUnea,duration,service,orig_p,resp_p,
           orig_h,resp_h,erasoaDa,erasoa);
    tcpKop = tcpKop + 1;
  }
  fclose(finTcpdump);
  tcpKop = tcpKop - 1; // to remove EOF (End of file) line
}
```

9.2 Identifier of connection

The identifier given by bro to connections goes from 1 to number of connections in a particular day. Therefore, to make the identifier unique, digits of the week and day were added generating an identifier of 8 digits, where the first digit indicated the week (Week1, Week2, Week3, Week4, Week5, Week6, Week7 represented as 1, 2, 3, 4, 5, 6, 7 respectively); and the second digit indicates the day inside the week (Monday, Tuesday, Wednesday, Thursday, Friday represented as 1, 2, 3, 4, 5 respectively). The script that does this work is shown in Listing 40. For example the identifiers in Week1 and Monday were transformed as follows:

- 1 to 11000001.
- ...
- 123456 to 11123456.

Listing 40: The konexioId.sh bash script to add week and day information to connection identifiers.

```
#!/bin/bash
# To identifiers of gureKddcup.list
# it includes the information of week and day.

echo $3","$4 > $3/$4/lineId
cp $1 $3/$4/tmpId
cat $3/$4/lineId $3/$4/tmpId |
awk 'BEGIN{FS=","
           firstLine=1}
  {if(firstLine==1){
     w=$1
     if(w=="Week1") week=1
     else if(w=="Week2") week=2
     else if(w=="Week3") week=3
     else if(w=="Week4") week=4
     else if(w=="Week5") week=5
     else if(w=="Week6") week=6
     else if(w=="Week7") week=7
     d=$2
     if(d=="Monday") day=1
     else if(d=="Tuesday") day=2
     else if(d=="Wednesday") day=3
```

```
    else if(d=="Thursday") day=4
    else if(d=="Friday") day=5
    firstLine=0
  }
  else{
    if(length($1)==6)
      print week""day""$1","$2","$3","$4","$5","
            $6","$7","$8","$9","$10","
            $11","$12","$13","$14","$15","
            $16","$17","$18","$19","$20","
            $21","$22","$23","$24","$25","
            $26","$27","$28","$29","$30","
            $31","$32","$33","$34","$35","
            $36","$37","$38","$39","$40","
            $41","$42","$43","$44","$45","
            $46","$47","$48","$49","$50
    else if(length($1)==5)
      print week""day"0"$1","$2","$3","$4","$5","
            $6","$7","$8","$9","$10","
            $11","$12","$13","$14","$15","
            $16","$17","$18","$19","$20","
            $21","$22","$23","$24","$25","
            $26","$27","$28","$29","$30","
            $31","$32","$33","$34","$35","
            $36","$37","$38","$39","$40","
            $41","$42","$43","$44","$45","
            $46","$47","$48","$49","$50
    else if(length($1)==4)
      print week""day"00"$1","$2","$3","$4","$5","
            $6","$7","$8","$9","$10","
            $11","$12","$13","$14","$15","
            $16","$17","$18","$19","$20","
            $21","$22","$23","$24","$25","
            $26","$27","$28","$29","$30","
            $31","$32","$33","$34","$35","
            $36","$37","$38","$39","$40","
            $41","$42","$43","$44","$45","
            $46","$47","$48","$49","$50
    else if(length($1)==3)
      print week""day"000"$1","$2","$3","$4","$5","
            $6","$7","$8","$9","$10","
            $11","$12","$13","$14","$15","
            $16","$17","$18","$19","$20","
            $21","$22","$23","$24","$25","
            $26","$27","$28","$29","$30","
            $31","$32","$33","$34","$35","
            $36","$37","$38","$39","$40","
            $41","$42","$43","$44","$45","
            $46","$47","$48","$49","$50
    else if(length($1)==2)
      print week""day"0000"$1","$2","$3","$4","$5","
            $6","$7","$8","$9","$10","
            $11","$12","$13","$14","$15","
            $16","$17","$18","$19","$20","
            $21","$22","$23","$24","$25","
            $26","$27","$28","$29","$30","
            $31","$32","$33","$34","$35","
            $36","$37","$38","$39","$40","
            $41","$42","$43","$44","$45","
            $46","$47","$48","$49","$50
    else if(length($1)==1)
      print week""day"00000"$1","$2","$3","$4","$5","
            $6","$7","$8","$9","$10","
            $11","$12","$13","$14","$15","
            $16","$17","$18","$19","$20","
            $21","$22","$23","$24","$25","
            $26","$27","$28","$29","$30","
            $31","$32","$33","$34","$35","
            $36","$37","$38","$39","$40","
            $41","$42","$43","$44","$45","
            $46","$47","$48","$49","$50
  }
}' > $2
```

```
rm $3/$4/tmpId
rm $3/$4/lineId
```

9.3 Whole day execution script: kddcupEguna.sh

The script kddcupEguna.sh (see Listing 41) starts from the tcpdump.pcap captured in a particular week and day and the labelled tcpdump.list and it creates a folder where saves the database gureKDDCup.list, all payload files, the log file emaitza.log and gureKddcup.est file which contains statistics of the created database.

Listing 41: The kddcupEguna.sh bash script that generates the gureKDDCup.list database of a given week and day.

```
#!/bin/sh
# the main script, it creates week/day/gurekddcup.list
# with its payloads

mkdir -p $1 # create week folder

# create day folder
if [ -d $1/$2 ]
then
  rm -rf $1/$2
fi
mkdir $1/$2
mkdir $1/$2/tmp

echo ""
echo "$1/$2"
echo "START:" `awk 'BEGIN{printf "\t%s",
                 strftime("%Y/%m/%d %H:%M:%S",systime())}'`
: $((hasi=`awk 'BEGIN{print systime()}'`))

# run bro
echo "  bro hasi: "
      `awk 'BEGIN{print strftime("%Y/%m/%d %H:%M:%S",systime())}'`
./broBakarrik.sh $1 $2 $4
echo "  bro amaitu: "
      `awk 'BEGIN{print strftime("%Y/%m/%d %H:%M:%S",systime())}'`

sort -n -t" " -k 1 $1/$2/policy.list > $1/$2/policySort.list

# create trafAld.list
echo "  Trafiko aldagaiak kalkulatzen hasi: "
      `awk 'BEGIN{print strftime("%Y/%m/%d %H:%M:%S",systime())}'`
./trafAld $1/$2/policySort.list $1 $2
#trafiko aldagaien estatistikak
./policyEstatistikak $1/$2/trafAld.list > $1/$2/trafAld.est
echo "  Trafiko aldagaiak kalkulatzen amaitu: "
      `awk 'BEGIN{print strftime("%Y/%m/%d %H:%M:%S",systime())}'`

cp $1/$2/trafAld.list $1/$2/trafAldLag.list
sort -n -t" " -k 1 $1/$2/trafAldLag.list > $1/$2/trafAld.list
rm $1/$2/trafAldLag.list

# order tcpdump.list
./time $3 > $1/$2/tcpdump.list
sort -n -k 2 -t" " $1/$2/tcpdump.list > $1/$2/tcpdumpSort.list

# label the connections
echo "  Konexioak katalogatzen hasi: "
      `awk 'BEGIN{print strftime("%Y/%m/%d %H:%M:%S",systime())}'`
./etiketatu $1/$2/tcpdumpSort.list $1/$2/trafAld.list $1 $2
sort -n $1/$2/ourKddcup.list > $1/$2/gureKddcupZuriune.list
echo "  Konexioak katalogatzen amaitu: "
      `awk 'BEGIN{print strftime("%Y/%m/%d %H:%M:%S",systime())}'`

# format the file as KDDCup99 does,
```

```
# convert gaps into colon;
# and convert misspelling: warzclient to warezclient
cat $1/$2/gureKddcupZuriune.list |
  awk 'BEGIN{FS=" "}
    {if($49=="warzclient")
       print $1","$2","$3","$4","$5","
             $6","$7","$8","$9","$10","
             $11","$12","$13","$14","$15","
             $16","$17","$18","$19","$20","
             $21","$22","$23","$24","$25","
             $26","$27","$28","$29","$30","
             $31","$32","$33","$34","$35","
             $36","$37","$38","$39","$40","
             $41","$42","$43","$44","$45","
             $46","$47","$48",warezclient,"$50
     else
       print $1","$2","$3","$4","$5","
             $6","$7","$8","$9","$10","
             $11","$12","$13","$14","$15","
             $16","$17","$18","$19","$20","
             $21","$22","$23","$24","$25","
             $26","$27","$28","$29","$30","
             $31","$32","$33","$34","$35","
             $36","$37","$38","$39","$40","
             $41","$42","$43","$44","$45","
             $46","$47","$48","$49","$50
    }' > $1/$2/gureKddcup.list

# Put the connections' identifier in WeekDayID format.
# 1 -> 71000001
./konexioId.sh $1/$2/gureKddcup.list $1/$2/gureKddcup.list $1 $2

# statistics
> $1/$2/emaitzak.log
echo "bro-n bakarrik: "
     `wc -l $1/$2/1ourKddcup.list ` >> $1/$2/emaitzak.log
echo "Detektatu ez diren konexioak (tcpdump-list-en bakarrik): "
     `wc -l $1/$2/2ourKddcup.list ` >> $1/$2/emaitzak.log
echo "Detektatu diren konexioak (bietan): "
     `wc -l $1/$2/3ourKddcup.list ` >> $1/$2/emaitzak.log

echo "Detektatu diren erasoak: " >> $1/$2/emaitzak.log
more $1/$2/3ourKddcup.list |
     awk '{maiz[$49]++}
     END{for(i in maiz)printf "%s ---> %d aldiz\n", i, maiz[i]}'
     >> $1/$2/emaitzak.log
echo "Detektatu ez diren erasoak: " >> $1/$2/emaitzak.log
more $1/$2/2ourKddcup.list |
     awk '{maiz[$11]++}
     END{for(i in maiz)printf "%s ---> %d aldiz\n", i, maiz[i]}'
     >> $1/$2/emaitzak.log

echo "END  :"
     `awk 'BEGIN{printf "\t%s",
           strftime("%Y/%m/%d %H:%M:%S",systime())}'`
: $((amaitu=`awk 'BEGIN{print systime()}'`))
: $((guztira = amaitu - hasi))
echo "PASS TIME: " `echo $guztira |
      awk '{print strftime("%Y %m %d %H %M %S",$1)}' |
      awk '{print $1-1970"/"$2-1"/"$3-1" "$4-1":"$5":"$6}'`
```

The process can be summarized in the following way:

First of all the bro policy is executed and the connections are ordered by their connection identifier because it is the same as ordering them by timestamp because bro treats packages of tcpdump.pcap sequentially in time assigning numbers an incrementally number to each connection.

Afterwards traffic variables and statistics of them are added to the previously computed attributes.

Later, once all KDDCup99 attributes are computed the type of the connection, the class is added.

Next, the 'warzclient' attack name was replaced to 'warezclient' to fix the typography inconsistency found in tcpdump.list. Besides, the field delimiter symbol is changed to comma (',').

Finally, the connection identifier, week and day information is added.

9.4 Validation of gureKDDCup

To verify that the criteria used to compute attributes were correct, the values of those attributes were compared with the values in kddcup10percent.data. This comparison is included in the script kddcupEguna.sh (see Listing 42). Therefore, iteratively new computation criteria were included to the bro policy and if needed to be executed it again, until acceptable statistical values were obtained. In the policy code, lines devoted to payload writing were commented to make the execution faster.

Listing 42: Call to kddcupEguna.sh bash script.

```
$./kddcupEguna.sh Week1 Monday 11tcpdump.list 11tcpdump
```

After executing the command indicated in Listing 42 the files emaitzak.log and gureKddcup.est in the folders indicated in the parameters should be analysed (in the example of Listing 42 the sub-folder would be Week1/Monday). To calculate the matching rates between gureKDDCup and kddcup10percent.data databases and statistics of attributes' values. Moreover, other files to analyse are generated: 1gureKddcup.list (unmatched bro connections), 2gureKddcup.list (unmatched tcpdump.list connections) and 3gureKddcup.list (matched connections); To do a deeper analysis of the unmatched connections. Likewise intermediate files can be analysed, specially the output of each step: tcpdumpSort.list, policySort.list, trafAld.list and the output gureKddcup.list.

Acknowledgements

This work was funded by the University of the Basque Country UPV/EHU (BAILab, grant UFI11/45); by the Department of Education, Universities and Research of the Basque Government (grant IT-395-10); and by the Ministry of Economy and Competitiveness of the Spanish Government and by the European Regional Development Fund - ERDF (eGovernAbility, grant TIN2014-52665-C2-1-R).

References

[1] Machine Learning Group at the University of Waikato. Weka (data mining software in java). `http://www.cs.waikato.ac.nz/ml/weka/` (accesed on 2015/04/13).

[2] Rebecca Bace and Peter Mell. Intrusion detection systems. Technical report, NIST Special Publication on Intrusion Detection Systems, 2001.

[3] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Salvatore Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. In D. Barbara and S. Jajodia, editors, *Applications of Data Mining in Computer Security*. Kluwer, 2002.

[4] Brian Fox for the GNU Project. Bash reference manual. `http://www.gnu.org/software/bash/manual/bashref.html` (accesed on 2015/04/13).

[5] Inc. Free Software Foundation. The gnu awk users guide. `http://www.gnu.org/software/gawk/manual/gawk.html` (accesed on 2015/04/13).

[6] Ibai Gurrutxaga. Darpa98 datu basearen azterketa (in basque). Technical report, University of Basque Country (UPV/EHU), 2006.

[7] Ibai Gurrutxaga. Kddcup99 datu basearen azterketa (in basque). Technical report, University of Basque Country (UPV/EHU), 2006.

[8] Eric Huss. The c library reference guide. `https://www-s.acm.illinois.edu/webmonkeys/book/c_guide/` (accesed on 2015/04/13).

[9] CA; International Computer Science Institute in Berkeley and IL. the National Center for Supercomputing Applications in Urbana-Champaign. The bro network security monitor. `https://www.bro.org/` (accesed on 2015/04/13).

[10] MIT Lincoln Laboratory. Darpa intrusion detection evaluation. `http://www.ll.mit.edu/ideval/data/` (accesed on 2015/04/13), 1998.

[11] Wenke Lee and Salvatore J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Trans. Inf. Syst. Secur.*, 3(4):227–261, November 2000.

[12] Urko Zurutuza Ortega. Sistemas de detección de intrusos (in spanish). Technical report, Department of Informatics in Mondragon Unibertsitatea, 2004.

A The most noteworthy program parts

A.1 darpa2gurekddcup.bro

The aim of the darpa2gurekddcup.bro policy is to create connections with its basic, intrinsic and content attributes. In addition it creates 3 files for each connection for storing the payloads transmitted in the connection. The command Listing 43 indicates how to execute the policy in bro.

Listing 43: Executing an user defined bro policy.

```
$ bro -r tcpdump.pcap policy.bro > policy.list
```

This script processes in average 20,000 connections per hour, taking into account that the captured tcpdump.pcap data was from Darpa98 and we processed it in a machine that had 1GHz processor with 1GB of RAM memory. The following script was used frequently to create connections without payloads because the process of writing payloads is very time consuming, thus the lines devoted to write the payloads were commented. Therefore, the validation of how we extracted content variables and iterations of changes done in the attribute definitions to make gureKDDCup resemble to KDDCup99 were done faster. The time needed to create 20,000 connections without payloads was 1 minute.

Listing 44: Bro policy to create (0) connection global attributes (1) intrinsic attributes and (2) content attributes.

```
global num_conn=0;

type konexioa: record {
  num_conn : count &default = 0;

  orig_h : string;
  resp_h : string;
  orig_p : string;
  resp_p : string;

  duration : string &default = "0";
  protokoloa : string &default = "tcp";
  service : string &default = "";
  flag : string &default = "OTH";
  src_bytes : string &default = "0";
  dst_bytes : string &default = "0";
  land : count &default = 0;
  wrong_fragment : count &default = 0;
  urg: count &default = 0;
  hot : count &default = 0;
  num_failed_logins: count &default = 0;
  logged_in : count &default = 0;
  num_compromised : count &default = 0;
  root_shell : count &default = 0;
  root_shell_num : count &default = 0;
  su_attempted : count &default = 0;
  num_root : count &default = 0;
  num_file_creations : count &default = 0;
  num_shells : count &default = 0;
  rootDa : bool &default = F;
  num_access_files : count &default = 0;
  num_outbound_cmds : count &default = 0;
  is_hot_login : count &default = 0;
  is_guest_login : count &default = 0;

  f1 : file;
  f2 : file;
  f3 : file;
```

```
  fitxSortuak : bool &default = F;

  payDu : bool &default = F;
  payload : string &default = "";

  isOrig : bool &default = T;
};

global konexioak: table[string, string, addr,
                        port, addr, port] of konexioa;
global konexioaktcp: table[string, addr, port, addr, port] of konexioa;

type byte: record {
  src_bytes : count &default = 0;
  dst_bytes : count &default = 0;
};

global byteak: table[string, addr, port, addr, port] of byte;

const conn_closed = { TCP_CLOSED, TCP_RESET };

function conn_state(c: connection, trans: string): string
{
  # get the connections state

  local os = c$orig$state;
  local rs = c$resp$state;

  local o_inactive = os == TCP_INACTIVE || os == TCP_PARTIAL;
  local r_inactive = rs == TCP_INACTIVE || rs == TCP_PARTIAL;

  if ( trans == "tcp" )
  {
    if ( rs == TCP_RESET )
    {
      if ( os == TCP_SYN_SENT || os == TCP_SYN_ACK_SENT ||
           (os == TCP_RESET && c$orig$size == 0 && c$resp$size == 0) )
        return "REJ";
      else if ( o_inactive )
        return "RSTRH";
      else
        return "RSTR";
    }
    else if ( os == TCP_RESET )
      return r_inactive ? "RSTOS0" : "RSTO";
    else if ( rs == TCP_CLOSED && os == TCP_CLOSED )
      return "SF";
    else if ( os == TCP_CLOSED )
      return r_inactive ? "SH" : "S2";
    else if ( rs == TCP_CLOSED )
      return o_inactive ? "SHR" : "S3";
    else if ( os == TCP_SYN_SENT && rs == TCP_INACTIVE )
      return "S0";
    else if ( os == TCP_ESTABLISHED && rs == TCP_ESTABLISHED )
      return "S1";
    else
      return "OTH";
  }

  else if ( trans == "udp" )
  {
    if ( os == UDP_ACTIVE )
      return rs == UDP_ACTIVE ? "SF" : "S0";
    else
      return rs == UDP_ACTIVE ? "SHR" : "OTH";
  }

  else if ( trans == "icmp" )
    if ( c$orig$size > 0 )
    {
      if ( c$resp$size > 0 )
        return "SF";
      else
        return "SH";
```

```
    }
    else if ( c$resp$size > 0 )
      return "SHR";
    else
      return "OTH";

    else
      return "OTH";
  }


########################## TCP ##############################

function record_connectionTCP(c: connection){
  local startTime : string = fmt ("%.6f", c$start_time-6*60min);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;
  if ([startTime, orig_h, orig_p, resp_h, resp_p] !in konexioaktcp)
  {
    local empty_record: konexioa;
    konexioaktcp[startTime, orig_h, orig_p,
                 resp_h, resp_p] = empty_record;

    ++num_conn;
    konexioaktcp[startTime, orig_h, orig_p, resp_h, resp_p]$num_conn =
      num_conn;
  }
  konexioaktcp[startTime, orig_h, orig_p, resp_h, resp_p]$duration =
    fmt("%.6f", c$duration);
  konexioaktcp[startTime, orig_h, orig_p, resp_h, resp_p]$flag =
    conn_state(c, "tcp");
  konexioaktcp[startTime, orig_h, orig_p, resp_h, resp_p]$src_bytes =
    fmt("%d", c$orig$size);
  konexioaktcp[startTime, orig_h, orig_p, resp_h, resp_p]$dst_bytes =
    fmt("%d", c$resp$size);
}

event new_connection(c: connection){
  local trans : string = fmt("%s",
                             get_port_transport_proto(c$id$resp_p));
  if(trans=="tcp"){
    record_connectionTCP(c);
  }
}

event connection_state_remove(c: connection){
  local trans : string = fmt("%s",
                             get_port_transport_proto(c$id$resp_p));
  if (trans == "tcp"){
    record_connectionTCP(c);
  }
}

event tcp_packet (c: connection, is_orig: bool, flags: string,
    seq: count, ack: count, len: count, payload: string){
  local startTime : string = fmt ("%.6f", c$start_time-6*60min);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;

  record_connectionTCP(c);

  if (konexioaktcp[startTime, orig_h, orig_p,
                   resp_h, resp_p]$fitxSortuak
       == F){
    local izen1 : string = fmt("payload/%d.a",
      konexioaktcp[startTime, orig_h, orig_p, resp_h, resp_p]$num_conn);
    local izen2 : string = fmt("payload/%d.b",
      konexioaktcp[startTime, orig_h, orig_p, resp_h, resp_p]$num_conn);
    local izen3 : string = fmt("payload/%d.c",
      konexioaktcp[startTime, orig_h, orig_p, resp_h, resp_p]$num_conn);
```

```
    konexioaktcp[startTime, orig_h, orig_p, resp_h, resp_p]$f1 =
      open(izen1);
    konexioaktcp[startTime, orig_h, orig_p, resp_h, resp_p]$f2 =
      open(izen2);
    konexioaktcp[startTime, orig_h, orig_p, resp_h, resp_p]$f3 =
      open(izen3);
     konexioaktcp[startTime, orig_h, orig_p,
       resp_h, resp_p]$fitxSortuak = T;
  }

  if (is_orig)
    print konexioaktcp[startTime, orig_h, orig_p, resp_h, resp_p]$f1,
      payload;
  else
    print konexioaktcp[startTime, orig_h, orig_p, resp_h, resp_p]$f2,
      payload;

  print konexioaktcp[startTime, orig_h, orig_p, resp_h, resp_p]$f3,
    payload;
  konexioaktcp[startTime, orig_h, orig_p, resp_h, resp_p]$payDu = T;
}


################ UDP ########################

redef udp_content_deliver_all_orig = T;
redef udp_content_deliver_all_resp = T;

event udp_contents(u: connection, is_orig: bool, contents: string)
{
  local startTime : string = fmt ("%.6f", u$start_time-6*60min);
  local duration : string = fmt ("%.6f", u$duration);
  local orig_h : addr = u$id$orig_h;
  local resp_h : addr = u$id$resp_h;
  local orig_p : port = u$id$orig_p;
  local resp_p : port = u$id$resp_p;
  if ([startTime, duration, orig_h, orig_p, resp_h, resp_p]
       !in konexioak)
  {
    local empty_record: konexioa;
    konexioak[startTime, duration, orig_h, orig_p, resp_h, resp_p] =
      empty_record;

    ++num_conn;
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_conn = num_conn;
  }

  if ([startTime, orig_h, orig_p, resp_h, resp_p] !in byteak){
    local empty_byte: byte;
    byteak[startTime, orig_h, orig_p, resp_h, resp_p] = empty_byte;
  }

  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$protokoloa = "udp";
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$flag = conn_state(u, "udp");

  if (is_orig){
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$isOrig = T;

    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$src_bytes =
        fmt("%d", u$orig$size-byteak[startTime, orig_h, orig_p,
          resp_h, resp_p]$src_bytes);
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$dst_bytes = "0";
    byteak[startTime, orig_h, orig_p,
      resp_h, resp_p]$src_bytes = u$orig$size;

    # Payload
    local izena11 : string = fmt("payload/%d.a",
      konexioak[startTime, duration, orig_h, orig_p,
```

```
        resp_h, resp_p]$num_conn);
    local f11 : file = open(izena11);
    print f11, contents;
    close(f11);
    local izena12 : string = fmt("payload/%d.b",
      konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$num_conn);
    local f12 : file = open(izena12);
    close(f12);
    local izena13 : string = fmt("payload/%d.c",
      konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$num_conn);
    local f13 : file = open(izena13);
    print f13, contents;
    close(f13);

    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$payDu = T;
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$payload = "";
  }

  else{
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$isOrig = F;

    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$dst_bytes = fmt("%d",
        u$resp$size-byteak[startTime, orig_h, orig_p,
          resp_h, resp_p]$dst_bytes);
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$src_bytes = "0";
    byteak[startTime, orig_h, orig_p,
      resp_h, resp_p]$dst_bytes = u$resp$size;

    # Payload
    local izena21 : string = fmt("payload/%d.a",
      konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$num_conn);
    local f21 : file = open(izena21);
    close(f21);
    local izena22 : string = fmt("payload/%d.b",
      konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$num_conn);
    local f22 : file = open(izena22);
    print f22, contents;
    close(f22);
    local izena23 : string = fmt("payload/%d.c",
      konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$num_conn);
    local f23 : file = open(izena23);
    print f23, contents;
    close(f23);
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$payDu = T;
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$payload = "";
  }
}


################## ICMP ########################

function record_connectionICMP(c: connection){
  local startTime : string = fmt ("%.6f", c$start_time-6*60min);
  local duration : string = fmt ("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;
  if ([startTime, duration, orig_h, orig_p, resp_h, resp_p]
      !in konexioak)
  {
    local empty_record: konexioa;
```

```
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p] = empty_record;

    ++num_conn;
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_conn = num_conn;
  }
}

event icmp_echo_request(c: connection, icmp: icmp_conn,
  id: count, seq: count, payload: string)
{
  local startTime : string = fmt ("%.6f", c$start_time-6*60min);
  local duration : string = fmt ("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;

  record_connectionICMP(c);

  if ([startTime, orig_h, orig_p, resp_h, resp_p] !in byteak){
    local empty_byte: byte;
    byteak[startTime, orig_h, orig_p, resp_h, resp_p] = empty_byte;
  }

  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$isOrig = T;
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$protokoloa = "icmp";
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$flag = conn_state(c, "icmp");

  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$src_bytes = fmt("%d",
      c$orig$size-byteak[startTime, orig_h, orig_p,
        resp_h, resp_p]$src_bytes);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$dst_bytes = "0";
  byteak[startTime, orig_h, orig_p,
    resp_h, resp_p]$src_bytes = c$orig$size;

  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$orig_p = fmt("%d", icmp$itype);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$resp_p = fmt("%d", icmp$icode);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$orig_h = fmt("%s", icmp$orig_h);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$resp_h = fmt("%s", icmp$resp_h);

  # Payload
  local izena1 : string = fmt("payload/%d.a",
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_conn);
  local f1 : file = open(izena1);
  print f1, payload;
  close(f1);
  local izena2 : string = fmt("payload/%d.b",
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_conn);
  local f2 : file = open(izena2);
  close(f2);
  local izena3 : string = fmt("payload/%d.c",
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_conn);
  local f3 : file = open(izena3);
  print f3, payload;
  close(f3);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$payDu = T;
}

event icmp_echo_reply(c: connection, icmp: icmp_conn,
```

```
  id: count, seq: count, payload: string)
{
  local startTime : string = fmt ("%.6f", c$start_time-6*60min);
  local duration : string = fmt ("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;

  record_connectionICMP(c);

  if ([startTime, orig_h, orig_p, resp_h, resp_p] !in byteak){
    local empty_byte: byte;
    byteak[startTime, orig_h, orig_p, resp_h, resp_p] = empty_byte;
  }

  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$isOrig = F;
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$protokoloa = "icmp";
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$flag = conn_state(c, "icmp");

  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$dst_bytes = fmt("%d",
      c$resp$size-byteak[startTime, orig_h, orig_p,
        resp_h, resp_p]$dst_bytes);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$src_bytes = "0";
  byteak[startTime, orig_h, orig_p,
    resp_h, resp_p]$dst_bytes = c$resp$size;

  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$orig_p = fmt("%d", icmp$itype);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$resp_p = fmt("%d", icmp$icode);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$orig_h = fmt("%s", icmp$orig_h);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$resp_h = fmt("%s", icmp$resp_h);

  # Payload
  local izena1 : string = fmt("payload/%d.a",
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_conn);
  local f1 : file = open(izena1);
  close(f1);
  local izena2 : string = fmt("payload/%d.b",
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_conn);
  local f2 : file = open(izena2);
  print f2, payload;
  close(f2);
  local izena3 : string = fmt("payload/%d.c",
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_conn);
  local f3 : file = open(izena3);
  print f3, payload;
  close(f3);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$payDu = T;
}

event icmp_unreachable(c: connection, icmp: icmp_conn,
  code: count, context: icmp_context)
{
  local startTime : string = fmt ("%.6f", c$start_time-6*60min);
  local duration : string = fmt ("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;

  record_connectionICMP(c);
```

```
  if ([startTime, orig_h, orig_p, resp_h, resp_p] !in byteak){
    local empty_byte: byte;
    byteak[startTime, orig_h, orig_p, resp_h, resp_p] = empty_byte;
  }

  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$protokoloa = "icmp";
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$flag = conn_state(c, "icmp");

  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$src_bytes = fmt("%d",
      c$orig$size-byteak[startTime, orig_h, orig_p,
        resp_h, resp_p]$src_bytes);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$dst_bytes = "0";
  byteak[startTime, orig_h, orig_p,
    resp_h, resp_p]$src_bytes = c$orig$size;

  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$orig_p = fmt("%d", icmp$itype);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$resp_p = fmt("%d", icmp$icode);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$orig_h = fmt("%s", icmp$orig_h);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$resp_h = fmt("%s", icmp$resp_h);

  # Payload
  local izena1 : string = fmt("payload/%d.a",
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_conn);
  local f1 : file = open(izena1);
  print f1, context;
  close(f1);
  local izena2 : string = fmt("payload/%d.b",
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_conn);
  local f2 : file = open(izena2);
  close(f2);
  local izena3 : string = fmt("payload/%d.c",
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_conn);
  local f3 : file = open(izena3);
  print f3, context;
  close(f3);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$payDu = T;
}

event icmp_time_exceeded(c: connection, icmp: icmp_conn,
  code: count, context: icmp_context)
{
  local startTime : string = fmt ("%.6f", c$start_time-6*60min);
  local duration : string = fmt ("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;

  record_connectionICMP(c);

  if ([startTime, orig_h, orig_p, resp_h, resp_p] !in byteak){
    local empty_byte: byte;
    byteak[startTime, orig_h, orig_p, resp_h, resp_p] = empty_byte;
  }

  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$protokoloa = "icmp";
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$flag = conn_state(c, "icmp");

  konexioak[startTime, duration, orig_h, orig_p,
```

```
    resp_h, resp_p]$src_bytes = fmt("%d", c$orig$size);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$dst_bytes = fmt("%d", c$resp$size);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$src_bytes = fmt("%d",
      c$orig$size-byteak[startTime, orig_h, orig_p,
        resp_h, resp_p]$src_bytes);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$dst_bytes = "0";
  byteak[startTime, orig_h, orig_p,
    resp_h, resp_p]$src_bytes = c$orig$size;

  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$orig_p = fmt("%d", icmp$itype);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$resp_p = fmt("%d", icmp$icode);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$orig_h = fmt("%s", icmp$orig_h);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$resp_h = fmt("%s", icmp$resp_h);

  # Payload
  local izena1 : string = fmt("payload/%d.a",
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_conn);
  local f1 : file = open(izena1);
  print f1, context;
  close(f1);
  local izena2 : string = fmt("payload/%d.b",
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_conn);
  local f2 : file = open(izena2);
  close(f2);
  local izena3 : string = fmt("payload/%d.c",
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_conn);
  local f3 : file = open(izena3);
  print f3, context;
  close(f3);
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$payDu = T;
  konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$payload = "";
}

# At the end write all the connections
event bro_done()
{
  local land : count;
  local root_shell : count;
  local is_hot_login : count;
  local is_guest_login : count;

  # TCP
  for ([startTimet, orig_ht, orig_pt, resp_ht, resp_pt]
       in konexioaktcp){
    land = (orig_pt == resp_pt && orig_ht == resp_ht) ? 1 : 0;
    root_shell = (konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$root_shell_num>0) ? 1 : 0;
    is_hot_login = (konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$is_hot_login > 0) ? 1 : 0;
    is_guest_login = (konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$is_guest_login > 0) ? 1 : 0;
    print fmt("%d %s %d %d %s %s %s %s %d %s
%s %s %d %d %d %d %d %d %d %d
%d %d %d %d %d %d %d %d %d",
      konexioaktcp[startTimet, orig_ht, orig_pt,
                   resp_ht, resp_pt]$num_conn,
      startTimet, orig_pt, resp_pt, orig_ht, resp_ht,
      konexioaktcp[startTimet, orig_ht, orig_pt,
        resp_ht, resp_pt]$duration, "tcp", resp_pt,
      konexioaktcp[startTimet, orig_ht, orig_pt, resp_ht, resp_pt]$flag,
      konexioaktcp[startTimet, orig_ht, orig_pt,
        resp_ht, resp_pt]$src_bytes,
```

```
    konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$dst_bytes, land,
    konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$wrong_fragment,
    konexioaktcp[startTimet, orig_ht, orig_pt, resp_ht, resp_pt]$urg,
    konexioaktcp[startTimet, orig_ht, orig_pt, resp_ht, resp_pt]$hot,
    konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$num_failed_logins,
    konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$logged_in,
    konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$num_compromised, root_shell,
    konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$su_attempted,
    konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$num_root,
    konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$num_file_creations,
    konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$num_shells,
    konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$num_access_files,
    konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$num_outbound_cmds,
    is_hot_login, is_guest_login);
  if (konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$payDu){
    close(konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$f1);
    close(konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$f2);
    close(konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$f3);
  }
  else {
    # Payload
    local izenat1 : string = fmt("payload/%d.a",
      konexioaktcp[startTimet, orig_ht, orig_pt,
        resp_ht, resp_pt]$num_conn);
    local ft1 : file = open(izenat1);
    print ft1, konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$payload;
    close(ft1);
    local izenat2 : string = fmt("payload/%d.b",
      konexioaktcp[startTimet, orig_ht, orig_pt,
        resp_ht, resp_pt]$num_conn);
    local ft2 : file = open(izenat2);
    close(ft2);
    local izenat3 : string = fmt("payload/%d.c",
      konexioaktcp[startTimet, orig_ht, orig_pt,
        resp_ht, resp_pt]$num_conn);
    local ft3 : file = open(izenat3);
    print ft3, konexioaktcp[startTimet, orig_ht, orig_pt,
      resp_ht, resp_pt]$payload;
    close(ft3);
  }
}

# UDP & ICMP
for ([startTime, duration, orig_h, orig_p, resp_h, resp_p] in
      konexioak)
{
  land = (orig_p == resp_p && orig_h == resp_h) ? 1 : 0;
  root_shell = (konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$root_shell_num>0) ? 1 : 0;
  is_hot_login = (konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$is_hot_login > 0) ? 1 : 0;
  is_guest_login = (konexioak[startTime, duration, orig_h, orig_p,
    resp_h, resp_p]$is_guest_login > 0) ? 1 : 0;

  # icmp
  if(konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$protokoloa == "icmp"){
    print fmt("%d %s %s %s %s %s %s %s %s %s
```

```
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%s␣%s␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d",
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_conn, startTime,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$orig_p,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$resp_p,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$orig_h,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$resp_h,
        duration, "icmp",
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$orig_p,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$flag,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$src_bytes,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$dst_bytes, land,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$wrong_fragment,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$urg,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$hot,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_failed_logins,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$logged_in,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_compromised,
        root_shell,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$su_attempted,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_root,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_file_creations,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_shells,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_access_files,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_outbound_cmds,
        is_hot_login, is_guest_login);
    }

    # udp
    else if (konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$protokoloa == "udp"){
      print fmt("%d␣%s␣%d␣%d␣%s␣%s␣%s␣%s␣%d␣%s
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%s␣%s␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d",
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_conn, startTime,
        orig_p, resp_p, orig_h, resp_h, duration, "udp", resp_p,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$flag,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$src_bytes,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$dst_bytes, land,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$wrong_fragment,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$urg,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$hot,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_failed_logins,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$logged_in,
```

```
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_compromised, root_shell,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$su_attempted,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_root,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_file_creations,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_shells,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_access_files,
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_outbound_cmds,
        is_hot_login, is_guest_login);
    }

    if(konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$payDu==F){
      # Payload
      local izena1 : string = fmt("payload/%d.a",
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_conn);
      local f1 : file = open(izena1);
      print f1, konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$payload;
      close(f1);
      local izena2 : string = fmt("payload/%d.b",
      konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$num_conn);
      local f2 : file = open(izena2);
      close(f2);
      local izena3 : string = fmt("payload/%d.c",
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$num_conn);
      local f3 : file = open(izena3);
      print f3, konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$payload;
      close(f3);
    }
  }
}


################ PAYLOAD (TCP,UDP,ICMP) ########################

event packet_contents(c: connection, contents: string)
{
  local startTime : string = fmt ("%.6f", c$start_time-6*60min);
  local duration : string = fmt ("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;

  local trans : string = fmt("%s",
                          get_port_transport_proto(c$id$resp_p));
  local empty_record : konexioa;

  if (trans=="tcp"){
    if ([startTime, orig_h, orig_p, resp_h, resp_p] !in
          konexioaktcp){
      konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p] = empty_record;

      ++num_conn;
      konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$num_conn = num_conn;
      konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$payload = contents;
    }
    else{
      if (konexioaktcp[startTime, orig_h, orig_p,
            resp_h, resp_p]$payDu==F){
```

```
        konexioaktcp[startTime, orig_h, orig_p,
          resp_h, resp_p]$payload = fmt("%s\n%s",
            konexioak[startTime, duration, orig_h, orig_p,
              resp_h, resp_p]$payload, contents);
      }
    }
    konexioaktcp[startTime, orig_h, orig_p,
      resp_h, resp_p]$duration = fmt("%.6f", c$duration);
    konexioaktcp[startTime, orig_h, orig_p,
      resp_h, resp_p]$flag = conn_state(c, "tcp");
    konexioaktcp[startTime, orig_h, orig_p,
      resp_h, resp_p]$src_bytes = fmt("%s", c$orig$size);
    konexioaktcp[startTime, orig_h, orig_p,
      resp_h, resp_p]$dst_bytes = fmt("%s", c$resp$size);
  }
  else{
    if ([startTime, duration, orig_h, orig_p, resp_h, resp_p] !in
        konexioak)
    {
      konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p] = empty_record;

      ++num_conn;
      konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$num_conn = num_conn;
    }

    if ([startTime, orig_h, orig_p, resp_h, resp_p] !in byteak){
      local empty_byte: byte;
      byteak[startTime, orig_h, orig_p, resp_h, resp_p] = empty_byte;
    }

    if(konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$payDu==F){
      if (trans=="icmp"){
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$protokoloa = "icmp";
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$flag = conn_state(c, "icmp");
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$orig_p = fmt("%d", c$id$orig_p);
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$resp_p = fmt("%d", c$id$resp_p);
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$orig_h = fmt("%s", c$id$orig_h);
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$resp_h = fmt("%s", c$id$resp_h);
      }
      else if (trans=="udp"){
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$protokoloa = "udp";
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$flag = conn_state(c, "udp");
      }

      if(konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$isOrig){
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$src_bytes = fmt("%d",
            c$orig$size-byteak[startTime, orig_h, orig_p,
              resp_h, resp_p]$src_bytes);
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$dst_bytes = "0";
        byteak[startTime, orig_h, orig_p,
          resp_h, resp_p]$src_bytes = c$orig$size;
      }
      else{
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$src_bytes = "0";
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$dst_bytes = fmt("%d",
            c$orig$size-byteak[startTime, orig_h, orig_p,
              resp_h, resp_p]$dst_bytes);
        byteak[startTime, orig_h, orig_p,
```

```
          resp_h, resp_p]$dst_bytes = c$resp$size;
      }
      konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$payload = contents;
    }
  }
}


############### GAINERAKO ALDAGAIAK LORTU ###################

function konexioaSortu(c: connection){
  local trans : string = fmt("%s",
                        get_port_transport_proto(c$id$resp_p));
  local startTime : string = fmt("%.6f", c$start_time-6*60min);
  local duration : string = fmt("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;

  local empty_record : konexioa;
  if (trans=="tcp"){
    if ([startTime, orig_h, orig_p, resp_h, resp_p] !in konexioaktcp){
      konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p] = empty_record;

      ++num_conn;
      konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$num_conn = num_conn;
      konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$duration = fmt("%.6f", c$duration);
      konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$flag = conn_state(c, "tcp");
      konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$src_bytes = fmt("%d", c$orig$size);
      konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$dst_bytes = fmt("%d", c$resp$size);
    }
  }
  else{
    if ([startTime, duration, orig_h, orig_p, resp_h, resp_p] !in
          konexioak)
    {
      konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p] = empty_record;

      ++num_conn;
      konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$num_conn = num_conn;
    }

    if(konexioak[startTime, duration, orig_h, orig_p,
         resp_h, resp_p]$payDu==F){
      if (trans=="icmp"){
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$protokoloa = "icmp";
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$flag = conn_state(c, "icmp");
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$orig_p = fmt("%d", c$id$orig_p);
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$resp_p = fmt("%d", c$id$resp_p);
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$orig_h = fmt("%s", c$id$orig_h);
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$resp_h = fmt("%s", c$id$resp_h);
      }
      else if (trans=="udp"){
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$protokoloa = "udp";
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$flag = conn_state(c, "udp");
      }
```

```
    }
  }
}

# logged_in
function logged_in(c: connection){
  local trans : string = fmt("%s",
                        get_port_transport_proto(c$id$resp_p));
  local startTime : string = fmt("%.6f", c$start_time-6*60min);
  local duration : string = fmt("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;

  konexioaSortu(c);

  if(trans=="tcp")
    konexioaktcp[startTime, orig_h, orig_p, resp_h, resp_p]$logged_in=1;
  else
    konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$logged_in=1;
}

#host, guest
function inkrHotGuest(c: connection, username: string){
  local trans : string = fmt("%s",
                        get_port_transport_proto(c$id$resp_p));
  local startTime : string = fmt("%.6f", c$start_time-6*60min);
  local duration : string = fmt("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;
  if (/[gG][uU][eE][sS][tT]/ in username ||
      /[aA][nN][oO][nN][yY][mM][oO][uU][sS]/ in username ||
      /[vV][iI][sS][iI][tT][oO][rR]/ in username ||
      /[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-zA-Z0-9]+/ in username){
    if(trans=="tcp")
      ++konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$is_guest_login;
    else
      ++konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$is_guest_login;
  }
  if (/[rR][oO][oO][tT]/ in username ||
      /[aA][dD][mM]/ in username){
    if(trans=="tcp")
      ++konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$is_hot_login;
    else
      ++konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$is_hot_login;
  }
}

#num_failed_logins
function num_failed_logins(c: connection){
  local trans : string = fmt("%s",
                        get_port_transport_proto(c$id$resp_p));
  local startTime : string = fmt("%.6f", c$start_time-6*60min);
  local duration : string = fmt("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;
  if(trans=="tcp")
    ++konexioaktcp[startTime, orig_h, orig_p,
      resp_h, resp_p]$num_failed_logins;
  else
    ++konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_failed_logins;
}
```

```
#num_compromised
function not_found(c: connection, line: string){
  local trans : string = fmt("%s",
                        get_port_transport_proto(c$id$resp_p));
  local startTime : string = fmt("%.6f", c$start_time-6*60min);
  local duration : string = fmt("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;
  if (/[nN][oO][tT] [fF][oO][uU][nN][dD]/ in line){
   if(trans=="tcp")
    ++konexioaktcp[startTime, orig_h, orig_p,
      resp_h, resp_p]$num_compromised;
   else
    ++konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_compromised;
  }
}

#num_root
function num_root(c: connection, user: string){
  local trans : string = fmt("%s",
                        get_port_transport_proto(c$id$resp_p));
  local startTime : string = fmt("%.6f", c$start_time-6*60min);
  local duration : string = fmt("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;
  if(/[rR][oO][oO][tT]/ in user){
    if(trans=="tcp")
      ++konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$num_root;
    else
      ++konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$num_root;
  }
}

#num_rootInkr
function num_rootInkr(c: connection){
  local trans : string = fmt("%s",
                        get_port_transport_proto(c$id$resp_p));
  local startTime : string = fmt("%.6f", c$start_time-6*60min);
  local duration : string = fmt("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;
  if(trans=="tcp")
    ++konexioaktcp[startTime, orig_h, orig_p,
      resp_h, resp_p]$num_root;
  else
    ++konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_root;
}

#root_shell_num
function root_shell(c: connection, user: string){
  local trans : string = fmt("%s",
                        get_port_transport_proto(c$id$resp_p));
  local startTime : string = fmt("%.6f", c$start_time-6*60min);
  local duration : string = fmt("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;
  if(/[rR][oO][oO][tT]/ in user){
    if(trans=="tcp")
      ++konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$root_shell_num;
    else
      ++konexioak[startTime, duration, orig_h, orig_p,
```

```
        resp_h, resp_p]$root_shell_num;
  }
}

#num_file_creations
function num_file_creations(c: connection){
  local trans : string = fmt("%s",
                        get_port_transport_proto(c$id$resp_p));
  local startTime : string = fmt("%.6f", c$start_time-6*60min);
  local duration : string = fmt("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;
  if(trans=="tcp")
    ++konexioaktcp[startTime, orig_h, orig_p,
      resp_h, resp_p]$num_file_creations;
  else
    ++konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$num_file_creations;
}

#hot
function hot(c: connection){
  local trans : string = fmt("%s",
                        get_port_transport_proto(c$id$resp_p));
  local startTime : string = fmt("%.6f", c$start_time-6*60min);
  local duration : string = fmt("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;
  if(trans=="tcp")
    ++konexioaktcp[startTime, orig_h, orig_p,
      resp_h, resp_p]$hot;
  else
    ++konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$hot;
}

#root_shell_num
function root_shell_num(c: connection){
  local trans : string = fmt("%s",
                        get_port_transport_proto(c$id$resp_p));
  local startTime : string = fmt("%.6f", c$start_time-6*60min);
  local duration : string = fmt("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;
  if(trans=="tcp")
    ++konexioaktcp[startTime, orig_h, orig_p,
      resp_h, resp_p]$root_shell_num;
  else
    ++konexioak[startTime, duration, orig_h, orig_p,
      resp_h, resp_p]$root_shell_num;
}

#outbound
function outbound (c: connection, command: string){
  local trans : string = fmt("%s",
                        get_port_transport_proto(c$id$resp_p));
  local startTime : string = fmt("%.6f", c$start_time-6*60min);
  local duration : string = fmt("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;
  if (/[oO][uU][tT][bB][oO][uU][nN][dD]/ in command)
    if(trans=="tcp")
      ++konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$num_outbound_cmds;
    else
      ++konexioak[startTime, duration, orig_h, orig_p,
```

```
        resp_h, resp_p]$num_outbound_cmds;
}

event new_packet (c: connection, p: pkt_hdr){
 local trans : string = fmt("%s",
                        get_port_transport_proto(c$id$resp_p));
  local startTime : string = fmt("%.6f", c$start_time-6*60min);
  local duration : string = fmt("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;

  konexioaSortu(c);

  #urgent
  if (get_port_transport_proto(c$id$resp_p) == tcp){
    if (p$tcp$flags >= 32){
      if (trans=="tcp")
        ++konexioaktcp[startTime, orig_h, orig_p,
          resp_h, resp_p]$urg;
      else
        ++konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$urg;
    }
  }
}

event login_input_line(c: connection, line: string){
  local trans : string = fmt("%s",
                         get_port_transport_proto(c$id$resp_p));
  local startTime : string = fmt("%.6f", c$start_time-6*60min);
  local duration : string = fmt("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;

  konexioaSortu(c);

  # su_attempted
  if ( /su -/ in line){
    if(trans=="tcp")
      ++konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$su_attempted;
    else
      ++konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$su_attempted;
  }
  # num_access_files
  if ( /cat \// in line ||
     ((/vi / in line) && !(/.tex/ in line))||
     ((/rm / in line) && !(/.tex/ in line)) ){
    if(trans=="tcp")
      ++konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$num_access_files;
    else
      ++konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$num_access_files;
  }

  # is_hot_login
  if (/su - root/ in line ||
      /^[0-9]*root/ in line){
    if(trans=="tcp")
      ++konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$is_hot_login;
    else
      ++konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$is_hot_login;
  }

  # hot
  # into system directories
```

```
  if ( (/cd / in line) && !(/dir/ in line)){
    hot(c);
  }
  # create programs
  if (/gcc / in line ||
      /g++ / in line){
    num_file_creations(c);
    hot(c);
  }
  # run programs
  if(/^\.\// in line){
    hot(c);
  }
  if(/^\/tmp\/[0-9]+/ in line){
    hot(c);
  }

  if(/a.out/ in line ||
     /auditd/ in line ||
     /automountd/ in line ||
     /cron/ in line ||
     /find/ in line ||
     /fsck/ in line ||
     /ftp/ in line ||
     /in.comsat/ in line ||
     /inetd/ in line ||
     /in.ftpd/ in line ||
     /init/ in line ||
     /in.telnetd/ in line ||
     /kerbd/ in line ||
     /keyserv/ in line ||
     /lockd/ in line ||
     /login/ in line ||
     /lp.cat/ in line ||
     /lpNet/ in line ||
     /lpsched/ in line ||
     /lp.tell/ in line ||
     /lynx/ in line ||
     /mail/ in line ||
     /man/ in line ||
     /mlp/ in line ||
     /more/ in line ||
     /netscape/ in line ||
     /nscd/ in line ||
     /primes/ in line ||
     /sendmail/ in line ||
     /sh/ in line ||
     /sleep/ in line ||
     /sshd/ in line ||
     /statd/ in line ||
     /syslogd/ in line ||
     /tcsh/ in line ||
     /telnet/ in line ||
     /tex/ in line ||
     /top/ in line ||
     /ttymon/ in line ||
     /vi/ in line ||
     /vold/ in line ||
     /xntpd/ in line){
    hot(c);
  }
  # num_file_creations
  if(/cp[ \t]+/ in line ||
     /mv[ \t]+/ in line ||
     /cat >/ in line){
    num_file_creations(c);
  }

  inkrHotGuest(c,line);
}

event login_output_line(c: connection, line: string){
  local trans : string = fmt("%s",
                             get_port_transport_proto(c$id$resp_p));
```

```
  local startTime : string = fmt("%.6f", c$start_time-6*60min);
  local duration : string = fmt("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;

  konexioaSortu(c);

  # num_failed_logins
  if (/Login incorrect/ in line){
    num_failed_logins(c);
  }

  # num_root
  if (/^root@/ in line){
    if(trans=="tcp")
      ++konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$num_root;
    else
      ++konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$num_root;
    root_shell_num(c);
  }

  if (/su - root/ in line){
    if(trans=="tcp")
      konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$rootDa = T;
    else
      konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$rootDa = T;
  }
  if (/^Sun Microsystems/ in line){
    if(trans=="tcp"){
      if (konexioaktcp[startTime, orig_h, orig_p,
            resp_h, resp_p]$rootDa){
        ++konexioaktcp[startTime, orig_h, orig_p,
          resp_h, resp_p]$root_shell_num;
        konexioaktcp[startTime, orig_h, orig_p,
          resp_h, resp_p]$rootDa = F;
      }
    }
    else{
      if (konexioak[startTime, duration, orig_h, orig_p,
            resp_h, resp_p]$rootDa){
        ++konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$root_shell_num;
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$rootDa = F;
      }
    }
  }

  # num_shells
  if (/^Last login:/ in line){
    if(trans=="tcp")
      ++konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$num_shells;
    else
      ++konexioak[startTime, duration, orig_h, orig_p,
        resp_h, resp_p]$num_shells;
  }

  # num_file_creations
  if(/\[New file\]/ in line)
    num_file_creations(c);

  not_found(c,line);
}

# wrong login
event login_failure(c: connection, user: string,
      client_user: string, password: string, line: string){
```

```
  konexioaSortu(c);
  num_failed_logins(c);
}

# correct login
event login_success(c: connection, user: string,
      client_user: string, password: string, line: string){
  konexioaSortu(c);
  logged_in(c);
  inkrHotGuest(c,user);
}

event login_terminal(c: connection, terminal: string){
  konexioaSortu(c);
  logged_in(c);
  inkrHotGuest(c,terminal);
}

event login_display(c: connection, display: string){
  konexioaSortu(c);
  logged_in(c);
  inkrHotGuest(c,display);
}

event login_prompt(c: connection, prompt: string){
  konexioaSortu(c);
  logged_in(c);
  inkrHotGuest(c,prompt);
  root_shell(c, prompt);
}

# signatures
event ssh_signature_found(c: connection, is_orig: bool){
  konexioaSortu(c);
  logged_in(c);
}

event telnet_signature_found(c: connection, is_orig: bool, len: count){
  konexioaSortu(c);
  logged_in(c);
}

event rlogin_signature_found(c: connection, is_orig: bool,
      num_null: count, len: count){
  konexioaSortu(c);
  logged_in(c);
}

event root_backdoor_signature_found(c: connection){
  konexioaSortu(c);
  logged_in(c);
}

event ftp_signature_found(c: connection){
  konexioaSortu(c);
  logged_in(c);
}

event napster_signature_found(c: connection){
  konexioaSortu(c);
  logged_in(c);
}

event gnutella_signature_found(c: connection){
  konexioaSortu(c);
  logged_in(c);
}

event kazaa_signature_found(c: connection){
  konexioaSortu(c);
  logged_in(c);
}

event http_signature_found(c: connection){
```

```
  konexioaSortu(c);
  logged_in(c);
}

event http_proxy_signature_found(c: connection){
  konexioaSortu(c);
  logged_in(c);
}

event smtp_signature_found(c: connection){
  konexioaSortu(c);
  logged_in(c);
}

event irc_signature_found(c: connection){
  konexioaSortu(c);
  logged_in(c);
}

event gaobot_signature_found (c: connection){
  konexioaSortu(c);
  logged_in(c);
}

# concret servides that have usr
event finger_request(c: connection, full: bool,
      username: string, hostname: string){
  konexioaSortu(c);
  inkrHotGuest(c,username);
  num_root(c,username);
}

event ident_reply(c: connection, lport: port, rport: port,
      user_id: string, system: string){
  konexioaSortu(c);
  inkrHotGuest(c,user_id);
  num_root(c,user_id);
}

event rsh_request(c: connection, client_user: string,
      server_user: string, line: string, new: bool){
  konexioaSortu(c);
  inkrHotGuest(c,client_user);
  inkrHotGuest(c,server_user);
  num_root(c,client_user);
  num_root(c,server_user);
}

event rsh_reply(c: connection, client_user: string,
      server_user: string, line: string){
  konexioaSortu(c);
  inkrHotGuest(c,client_user);
  inkrHotGuest(c,server_user);
  num_root(c,client_user);
  num_root(c,server_user);
}

event pop3_login_success(c: connection, is_orig: bool,
      user: string, password: string){
  konexioaSortu(c);
  inkrHotGuest(c,user);
  num_root(c,user);
}

event pop3_login_failure(c: connection, is_orig: bool,
      user: string, password: string){
  konexioaSortu(c);
  inkrHotGuest(c,user);
  num_root(c,user);
}

event irc_who_line(c: connection, is_orig: bool,
      target_nick: string, channel: string, user: string,
      host: string, server: string, nick: string, params: string,
```

```
      hops: count, real_name: string){
  konexioaSortu(c);
  inkrHotGuest(c,user);
  num_root(c,user);
}

event irc_whois_message(c: connection, is_orig: bool,
      server: string, users: string){
  konexioaSortu(c);
  inkrHotGuest(c,users);
  num_root(c,users);
}

event irc_whois_user_line(c: connection, is_orig: bool,
      nick: string, user: string, host: string, real_name: string){
  konexioaSortu(c);
  inkrHotGuest(c,user);
  num_root(c,user);
}

event irc_oper_message(c: connection, is_orig: bool,
      user: string, password: string){
  konexioaSortu(c);
  inkrHotGuest(c,user);
  num_root(c,user);
}

event irc_kick_message(c: connection, is_orig: bool,
      prefix: string, chans: string, users: string, comment: string){
  konexioaSortu(c);
  inkrHotGuest(c,users);
  num_root(c,users);
}

event irc_names_info(c: connection, is_orig: bool,
      c_type: string, channel: string, users: string_set){
  konexioaSortu(c);
  for([erab] in users){
    inkrHotGuest(c,erab);
    num_root(c,erab);
  }
}

# outbound
event ftp_request(c: connection, command: string, arg: string){
  konexioaSortu(c);
  outbound(c,command);
}

##################### WEIRD ##############################

global wrong_fragment_set: set[string] = {
  ["bad_ICMP_checksum"],
  ["bad_TCP_checksum"],
  ["bad_UDP_checksum"],
};

event conn_weird (name: string, c: connection, addl: string){
  local trans : string = fmt("%s",
                        get_port_transport_proto(c$id$resp_p));
  local startTime : string = fmt("%.6f", c$start_time-6*60min);
  local duration : string = fmt("%.6f", c$duration);
  local orig_h : addr = c$id$orig_h;
  local resp_h : addr = c$id$resp_h;
  local orig_p : port = c$id$orig_p;
  local resp_p : port = c$id$resp_p;

  konexioaSortu(c);

  if (name in wrong_fragment_set){
    if(trans=="tcp")
      ++konexioaktcp[startTime, orig_h, orig_p,
        resp_h, resp_p]$wrong_fragment;
    else
```

```
        konexioak[startTime, duration, orig_h, orig_p,
          resp_h, resp_p]$wrong_fragment=1;
    }
}

event conn_weird_addl(name: string, c: connection, addl: string)
{
}

event flow_weird(name: string, src: addr, dst: addr)
{
}

event net_weird(name: string)
{
}
```

A.2 Calling to darpa2gurekddcup.bro

The files created by bro need to be renamed with week and day information. In the following script how the call to darpa2gurekddcup.bro policy was done and afterwards how the rename of payloads was done can be seen.

Listing 45: How to call the onlybro.sh bash script.

```
$ ./onlybro.sh Week1 Monday tcpdump
```

Next, the bash script who calls to darpa2gurekddcup.bro.

Listing 46: The onlybro.sh script.

```
#!/bin/sh

# create Week folder
mkdir    p  $1

# create day folder
if [ -d $1/$2 ]
then
        rm -R $1/$2
        mkdir $1/$2
else
        mkdir $1/$2
fi

# create payload folder
if [ -d payload ]
then
  rm -R payload
  mkdir payload
else
  mkdir payload
fi

echo "UNE GARRANTZITSUA: $1 $2 eguna-ren exekuzioa HASI: "
     `awk 'BEGIN{print strftime("%Y/%m/%d %H:%M:%S",systime())}'`

# run bro
echo "  bro + payload HASI: " `awk 'BEGIN{print strftime(
     "%Y/%m/%d %H:%M:%S",systime())}'`
bro -r $3 policyONA.bro > policy.list
echo "  bro + payload BUKATU: " `awk 'BEGIN{print strftime(
     "%Y/%m/%d %H:%M:%S",systime())}'`

# rename all payloads of the folder
echo "  payload berrizendatu HASI: " `awk 'BEGIN{print strftime(
     "%Y/%m/%d %H:%M:%S",systime())}'`

 if [ $1 = "Week1" ]; then
```

```
    week=1
 elif [ $1 = "Week2" ]; then
    week=2
 elif [ $1 = "Week3" ]; then
    week=3
 elif [ $1 = "Week4" ]; then
    week=4
 elif [ $1 = "Week5" ]; then
    week=5
 elif [ $1 = "Week6" ]; then
    week=6
 elif [ $1 = "Week7" ]; then
    week=7
 fi

 if [ $2 = "Monday" ]; then
    day=1
 elif [ $2 = "Tuesday" ]; then
    day=2
 elif [ $2 = "Wednesday" ]; then
    day=3
 elif [ $2 = "Thursday" ]; then
    day=4
 elif [ $2 = "Friday" ]; then
    day=5
 fi

 cd payload
 for k in 0 1 2 3 4 5 6 7 8 9
 do
  echo "    *$k.* payload berrizendatu HASI: " `awk 'BEGIN{
      print strftime(
        "%Y/%m/%d %H:%M:%S",
        systime())}'`
  for i in *$k.*
  do
         luz=`expr length "$i"`
         if [ $luz -eq 8 ]; then
                 mv $i $week$day$i
         elif [ $luz -eq 7 ]; then
                 mv $i $week$day"0"$i
         elif [ $luz -eq 6 ]; then
                 mv $i $week$day"00"$i
         elif [ $luz -eq 5 ]; then
                 mv $i $week$day"000"$i
         elif [ $luz -eq 4 ]; then
                 mv $i $week$day"0000"$i
         elif [ $luz -eq 3 ]; then
                 mv $i $week$day"00000"$i
         fi
  done
  echo "    *$k.* payload berrizendatu AMAITU: " `awk 'BEGIN{
      print strftime(
        "%Y/%m/%d %H:%M:%S",systime())}'`
 done
 cd ..

echo "  payload berrizendatu AMAITU: " `awk 'BEGIN{
    print strftime(
      "%Y/%m/%d %H:%M:%S",systime())}'`

mv policy.list $1/$2/
mv payload $1/$2/payload

echo "UNE GARRANTZITSUA: $1 $2 eguna-ren exekuzioa AMAITU: "
      `awk 'BEGIN{print strftime("%Y/%m/%d %H:%M:%S",systime())}'`
```

A.3 Computing the traffic variables

The darpa2gurekddcup.bro policy's output, darpa2gurekddcup.list, has intrinsic and content variables of each connection with its basic information. Therefore,

in this point we are going to compute the traffic variables providing to the script the sub-folders week and weekday as parameters (see Listing 47).

Listing 47: The call to the script to compute traffic variables.

```
$ ./trafAld.out policy.list Week1 Monday
```

Next in Listing 48, the C program to compute the traffic variables.

Listing 48: The script that computes traffic variables.

```
#include <stdio.h>
#include <string.h>

#define MaxCon 800000
#define MaxLine 800
#define MaxToken 20

char policy[MaxCon][MaxLine];
int policyKop;

int main(int argc, char **argv){
  FILE *fpolicy, *ftrafAld;
  int i,j;
  char galdera[MaxLine];
  int noraino;
  int nondik2, nondik100;
  char line[MaxLine];

  // for current connection's data
  char line1[MaxLine];
  int konZenb1;
  char hasUnea1s[MaxToken];
  int hasUnea1, hasUnea12;
  int orig_p1,resp_p1;
  char orig_h1[MaxToken], resp_h1[MaxToken];
  char duration1[MaxToken], protokoloa1[MaxToken],
       service1[MaxToken], flag1[MaxToken];

  // for previous connections' data
  char line2[MaxLine];
  int konZenb2;
  char hasUnea2s[MaxToken];
  int hasUnea2, hasUnea22;
  int orig_p2,resp_p2;
  char orig_h2[MaxToken], resp_h2[MaxToken];
  char duration2[MaxToken], protokoloa2[MaxToken],
       service2[MaxToken], flag2[MaxToken];
  int sartuDa2;

  // for traffic variables
  int count,srv_count,serror,rerror,same_srv,diff_srv,srv_serror,
      srv_error,srv_diff_host;
  int same_src_port;
  float serror_rate,srv_serror_rate,rerror_rate,srv_error_rate,
      same_srv_rate,diff_srv_rate,srv_diff_host_rate;
  float srv_rerror_rate,same_src_port_rate;

  if (argc != 4){
    printf("Deia: %s policy.list Week7 Monday\n", argv[0]);
    return(1);
  }

  // read policy.list and save
  sprintf(galdera, "%s/%s/policySort.list", argv[2], argv[3]);
  fpolicy = fopen(galdera, "r");
  policyKop = 0;
  while(!feof(fpolicy)){
        fgets(line2, MaxLine, fpolicy);
        line2[strlen(line2)-1] = '\0';
    strcpy(policy[policyKop],line2);
    policyKop = policyKop + 1;
  }
```

```
fclose(fpolicy);
policyKop=policyKop-1;//feof-z lerro bat gehiago irakurtzen baita

        // compute traffic variables
        nondik2=0;
        for(noraino=0; noraino<policyKop; noraino++){
        strcpy(line1, policy[noraino]);
          sscanf(line1, "%d %s %d %d %s %s %s %s %s %s %s",
            &konZenb1,&hasUnea1s,&orig_p1,&resp_p1,
          &orig_h1,&resp_h1,
            &duration1,&protokoloa1,&service1,&flag1);
          sscanf(hasUnea1s, "%d.%d", &hasUnea1, &hasUnea12);
          orig_h1[strlen(orig_h1)] = '\0';
          resp_h1[strlen(resp_h1)] = '\0';
          service1[strlen(service1)] = '\0';
          flag1[strlen(flag1)] = '\0';

          // the last connections established in last 2 seconds
          sartuDa2=0;
          count=0;
          serror=0;
          rerror=0;
          same_srv=0;
          diff_srv=0;
          srv_count=0;
          srv_serror=0;
          srv_error=0;
          srv_diff_host=0;
          for(j=nondik2; j<noraino; j++){
            strcpy(line2, policy[j]);
            sscanf(line2, "%d %s %d %d %s %s %s %s %s %s %s",
                &konZenb2,&hasUnea2s,&orig_p2,&resp_p2,
              &orig_h2,&resp_h2,
                &duration2,&protokoloa2,&service2,&flag2);
            sscanf(hasUnea2s, "%d.%d", &hasUnea2, &hasUnea22);
            orig_h2[strlen(orig_h2)] = '\0';
            resp_h2[strlen(resp_h2)] = '\0';
            service2[strlen(service2)] = '\0';
            flag2[strlen(flag2)] = '\0';
            if((hasUnea1-2)<=hasUnea2 && hasUnea2<=hasUnea1){
                if(sartuDa2==0){
                  nondik2=j;
                  sartuDa2=1;
                }
                if (strcmp(resp_h1,resp_h2)==0){
                  count= count + 1;
                  if (strcmp(flag2,"S0")==0 || strcmp(flag2,"S1")==0 ||
                  strcmp(flag2,"S2")==0 || strcmp(flag2,"S3")==0){
                    serror= serror + 1;
                  }
                  if (strcmp(flag2,"REJ")==0){
                    rerror= rerror + 1;
                  }
                  if (strcmp(service2,"other")!=0 &&
                  strcmp(service1,service2)==0){
                    same_srv= same_srv + 1;
                  }
                  if (strcmp(service1,service2)!=0){
                    diff_srv= diff_srv + 1;
                  }
                }
                if (resp_p1==resp_p2){
                  srv_count= srv_count + 1;
                  if (strcmp(flag2,"S0")==0 || strcmp(flag2,"S1")==0 ||
                  strcmp(flag2,"S2")==0 || strcmp(flag2,"S3")==0){
                        srv_serror= srv_serror + 1;
                  }
                  if (strcmp(flag2,"REJ")==0){
                    srv_error= srv_error + 1;
                  }
                  if (strcmp(resp_h1,resp_h2)!=0){
                    srv_diff_host= srv_diff_host + 1;
                  }
                }
```

```
      }
    }
    if (count!=0){
      serror_rate=(float)serror/(float)count;
      rerror_rate=(float)rerror/(float)count;
      same_srv_rate=(float)same_srv/(float)count;
      diff_srv_rate=(float)diff_srv/(float)count;
    }
    else{
      serror_rate= (float)0;
      rerror_rate= (float)0;
      same_srv_rate= (float)0;
      diff_srv_rate= (float)0;
    }
    if (srv_count!=0){
      srv_serror_rate=(float)srv_serror/(float)srv_count;
      srv_error_rate=(float)srv_error/(float)srv_count;
      srv_diff_host_rate=(float)srv_diff_host/(float)srv_count;
    }
    else{
      srv_serror_rate= (float)0;
      srv_error_rate= (float)0;
      srv_diff_host_rate= (float)0;
    }
    sprintf(line, "%s %d %d %f %f %f %f %f %f %f",
      line1,
    count,srv_count,serror_rate,srv_serror_rate,rerror_rate,
    srv_error_rate,same_srv_rate,
    diff_srv_rate,srv_diff_host_rate);
  line[strlen(line)] = '\0';
    strcpy(policy[noraino],line);

    //azken 100 konexioak
    if(noraino<=100){
      nondik100=0;
    }
    else{
      nondik100=noraino-100;
    }
    count=0;
    serror=0;
    rerror=0;
    same_srv=0;
    diff_srv=0;
    srv_count=0;
    srv_serror=0;
    srv_error=0;
    srv_diff_host=0;
    same_src_port=0;
    for(j=nondik100; j<noraino; j++){
    strcpy(line2, policy[j]);
      sscanf(line2, "%d %s %d %d %s %s %s %s %s %s %s",
          &konZenb2,&hasUnea2s,&orig_p2,&resp_p2,
      &orig_h2,&resp_h2,
          &duration2,&protokoloa2,&service2,&flag2);
      sscanf(hasUnea2s, "%d.%d", &hasUnea2, &hasUnea22);
      orig_h2[strlen(orig_h2)] = '\0';
      resp_h2[strlen(resp_h2)] = '\0';
      service2[strlen(service2)] = '\0';
      flag2[strlen(flag2)] = '\0';

    if (strcmp(resp_h1,resp_h2)==0){
  count= count + 1;
      if (strcmp(flag2,"S0")==0 || strcmp(flag2,"S1")==0 ||
          strcmp(flag2,"S2")==0 || strcmp(flag2,"S3")==0)
            serror= serror + 1;
  if (strcmp(flag2,"REJ")==0)
          rerror= rerror + 1;
      if (strcmp(service2,"other")!=0 &&
          strcmp(service1,service2)==0)
            same_srv= same_srv + 1;
      if (strcmp(service1,service2)!=0)
         diff_srv= diff_srv + 1;
    }
```

```
            if (resp_p1==resp_p2){
                srv_count= srv_count + 1;
              if (strcmp(flag2,"S0")==0 || strcmp(flag2,"S1")==0 ||
                  strcmp(flag2,"S2")==0 || strcmp(flag2,"S3")==0)
                    srv_serror= srv_serror + 1;
              if (strcmp(flag2,"REJ")==0)
                    srv_error= srv_error + 1;
              if (strcmp(resp_h1,resp_h2)!=0)
                srv_diff_host= srv_diff_host + 1;
            }
            if (orig_p1==orig_p2){
                same_src_port= same_src_port + 1;
              }
          }
          if (count!=0){
            serror_rate=(float)serror/(float)count;
            rerror_rate=(float)rerror/(float)count;
            same_srv_rate=(float)same_srv/(float)count;
              diff_srv_rate=(float)diff_srv/(float)count;
          }
          else{
              serror_rate= (float)0;
            rerror_rate= (float)0;
            same_srv_rate= (float)0;
              diff_srv_rate= (float)0;
             }
             if (srv_count!=0){
             srv_serror_rate=(float)srv_serror/(float)srv_count;
               srv_rerror_rate=(float)srv_error/(float)srv_count;
               srv_diff_host_rate=
               (float)srv_diff_host/(float)srv_count;
             }
             else{
               srv_serror_rate= (float)0;
               srv_rerror_rate= (float)0;
               srv_diff_host_rate= (float)0;
             }
             if(noraino-nondik100!=0)
               same_src_port_rate=
              (float)same_src_port/(float)(noraino-nondik100);
             else
               same_src_port_rate=(float)0;

            strcpy(line1, policy[noraino]);
         sprintf(line, "%s %d %d %f %f %f %f %f %f %f",
            line1,
            count,srv_count,same_srv_rate,diff_srv_rate,
            same_src_port_rate,srv_diff_host_rate,serror_rate,
            srv_serror_rate,rerror_rate,srv_rerror_rate);
            line[strlen(line)] = '\0';
            strcpy(policy[noraino],line);
          }

          // write the variables
          sprintf(galdera, "%s/%s/trafAld.list", argv[2], argv[3]);
          ftrafAld = fopen(galdera, "w");
          for(i=0; i<policyKop; i++){
          fprintf(ftrafAld, "%s\n", policy[i]);
          }
          fclose(ftrafAld);
}
```

A.4 Labelling connections

In the following lines the process of labelling connections is indicated. The program needs the labelled connections from Darpa98 (tcpdump.list), the computed connections (trafAld.list) and the sub-folders week and weekday (see Listing 49).

Listing 49: The call to the script to label the connections.

```
$ ./labelled.out tcpdump.list trafAld.list Week1 Monday
```

Next in Listing 50, the C program we used to label the connections.

Listing 50: The C program used to label the connections.

```
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>

#define MaxCon 800000
#define MaxLine 800
#define MaxToken 20

char trafAld[MaxCon][MaxLine];
int trafKop;
char tcpdump[MaxCon][MaxLine];
int tcpKop;
char ezkonduak[MaxCon][MaxLine];
int ezkKop;

void zuzen(){
  int i,j,jLehen;
  char ezkLine[MaxLine];

  // for connections in tcpdump.list
  char line1[MaxLine];
  int konZenb1, erasoaDa;
  char orig_p1[MaxToken], resp_p1[MaxToken];
  int hasUnea1;
  char data1[MaxToken], ordua1[MaxToken], duration1[MaxToken],
       service1[MaxToken];
  char daybuf[MaxToken];
  int urt1, hil1, egu1, ord1, min1, seg1;
  char orig_h1[MaxToken], resp_h1[MaxToken], erasoa[MaxToken];
  int orig_h11, orig_h12, orig_h13, orig_h14, resp_h11, resp_h12,
      resp_h13, resp_h14;

  // for connections in trafAld.list
  char line2[MaxLine];
  int konZenb2;
  char orig_p2[MaxToken], resp_p2[MaxToken];
  int orig_p2Int, resp_p2Int;
  int hasUnea2, hasUnea22;
  char hasUnea2s[MaxToken];
  char data2[MaxToken], ordua2[MaxToken];
  char orig_h2[MaxToken], resp_h2[MaxToken];
  int orig_h21, orig_h22, orig_h23, orig_h24, resp_h21, resp_h22,
      resp_h23, resp_h24;
  char durations[MaxToken], protokoloa[MaxToken];

  jLehen = 0;
  for(i=0; i<tcpKop; i++){
     // get tcpdump.list fields
     strcpy(line1, tcpdump[i]);
     sscanf(line1, "%d %d %s %s %s %s %s %s %d %s\n",
         &konZenb1,&hasUnea1,&duration1,&service1,&orig_p1,&resp_p1,
         &orig_h1,&resp_h1,&erasoaDa,&erasoa);
     orig_p1[strlen(orig_p1)] = '\0';
     resp_p1[strlen(resp_p1)] = '\0';
     erasoa[strlen(erasoa)] = '\0';
     sscanf(orig_h1, "%d.%d.%d.%d", &orig_h11, &orig_h12,
            &orig_h13, &orig_h14);
     sscanf(resp_h1, "%d.%d.%d.%d", &resp_h11, &resp_h12,
            &resp_h13, &resp_h14);

     j = jLehen;
     while(j<trafKop){
        if(trafAld[j][0]=='\0'){
            j = j + 1;
            continue;
        }
        strcpy(line2, trafAld[j]);
```

```
        sscanf(line2, "%d %s %s %s %s %s %s",
           &konZenb2,&hasUnea2s,&orig_p2,&resp_p2,&orig_h2,&resp_h2,
           &durations,&protokoloa);
        orig_p2[strlen(orig_p2)] = '\0';
        resp_p2[strlen(resp_p2)] = '\0';
        protokoloa[strlen(protokoloa)] = '\0';
        sscanf(hasUnea2s, "%d.%d", &hasUnea2, &hasUnea22);
        sscanf(orig_h2, "%d.%d.%d.%d", &orig_h21, &orig_h22,
               &orig_h23, &orig_h24);
        sscanf(resp_h2, "%d.%d.%d.%d", &resp_h21, &resp_h22,
               &resp_h23, &resp_h24);
        if(hasUnea1>hasUnea2){
            jLehen = j;
            j = j + 1;
            continue;
        }
        if(hasUnea2>hasUnea1){
            break;
        }
        if(hasUnea1==hasUnea2 &&
           (orig_h11==orig_h21 && orig_h12==orig_h22 &&
            orig_h13==orig_h23 && orig_h14==orig_h24) &&
           (resp_h11==resp_h21 && resp_h12==resp_h22 &&
            resp_h13==resp_h23 && resp_h14==resp_h24)){
            if(strcmp(orig_p1,orig_p2)==0 &&
               strcmp(resp_p1, resp_p2)==0){
              sprintf(ezkLine, "%s %d %s %s",
                      line2, erasoaDa, erasoa, "0");
              strcpy(ezkonduak[ezkKop], ezkLine);
              tcpdump[i][0] = '\0';
              trafAld[j][0] = '\0';
              ezkKop = ezkKop + 1;
              break;
            }
            else if(strcmp(protokoloa,"icmp")==0 ||
                  (strcmp(orig_p1,"-")==0 ||
                   strcmp(resp_p1,"-")==0)){
              sprintf(ezkLine, "%s %d %s %s",
                      line2, erasoaDa, erasoa, "0");
              strcpy(ezkonduak[ezkKop], ezkLine);
              tcpdump[i][0] = '\0';
              trafAld[j][0] = '\0';
              ezkKop = ezkKop + 1;
              break;
            }
        }
        j = j + 1;
      }
   }
   printf("zuzen: %d\n", ezkKop);
}

void trukatuta(){
  int i,j,jLehen;
  char ezkLine[MaxLine];

  // for connections in tcpdump.list
  char line1[MaxLine];
  int konZenb1, erasoaDa;
  char orig_p1[MaxToken], resp_p1[MaxToken];
  int hasUnea1;
  char data1[MaxToken], ordua1[MaxToken], duration1[MaxToken],
       service1[MaxToken];
  char daybuf[MaxToken];
  int urt1, hil1, egu1, ord1, min1, seg1;
  char orig_h1[MaxToken], resp_h1[MaxToken], erasoa[MaxToken];
  int orig_h11, orig_h12, orig_h13, orig_h14, resp_h11, resp_h12,
      resp_h13, resp_h14;

  // for connections in trafAld.list:
  char line2[MaxLine];
  int konZenb2;
  char orig_p2[MaxToken], resp_p2[MaxToken];
  int hasUnea2, hasUnea22;
```

```
char hasUnea2s[MaxToken];
char data2[MaxToken], ordua2[MaxToken];
char orig_h2[MaxToken], resp_h2[MaxToken];
int orig_h21, orig_h22, orig_h23, orig_h24, resp_h21, resp_h22,
    resp_h23, resp_h24;
char durations[MaxToken], protokoloa[MaxToken];

// to interchange:
int f1,f3,f4,f9,f11,f12,f13,f14,f15,f16,f17,f18,f19,f20,
    f21,f22,f23,f24,f25,f26,f27,f28,f29,f30,f38,f39;
float f7,f31,f32,f33,f34,f35,f36,f37,f40,f41,f42,f43,f44,f45,f46,f47;
char f2[MaxToken], f5[MaxToken], f6[MaxToken],
     f8[MaxToken], f10[MaxToken];

jLehen = 0;
i=0;
while(i<tcpKop){
  if(tcpdump[i][0]=='\0'){
      i = i + 1;
      continue;
  }
  // get fields from tcpdump.list
  strcpy(line1, tcpdump[i]);
  sscanf(line1, "%d %d %s %s %s %s %s %d %s\n",
         &konZenb1,&hasUnea1,&duration1,&service1,&orig_p1,&resp_p1,
         &orig_h1,&resp_h1,&erasoaDa,&erasoa);
  orig_p1[strlen(orig_p1)] = '\0';
  orig_p1Int = atoi(orig_p1); //orig_p1Int==0 bada icmp konexioa da
  resp_p1[strlen(resp_p1)] = '\0';
  resp_p1Int = atoi(resp_p1); //resp_p1Int==0 bada icmp konexioa da
  erasoa[strlen(erasoa)] = '\0';
  sscanf(orig_h1, "%d.%d.%d.%d", &orig_h11, &orig_h12,
         &orig_h13, &orig_h14);
  sscanf(resp_h1, "%d.%d.%d.%d", &resp_h11, &resp_h12,
         &resp_h13, &resp_h14);

  j = jLehen;
  while(j<trafKop){
      if(trafAld[j][0]=='\0'){
        j = j + 1;
        continue;
      }

  strcpy(line2, trafAld[j]);
  sscanf(line2, "%d %s %s %s %s %s %s",
         &konZenb2,&hasUnea2s,&orig_p2,&resp_p2,&orig_h2,&resp_h2,
         &durations,&protokoloa);
  orig_p2[strlen(orig_p2)] = '\0';
  resp_p2[strlen(resp_p2)] = '\0';
  protokoloa[strlen(protokoloa)] = '\0';
  sscanf(hasUnea2s, "%d.%d", &hasUnea2, &hasUnea22);
  sscanf(orig_h2, "%d.%d.%d.%d", &orig_h21, &orig_h22,
        &orig_h23, &orig_h24);
  sscanf(resp_h2, "%d.%d.%d.%d", &resp_h21, &resp_h22,
        &resp_h23, &resp_h24);

  if(hasUnea1>hasUnea2){
      jLehen = j;
      j = j + 1;
      continue;
  }

  if(hasUnea2>hasUnea1){
      break;
  }

  if(hasUnea1==hasUnea2 &&
     (orig_h11==resp_h21 && orig_h12==resp_h22 &&
      orig_h13==resp_h23 && orig_h14==resp_h24) &&
     (resp_h11==orig_h21 && resp_h12==orig_h22 &&
      resp_h13==orig_h23 && resp_h14==orig_h24)){

      if(strcmp(orig_p1,resp_p2)==0 &&
         strcmp(resp_p1,orig_p2)==0){ // TCP-UDP
```

```
          sscanf(line2, "%d %s %d %d %s %s %f %s %d
                    %s %d %d %d %d %d %d
                    %d %d %d %d %d %d %d
                    %d %d %d %d %d %d %d %f %f
                    %f %f %f %f %f %d %d %f %f
                    %f %f %f %f %f %f",
                &f1,&f2,&f3,&f4,&f5,&f6,&f7,&f8,&f9,
                      &f10,&f11,&f12,&f13,
                &f14,&f15,&f16,&f17,&f18,&f19,
                &f20,&f21,&f22,&f23,&f24,&f25,
                &f26,&f27,&f28,&f29,&f30,&f31,
                &f32,&f33,&f34,&f35,&f36,&f37,
                &f38,&f39,&f40,&f41,&f42,&f43,
                &f44,&f45,&f46,&f47);

          f2[strlen(f2)]='\0'; f5[strlen(f5)]='\0'; f6[strlen(f6)]='\0';
          f8[strlen(f8)]='\0'; f10[strlen(f10)]='\0';

          sprintf(ezkLine, "%d %s %d %d %s %s %f %s
                    %d %s %d %d %d %d %d
                    %d %d %d %d %d %d %d %d
                    %d %d %d %d %d %d %f %f
                    %f %f %f %f %f %d %d %f
                    %f %f %f %f %f %f %d %s 1",
                        f1,f2,f4,f3,f6,f5,f7,f8,f9,f10,f11,
                        f12,f13,f14,f15,f16,
                f17,f18,f19,f20,f21,f22,f23,f24,f25,
                f26,f27,f28,f29,f30,
                f31,f32,f33,f34,f35,f36,f37,f38,f39,
                f40,f41,f42,f43,f44,
                f45, f46, f47, erasoaDa, erasoa);

          strcpy(ezkonduak[ezkKop], ezkLine);
          tcpdump[i][0] = '\0';
          trafAld[j][0] = '\0';
          ezkKop = ezkKop + 1;
          break;
         }
         else if(strcmp(protokoloa,"icmp")==0 ||
               (strcmp(orig_p1,"-")==0 || strcmp(resp_p1,"-")==0)){
          //icmp denean
          sscanf(line2, "%d %s %d %d %s %s %f %s %d %s %d
                    %d %d %d %d %d %d
                    %d %d %d %d %d %d %d %d %d %d %d
                    %d %d %f %f %f %f %f %f %f
                    %d %d %f %f %f %f %f %f %f %f\n",
                &f1,&f2,&f3,&f4,&f5,&f6,&f7,&f8,
                &f9,&f10,&f11,&f12,&f13,
                &f14,&f15,&f16,&f17,&f18,&f19,
                &f20,&f21,&f22,&f23,&f24,
                &f25,&f26,&f27,&f28,&f29,&f30,
                &f31,&f32,&f33,&f34,&f35,&f36,
                &f37,&f38,&f39,&f40,&f41,&f42,
                &f43,&f44,&f45,&f46,&f47);

          f2[strlen(f2)]='\0'; f5[strlen(f5)]='\0'; f6[strlen(f6)]='\0';
          f8[strlen(f8)]='\0'; f10[strlen(f10)]='\0';

          sprintf(ezkLine, "%d %s %d %d %s %s %f %s %d
                    %s %d %d %d %d %d %d
                    %d %d %d %d %d %d %d %d %d
                    %d %d %d %d %d %f %f %f %f %f
                    %f %f %d %d %f %f %f %f %f %f
                    %f %f %d %s 1",
                        f1,f2,f3,f4,f6,f5,f7,f8,f9,f10,
                        f11,f12,f13,f14,f15,f16,
                f17,f18,f19,f20,f21,f22,f23,f24,
                f25,f26,f27,f28,f29,f30,
                f31,f32,f33,f34,f35,f36,f37,f38,
                f39,f40,f41,f42,f43, f44,
                f45,f46,f47,erasoaDa,erasoa);

          strcpy(ezkonduak[ezkKop], ezkLine);
          tcpdump[i][0] = '\0';
```

```
          trafAld[j][0] = '\0';
          ezkKop = ezkKop + 1;
          break;
         }
         j = j + 1;
      }
      i = i + 1;
    }
    printf("truka: %d\n", ezkKop);
}

void desfasearekin(){
  int i,j,jLehen;
  char ezkLine[MaxLine];

  // for connections in tcpdump.list
  char line1[MaxLine];
  int konZenb1, erasoaDa;
  char orig_p1[MaxToken], resp_p1[MaxToken];
  int orig_p1Int, resp_p1Int;
  int hasUnea1;
  char data1[MaxToken], ordua1[MaxToken], duration1[MaxToken],
       service1[MaxToken];
  char daybuf[MaxToken];
  int urt1, hil1, egu1, ord1, min1, seg1;
  char orig_h1[MaxToken], resp_h1[MaxToken], erasoa[MaxToken];
  int orig_h11, orig_h12, orig_h13, orig_h14, resp_h11, resp_h12,
      resp_h13, resp_h14;

  // for connections in trafAld.list
  char line2[MaxLine];
  int konZenb2;
  char orig_p2[MaxToken], resp_p2[MaxToken];
  int hasUnea2, hasUnea22;
  char hasUnea2s[MaxToken];
  char data2[MaxToken], ordua2[MaxToken];
  char orig_h2[MaxToken], resp_h2[MaxToken];
  int orig_h21, orig_h22, orig_h23, orig_h24, resp_h21, resp_h22,
      resp_h23, resp_h24;
  char durations[MaxToken], protokoloa[MaxToken];

  jLehen = 0;
  i=0;
  while(i<tcpKop){
    if(tcpdump[i][0]=='\0'){
        i = i + 1;
        continue;
    }
    // get fields from tcpdump.list
    strcpy(line1, tcpdump[i]);
    sscanf(line1, "%d %d %s %s %s %s %s %s %d %s\n",
         &konZenb1,&hasUnea1,&duration1,&service1,&orig_p1,&resp_p1,
         &orig_h1,&resp_h1,&erasoaDa,&erasoa);
    orig_p1[strlen(orig_p1)] = '\0';
    resp_p1[strlen(resp_p1)] = '\0';
    erasoa[strlen(erasoa)] = '\0';
    sscanf(orig_h1, "%d.%d.%d.%d", &orig_h11, &orig_h12,
           &orig_h13, &orig_h14);
    sscanf(resp_h1, "%d.%d.%d.%d", &resp_h11, &resp_h12,
           &resp_h13, &resp_h14);

    j = jLehen;
    while(j<trafKop){
        if(trafAld[j][0]=='\0'){
          j = j + 1;
          continue;
        }

      strcpy(line2, trafAld[j]);
      sscanf(line2, "%d %s %s %s %s %s %s",
           &konZenb2,&hasUnea2s,&orig_p2,&resp_p2,&orig_h2,&resp_h2,
           &durations,&protokoloa);
      orig_p2[strlen(orig_p2)] = '\0';
      resp_p2[strlen(resp_p2)] = '\0';
```

```
      protokoloa[strlen(protokoloa)] = '\0';
      sscanf(hasUnea2s, "%d.%d", &hasUnea2, &hasUnea22);
      sscanf(orig_h2, "%d.%d.%d.%d", &orig_h21, &orig_h22,
             &orig_h23, &orig_h24);
      sscanf(resp_h2, "%d.%d.%d.%d", &resp_h21, &resp_h22,
             &resp_h23, &resp_h24);

    if(hasUnea1-3>hasUnea2+3){
      jLehen = j;
      j = j + 1;
      continue;
    }

    if(hasUnea2-3>hasUnea1+3){
        break;
    }

    if((hasUnea2-3)<=hasUnea1 && hasUnea1<=(hasUnea2+3) &&
       (orig_h11==orig_h21 && orig_h12==orig_h22 &&
        orig_h13==orig_h23 && orig_h14==orig_h24) &&
        (resp_h11==resp_h21 && resp_h12==resp_h22 &&
        resp_h13==resp_h23 && resp_h14==resp_h24)){

        if(strcmp(orig_p1,orig_p2)==0 && strcmp(resp_p1, resp_p2)==0){
         sprintf(ezkLine, "%s %d %s %s", line2, erasoaDa, erasoa, "0");
         strcpy(ezkonduak[ezkKop], ezkLine);
         tcpdump[i][0] = '\0';
         trafAld[j][0] = '\0';
         ezkKop = ezkKop + 1;
         break;
      }
      else if(strcmp(protokoloa,"icmp")==0 ||
              (strcmp(orig_p1,"-")==0 || strcmp(resp_p1,"-")==0)){
         sprintf(ezkLine, "%s %d %s %s", line2, erasoaDa, erasoa, "0");
         strcpy(ezkonduak[ezkKop], ezkLine);
         tcpdump[i][0] = '\0';
         trafAld[j][0] = '\0';
         ezkKop = ezkKop + 1;
         break;
      }
     }
     j = j + 1;
   }
   i = i + 1;
  }
  printf("desbi: %d\n", ezkKop);
}

void desfasearekinTrukatuta(){
  int i,j,jLehen;
  char ezkLine[MaxLine];

  // for connections in tcpdump.list
  char line1[MaxLine];
  int konZenb1, erasoaDa;
  char orig_p1[MaxToken], resp_p1[MaxToken];
  int hasUnea1;
  char data1[MaxToken], ordua1[MaxToken], duration1[MaxToken],
       service1[MaxToken];
  char daybuf[MaxToken];
  int urt1, hil1, egu1, ord1, min1, seg1;
  char orig_h1[MaxToken], resp_h1[MaxToken], erasoa[MaxToken];
  int orig_h11, orig_h12, orig_h13, orig_h14, resp_h11, resp_h12,
      resp_h13, resp_h14;

  //for connections in trafAld.list
  char line2[MaxLine];
  int konZenb2;
  char orig_p2[MaxToken], resp_p2[MaxToken];
  int hasUnea2, hasUnea22;
  char hasUnea2s[MaxToken];
  char data2[MaxToken], ordua2[MaxToken];
  char orig_h2[MaxToken], resp_h2[MaxToken];
  int orig_h21, orig_h22, orig_h23, orig_h24, resp_h21, resp_h22,
```

```
      resp_h23, resp_h24;
  char durations[MaxToken], protokoloa[MaxToken];

  // to interchange:
  int f1,f3,f4,f9,f11,f12,f13,f14,f15,f16,f17,
      f18,f19,f20,f21,f22,f23,f24,
      f25,f26,f27,f28,f29,f30,f38,f39;
  float f7,f31,f32,f33,f34,f35,f36,f37,f40,f41,
        f42,f43,f44,f45,f46,f47;
  char f2[MaxToken], f5[MaxToken], f6[MaxToken],
       f8[MaxToken], f10[MaxToken];

  jLehen = 0;
  i=0;
  while(i<tcpKop){
    if(tcpdump[i][0]=='\0'){
        i = i + 1;
        continue;
    }
    //tcpdump.list-eko eremuak lortu
    strcpy(line1, tcpdump[i]);
    sscanf(line1, "%d %d %s %s %s %s %s %s %d %s\n",
        &konZenb1,&hasUnea1,&duration1,&service1,&orig_p1,&resp_p1,
        &orig_h1,&resp_h1,&erasoaDa,&erasoa);
    orig_p1[strlen(orig_p1)] = '\0';
    resp_p1[strlen(resp_p1)] = '\0';
    erasoa[strlen(erasoa)] = '\0';
    sscanf(orig_h1, "%d.%d.%d.%d", &orig_h11, &orig_h12,
           &orig_h13, &orig_h14);
    sscanf(resp_h1, "%d.%d.%d.%d", &resp_h11, &resp_h12,
           &resp_h13, &resp_h14);

    j = jLehen;
    while(j<trafKop){
    if(trafAld[j][0]=='\0'){
        j = j + 1;
        continue;
    }

    strcpy(line2, trafAld[j]);
    sscanf(line2, "%d %s %s %s %s %s %s",
        &konZenb2,&hasUnea2s,&orig_p2,&resp_p2,&orig_h2,&resp_h2,
        &durations,&protokoloa);
    orig_p2[strlen(orig_p2)] = '\0';
    resp_p2[strlen(resp_p2)] = '\0';
    protokoloa[strlen(protokoloa)] = '\0';
    sscanf(hasUnea2s, "%d.%d", &hasUnea2, &hasUnea22);
    sscanf(orig_h2, "%d.%d.%d.%d", &orig_h21, &orig_h22,
           &orig_h23, &orig_h24);
    sscanf(resp_h2, "%d.%d.%d.%d", &resp_h21, &resp_h22,
           &resp_h23, &resp_h24);

    if(hasUnea1-3>hasUnea2+3){
        jLehen = j;
        j = j + 1;
        continue;
    }

    if(hasUnea2-3>hasUnea1+3){
        break;
    }

    if((hasUnea2-3)<=hasUnea1 && hasUnea1<=(hasUnea2+3) &&
       (orig_h11==resp_h21 && orig_h12==resp_h22 &&
        orig_h13==resp_h23 && orig_h14==resp_h24) &&
       (resp_h11==orig_h21 && resp_h12==orig_h22 &&
        resp_h13==orig_h23 && resp_h14==orig_h24)){

        if(strcmp(orig_p1,resp_p2)==0 && strcmp(resp_p1,orig_p2)==0){

         sscanf(line2, "%d %s %d %d %s %s %f %s %d
                %s %d %d %d %d %d %d %d
                %d %d %d %d %d %d %d %d
                %d %d %d %d %d %f %f %f %f
```

```
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%f␣%f␣%f␣%d␣%d␣%f␣%f␣%f␣%f
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%f␣%f␣%f␣%f",
              &f1,&f2,&f3,&f4,&f5,&f6,&f7,
              &f8,&f9,&f10,&f11,
              &f12,&f13,&f14,&f15,&f16,&f17,
              &f18,&f19,&f20,&f21,&f22,
              &f23,&f24,&f25,&f26,&f27,&f28,
              &f29,&f30,&f31,&f32, &f33,
              &f34,&f35,&f36,&f37,&f38,&f39,
              &f40,&f41,&f42,&f43,&f44,
              &f45,&f46,&f47);

          f2[strlen(f2)]='\0'; f5[strlen(f5)]='\0'; f6[strlen(f6)]='\0';
          f8[strlen(f8)]='\0'; f10[strlen(f10)]='\0';

          sprintf(ezkLine, "%d␣%s␣%d␣%d␣%s␣%s␣%f␣%s␣%d
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%s␣%d␣%d␣%d␣%d␣%d␣%d
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%d␣%d␣%d␣%d␣%f␣%f
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%f␣%f␣%f␣%f␣%f␣%d␣%d␣%f␣%f␣%f␣%f
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%f␣%f␣%f␣%f␣%f␣%d␣%s␣1",
              f1,f2,f4,f3,f6,f5,f7,f8,f9,f10,f11,f12,f13,f14,f15,
              f16,f17,f18,f19,f20,f21,f22,f23,f24,f25,f26,f27,f28,
              f29,f30,f31,f32,f33,f34,f35,f36,f37,f38,f39,f40,f41,
              f42,f43,f44,f45,f46,f47,erasoaDa,erasoa);

          strcpy(ezkonduak[ezkKop], ezkLine);
          tcpdump[i][0] = '\0';
          trafAld[j][0] = '\0';
          ezkKop = ezkKop + 1;
          break;
        }
        else if(strcmp(protokoloa,"icmp")==0 ||
              (strcmp(orig_p1,"-")==0 || strcmp(resp_p1,"-")==0)){

          sscanf(line2, "%d␣%s␣%d␣%d␣%s␣%s␣%f␣%s
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%d␣%s␣%d␣%d␣%d␣%d␣%d
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%f␣%f
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%f␣%f␣%f␣%f␣%f␣%d␣%d␣%f␣%f
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%f␣%f␣%f␣%f␣%f␣%f␣%f\n",
              &f1,&f2,&f3,&f4,&f5,&f6,&f7,&f8,
              &f9,&f10,&f11,
              &f12,&f13,&f14,&f15,&f16,&f17,
              &f18,&f19,&f20,&f21,&f22,
              &f23,&f24,&f25,&f26,&f27,&f28,
              &f29,&f30,&f31,&f32,&f33,
              &f34,&f35,&f36,&f37,&f38,&f39,
              &f40,&f41,&f42,&f43,&f44,
              &f45,&f46,&f47);

          f2[strlen(f2)]='\0'; f5[strlen(f5)]='\0'; f6[strlen(f6)]='\0';
          f8[strlen(f8)]='\0'; f10[strlen(f10)]='\0';

          sprintf(ezkLine, "%d␣%s␣%d␣%d␣%s␣%s␣%f␣%s
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%d␣%s␣%d␣%d␣%d␣%d
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%f␣%f␣%f␣%f␣%f␣%f␣%f␣%f␣%d␣%d
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%f␣%f␣%f␣%f␣%f␣%f␣%f␣%f␣%f␣%d␣%s␣1",
              f1,f2,f3,f4,f6,f5,f7,f8,f9,f10,f11,
              f12,f13,f14,f15,
              f16,f17,f18,f19,f20,f21,f22,f23,f24,
              f25,f26,f27,f28,f29,
              f30,f31,f32,f33,f34,f35,f36,f37,f38,
              f39,f40,f41,f42,f43,
              f44,f45,f46,f47, erasoaDa, erasoa);

          strcpy(ezkonduak[ezkKop], ezkLine);
          tcpdump[i][0] = '\0';
          trafAld[j][0] = '\0';
          ezkKop = ezkKop + 1;
          break;
        }
```

```
      }
      j = j + 1;
    }
    i = i + 1;
  }
  printf("desbiTruk: %d\n", ezkKop);
}

int main(int argc, char **argv){
  FILE *ftcpdump, *ftrafAld;
  FILE *f1ourKddcup, *f2ourKddcup, *f3ourKddcup, *fourKddcup;
  int i;
  char galdera[MaxLine];
  char line1[MaxLine];
  char line2[MaxLine];

  if (argc != 5){
    printf("Deia: %s tcpdump.list trafAld.list Week7 Monday\n",
           argv[0]);
    return (1);
  }

  // order tcpdump.list and trafAld.list
  sprintf(galdera, "sort -k 2 %s > %s/%s/tmp/tcpdumpSort.list",
          argv[1], argv[3], argv[4]);
  system(galdera);
  sprintf(galdera, "sort -k 2 %s > %s/%s/tmp/trafAldSort.list",
          argv[2], argv[3], argv[4]);
  system(galdera);

  // read and save tcpdump.list
  sprintf(galdera, "%s/%s/tmp/tcpdumpSort.list", argv[3], argv[4]);
  ftcpdump = fopen(galdera, "r");
  tcpKop = 0;
  while(!feof(ftcpdump)){
    fgets(line1, MaxLine, ftcpdump);
      line1[strlen(line1)-1] = '\0';
      strcpy(tcpdump[tcpKop],line1);
      tcpKop = tcpKop + 1;
  }
  fclose(ftcpdump);
  tcpKop = tcpKop - 1; //feof-z lerro bat gehiago irakurtzen baita

  //read and save trafAld.list
  sprintf(galdera, "%s/%s/tmp/trafAldSort.list", argv[3], argv[4]);
  ftrafAld = fopen(galdera, "r");
  trafKop = 0;
  while(!feof(ftrafAld)){
    fgets(line2, MaxLine, ftrafAld);
    line2[strlen(line2)-1] = '\0';
    strcpy(trafAld[trafKop],line2);
    trafKop = trafKop + 1;
  }
  fclose(ftrafAld);
  trafKop = trafKop - 1; // quit EOF (End Of File)

  // initialize number of matching
  ezkKop = 0;

  // match correctly
  zuzen();

  // match interchanged
  trukatuta();

  // match with plus minus 3 seconds (time difference)
  desfasearekin();

  // with time difference and interchanged
  desfasearekinTrukatuta();

  // write the results
  // 1ourKddcup: only in bro
  sprintf(galdera, "%s/%s/1ourKddcup.list", argv[3], argv[4]);
```

```
  f1ourKddcup = fopen(galdera, "w");
  for(i=0; i<trafKop; i++){
    if(trafAld[i][0]!='\0'){
        fprintf(f1ourKddcup, "%s\n", trafAld[i]);
    }
  }
  fclose(f1ourKddcup);

  //2ourKddcup: only in tcpdump.list
  sprintf(galdera, "%s/%s/2ourKddcup.list", argv[3], argv[4]);
  f2ourKddcup = fopen(galdera, "w");
  for(i=0; i<tcpKop; i++){
    if(tcpdump[i][0]!='\0'){
        fprintf(f2ourKddcup, "%s\n", tcpdump[i]);
    }
  }
  fclose(f2ourKddcup);

  //3ourKddcup: matched ones
  sprintf(galdera, "%s/%s/3ourKddcup.list", argv[3], argv[4]);
  f3ourKddcup = fopen(galdera, "w");
  for(i=0; i<ezkKop; i++){
    fprintf(f3ourKddcup, "%s\n", ezkonduak[i]);
  }
  fclose(f3ourKddcup);

  // write ourKddcup
  sprintf(galdera, "%s/%s/ourKddcup.list", argv[3], argv[4]);
  fourKddcup = fopen(galdera, "w");
  for(i=0; i<trafKop; i++){
    if(trafAld[i][0]!='\0'){
          fprintf(fourKddcup, "%s %s\n", trafAld[i], "2 - 0");
       }
  }
  for(i=0; i<ezkKop; i++){
    fprintf(fourKddcup, "%s\n", ezkonduak[i]);
  }
  fclose(fourKddcup);
}
```

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
avg	47.98	31.36	216.66	5.37	69.03	5.05	0.01	259.82	442.56
max	58329	248692	58329	248397	248692	42448	105	15168	5161
min	0	0	0	0	0	0	0	0	0

Table 12: Intrinsic variable: DURATION.

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
icmp	283602	756416	1288	34628	668553	282050	53226	264	9
udp	20354	868491	19177	262852	597018	195	7528	982	1093
tcp	190065	3223620	76813	832376	823462	112056	1564942	1196	2840
sum	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 13: Intrinsic variable: PROTOCOL_TYPE.

B Compare gureKDDCup.list and KDDCup99-10percent.data

B.1 Intrinsic variables

The following tables present the values obtained for each intrinsic variable: for all the database (all), for all normal connections (normal), for flood type connections (flood) and non-flood connections (noFlood).

Intrinsic variables: (1) duration in Table 12; (2) protocol_type in Table 13; (3) services in Table 14 and 15; (4) flag in Table 16; (5) src_bytes in Table 17; (6) dst_bytes in Table 18; (7) land in Table 19; (8) wrong_fragment in Table 20; (9) urgent in Table 21;

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
auth	328	6978	220	2548	2914	108	1516		
domain	116	491294	3	60520	429243	113	1531		
domain_u	5863		5862			1			
echo	112				3846	112			
eco_i	1642	27376	389	8015	652240	1253	15511		4
ecr_i	281400	689941	345	4	6146	280796	37697	259	
finger	670	14177	468	5573		182	2324	20	134
ftp	798	11608	373	4270	5373	109	1539	316	426
ftp_data	4721	2413	3798	39	82	178	2291	745	1
http	64293	1284844	61886	665251	615048	2403	4540	4	5
http_alt		196		141	54		1		
imap4	117	1498			2	105	1496	12	
IRC	43	1508	42		3	1	1505		
ircu		3450		623	2817		10		
login	104	1509			9	102	1498	2	2
ntp_u	380	83470	380	3932	78036		1502		
other	7237	1648845	5632	43164	66760	1597	1537533	8	1388
private	110893		7366			102548		979	
pop_3	202	3835	79	1007	1313	123	1515		
red_i	1	11	1	3	8				
shell	112	1552	1	6	7	111	1535		4
snmp		287161		198339	87317		1505		
smtp	9723	220237	9598	106488	112001	125	1648		100
ssh	105		1			104			
telnet	513	19544	219	2524	12729	202	2420	92	1871
time	157	2324	52	575	148	105	1601		
tim_i	7	24	2	16	3			5	5
tftp_u	1	3104	1	10	44		3050		
urh_i	14	174	14	13	161				
urp_i	538	38890	537	26577	12295	1	18		
X11	11	535	9	159	230	2	144		2
bgp	106					106			
courier	108					108			
csnet-ns	126					126			
ctf	97					97			
sum_all	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 14: Intrinsic variable: SERVICES (part 1).

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
daytime	103					103			
discard	116					116			
efs	103					103			
exec	99					99			
gopher	117					117			
hostnames	104					104			
http_443	99					99			
iso-tsap	115					115			
klogin	106					106			
kshell	98					98			
ldap	101					101			
link	102					102			
mtp	107					107			
netb._dgm	99	2029		59	204	99	1766		
netb._ns	102					102			
netb._ssn	107					107			
netstat	95					95			
nnsp	105					105			
nntp	108					108			
name	98					98			
pm_dump	1					1			
pop_2	101					101			
printer	109					109			
rje	111					111			
remote_job	120					120			
sql_net	110					110			
sunrpc	107					107			
supdup	105					105			
systat	115					115			
uucp	106					106			
uucp-path	106					106			
vmnet	106					106			
whois	110					110			
Z39_50	92					92			
sum_all	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 15: Intrinsic variable: SERVICES (part 2).

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
S0	87007	1726452	51	164099	229559	86934	1331628	22	1166
S1	57	10908	54	6526	4206	2	4	1	172
S2	24	108	17	97	4	6	6	1	1
S3	10	517	7	461	41		1	3	14
OTH	8	1481362	1	281	1434527	7	46551		3
REJ	26875	300031	5341	66727	6874	21534	226430		
RSTOS0	11	5359		2	5357	11			
RSTO	579	6747	67	1547	422	465	4731	47	47
SF	378440	1253008	91709	857881	390125	284372	2476	2359	2526
SH	107	54061		31438	14634	103	7980	4	9
RSTR	903	118	31	17	19	867	78	5	4
RSTRH		8689		489	2390		5810		
SHR		1167		291	875		1		
sum	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 16: Intrinsic variable: FLAG.

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
avg	3025.00	887.92	1157.00	1361.86	92.78	2727.00	499.34	1.3E+5	4.5E+5
max	6.9E+8	6.9E+8	2.2E+6	9.0E+7	3.6E+6	6.9E+8	6.9E+8	5.1E+6	5.1E+6
min	0	0	0	0	0	0	0	0	0

Table 17: Intrinsic variable: SRC_BYTES.

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
avg	868.00	716.60	3384.00	2874.43	60.71	46.00	11.09	3.3E+4	2.1E+7
max	5.2E+6	1.7E+7	5.1E+6	1.7E+7	1.7E+7	8.3E+3	1.8E+4	5.2E+6	5.2E+6
min	0	0	0	0	0	0	0	0	0

Table 18: Intrinsic variable: DST_BYTES.

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
0	493999	4848485	97277	1129849	2089033	394301	1625696	2421	3907
1	22	42	1	7				21	35
sum	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 19: Intrinsic variable: LAND.

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
0	492783	4156321	97278	1129852	1436796	394301	1587126	1204	2547
1	268	690855			652237		38570	268	48
2		58		1					57
3	970	1150						970	1150
4		54		1					53
5		68		2					66
6		13							13
7		3							3
8		3							3
10		2							2
sum	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942
avg	2.5E-3	0.143	0	3.5E-6	0.312	0	0.024	0.507	0.354

Table 20: Intrinsic variable: WRONG_FRAGMENT.

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
0	494017	4848505	97277	1129842	2089028	394301	1625696	2439	3939
1	2	15		10	3			2	2
2	1	5		3	1			1	1
3	1		1						
5		1		1					
13		1			1				
sum	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 21: Intrinsic variable: URGENT.

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
0	490829	4844497	96736	1129302	2085598	392103	1625696	1990	3901
1	256	1129	122	89	1031	69		65	9
2	2192	506	41	51	453	2127		24	2
3	38	591	4	49	542	2		32	
4	112	445	97	39	394			15	12
5	51	257	35	34	223			16	
6	104	214	96	27	186			8	1
7	5	167	4	31	133			1	3
8		108		18	89				1
9	1	88	1	19	68				1
10	1	65		20	45			1	
11		68		25	41				2
12	2	59	2	22	37				
13		45		17	26				2
14	37	39	37	19	17				3
15	1	45		14	26			1	5
16	1	31	1	15	16				
17	2	26	2	7	19				
18	13	24	12	14	10			1	
19	23	18	18	12	6			5	
20	10	24	1	11	13			9	
21		16		9	7				
22	28	13	28	3	10				
23		6		2	4				
sum_all	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 22: Content variable: HOT (part 1).

B.2 Content variables

The following tables present the values obtained for each content variable: for all the database (all), for all normal connections (normal), for flood type connections (flood) and non-flood connections (noFlood).

Content variables: (10) hot in Table 22 and 23; (11) num_failed_logins in Table 24; (12) logged_in in Table 25; (13) num_compromised in Table 26; (14) root_shell in Table 27; (15) su_attempted in Table 28; (16) num_root in Table 29 and 30; (17) num_file_creations in Table 31 and 32; (18) num_shells_num in Table 33; (19) num_access_files in Table 34; (20) num_outbound_cmds in Table 35; (21) is_hot_login in Table 36; (22) is_guest_login in Table 37;

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
24	13	10	13		10				
25		1			1				
26		4		2	2				
27		6		1	5				
28	274	1			1			274	
29		1			1				
30	28		28						
31		2			2				
32		5			5				
33		2			2				
34		2		1	1				
36		2		1	1				
37		3			3				
46		1			1				
47		1		1					
50		1			1				
58		2			2				
59		1		1					
69		1			1				
sum	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 23: Content variable: HOT (part 2).

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
0	493958	4847555	97268	1129808	2089028	394301	1625696	2389	3023
1	57	948	5	27	4			52	917
2	3	14	3	13					1
3	1	5	1	5					
4	1	4	1	3	1				
5	1	1						1	1
sum	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 24: Content variable: NUM_FAILED_LOGINS.

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
0	420784	4134171	27339	425106	2083117	392088	1623418	1357	2530
1	73237	714356	69939	704750	5916	2213	2278	1085	1412
sum	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 25: Content variable: LOGGED_IN.

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
0	491797	4847995	97210	1129511	2088852	392172	1625696	2415	3936
1	2151	177	16	30	142	2129		6	5
2	24	175	16	161	14			8	
3	11	18	10	12	6			1	
4	16	15	9	10	5			7	
5	2	18	1	13	4			1	1
6	3	62	2	56	6			1	
7	2	10	2	9	1				
8		5		4	1				
9	1	6	1	5	1				
10		12		12					
11	1	5	1	5					
12	1	7	1	7					
13	1	9	1	8	1				
14		2		2					
16	1	2		2				1	
17		3		3					
18	1	5	1	5					
21	1		1						
22	1							1	
27		1		1					
38	1							1	
102	1		1						
238	1		1						
275	1		1						
281	1		1						
767	1		1						
884	1		1						
sum	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 26: Content variable: NUM_COMPROMISED.

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
0	493966	4848253	97255	1129847	2088768	394301	1625696	2410	3942
1	55	274	23	9	265			32	
sum	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 27: Content variable: ROOT_SHELL.

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
sum	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942
0	494009	4848244	97267	1129842	2088764	394301	1625696	2441	3942
1	6	250	5	10	240			1	
2	6	30	6	3	27				
3		2			2				
5		1		1					

Table 28: Content variable: SU_ATTEMPTED.

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
0	493436	4848118	96708	1129725	2088764	394301	1625687	2427	3942
1	233	126	230	117			9	3	
2	22		18					4	
3	3	14	2	1	13			1	
4	10	31	8	4	27			2	
5	12	26	12	1	25				
6	126	22	126	1	21				
7	1	20		2	20			1	
8		32		1	30				
9	167	24	167		23				
10		19			19				
11		21			21				
12	1	20	1	1	19				
13		21		1	20				
14	1	8		1	7			1	
sum_all	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 29: Content variable: NUM_ROOT (part 1).

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
15		10			10				
16	1	2			2			1	
17		1			1				
18		1			1				
19		3			3				
20		4			4				
21		2			2				
22		1			1				
39	1							1	
54	1							1	
119	1		1						
268	1		1						
278	1		1						
306	1		1						
857	1		1						
993	1		1						
513306		1		1					
sum_all	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 30: Content variable: NUM_ROOT (part 2).

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
0	493756	4846486	97045	1129464	2087417	394301	1625696	2410	3909
1	207	941	186	60	872			21	9
2	36	382	33	53	318			3	11
3		229		36	191				2
4	7	119		28	90			7	1
5	1	67	1	29	38				
6		69		25	44				
7	1	43	1	18	23				2
8	1	44	1	28	13				3
9	1	35	1	24	6				5
10	1	25	1	17	8				
11		27		22	5				
sum_all	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 31: Content variable: NUM_FILE_CREATIONS (part 1).

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
12	1	14	1	12	2				
13		12		9	3				
14	1	15	1	15					
15	1	16	1	16					
16	2	2	2		2				
20	1		1						
21	1							1	
22	1		1						
24		1			1				
25	1		1						
28	1		1						
sum_all	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 32: Content variable: NUM_FILE_CREATIONS (part 2).

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
0	493970	4846486	97235	1127964	2088928	394301	1625695	2434	3899
1	48	2031	43	1888	100		1	5	42
2	3	7		2	4			3	1
3		3		2	1				
sum	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 33: Content variable: NUM_SHELLS_NUM.

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
0	493567	4846764	96834	1129735	2087403	394301	1625696	2432	3930
1	424	1195	415	67	1122			9	6
2	25	397	24	31	362			1	4
3	2	99	2	10	88				1
4	1	43	1	2	40				1
5		18		4	14				
6	1	6	1	2	4				
7		2		2					
8	1		1						
10		1		1					
11		1		1					
12		1		1					
sum	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 34: Content variable: NUM_ACCESS_FILES.

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
0	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942
1									
sum	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 35: Content variable: NUM_OUTBOUND_CMDS.

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
0	494021	4847912	97278	1129694	2088719	394301	1625687	2442	3812
1		615		162	314		9		130
sum	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 36: Content variable: IS_HOT_LOGIN.

	all		normal			flood		noFlood	
	ref.	our	ref.	our	our2	ref.	our	ref.	our
0	493336	4847403	96907	1129798	2088030	394301	1625692	2128	3883
1	685	1124	371	58	1003		4	314	59
sum	494021	4848527	97278	1129856	2089033	394301	1625696	2442	3942

Table 37: Content variable: IS_GUEST_LOGIN.

	all		normal		
	ref.	our	ref.	our	our2
count2	332.286	187.894	8.163	11.199	235.181
srv_count2	292.907	122.781	10.936	16.868	238.611
serror_rate2	0.175	336.000	0.000	0.091	0.105
srv_serror_rate2	0.176	0.326	0.000	0.093	0.102
rerror_rate2	0.055	0.037	0.055	0.002	0.001
srv_error_rate2	0.000	0.000	0.000	0.000	0.000
same_srv_rate2	0.773	0.534	0.971	0.592	0.896
diff_srv_rate2	0.005	0.033	0.008	0.036	0.005
srv_diff_host_rate2	0.016	0.028	0.073	0.093	0.011

Table 38: Traffic variables: last 2 seconds (all, normal).

	flood		noFlood	
	ref.	our	ref.	our
count2	414.152	250.345	25.247	17.373
srv_count2	364.152	47.814	21.570	13.669
serror_rate2	0.219	0.804	0.012	0.251
srv_serror_rate2	0.221	0.776	0.011	0.281
rerror_rate2	0.055	0.109	0.020	0.001
srv_error_rate2	0.000	0.000	0.000	0.000
same_srv_rate2	0.724	0.027	0.904	0.265
diff_srv_rate2	0.004	0.065	0.004	0.242
srv_diff_host_rate2	0.002	0.003	0.012	0.013

Table 39: Traffic variables: last 2 seconds (flood, noFlood).

B.3 Traffic variables

The following tables present the average values obtained for each traffic variable: for all the database (all), for all normal connections (normal), for flood type connections (flood) and non-flood connections (noFlood).

Traffic variables from the last 2 seconds: (23) count2, (24) srv_count2, (25) serror_rate2, (26) srv_serror_rate2, (27) rerror_rate, (28) srv_error_rate, (29) same_srv_rate, (30) diff_srv_rate and (31) srv_diff_host_rate in Table 38 and Table 39.

Traffic variables from the last 100 connections: (32) count100, (33) srv_count100, (34) same_srv_rate100, (35) diff_srv_rate100, (36) same_src_port_rate100, (37) srv_diff_host_rate100, (38) serror_rate100, (39) srv_error_rate100, (40) rerror_rate100 and (41) srv_serror_rate100 in Table 40 and Table 41.

	all		normal		
	ref.	our	ref.	our	our2
count100	232.471	61.396	148.512	28.127	52.651
srv_count100	188.666	47.874	202.064	56.879	70.517
same_srv_rate100	0.704	0.530	0.646	0.619	0.872
diff_srv_rate100	0.004	0.040	0.000	0.055	0.008
same_src_port_rate100	0.000	0.000	0.000	0.000	0.000
srv_diff_host_rate100	0.001	0.038	0.000	0.142	0.007
serror_rate100	0.176	0.298	0.000	0.007	0.058
srv_error_rate100	0.000	0.000	0.000	0.000	0.000
rerror_rate100	0.053	0.048	0.048	0.028	0.003
srv_serror_rate100	0.176	0.278	0.000	0.004	0.040

Table 40: Traffic variables: last 100 connections (all, normal).

	flood		noFlood	
	ref.	our	ref.	our
count100	253.749	95.856	141.271	19.938
srv_count100	186.234	12.591	47.602	17.864
same_srv_rate100	0.720	0.027	0.379	0.382
diff_srv_rate100	0.005	0.070	0.006	0.285
same_src_port_rate100	0.000	0.000	0.000	0.000
srv_diff_host_rate100	0.001	0.007	0.005	0.030
serror_rate100	0.220	0.808	0.009	0.218
srv_error_rate100	0.000	0.000	0.000	0.000
rerror_rate100	0.054	0.121	0.000	0.001
srv_serror_rate100	0.221	0.776	0.005	0.223

Table 41: Traffic variables: last 100 connections (flood, noFlood).

C README: GureKDDCup database description

gureKddcup database link: `http://www.sc.ehu.es/acwaldap/`

C.1 Introduction

This document explains briefly the process we followed to create the database gureKDDCup[1]. It contains connections of KDDCup99 (database of UCI repository) but it adds its payload (content of network packets) to each of the connections. It will permit to extract information directly from the payload of each connection to be used in machine learning processes.

The Information System Technology (IST) group of Lincoln laboratories at MIT university under contract of DARPA and in collaboration with ARFL created a network. In this network, they simulated real traffic with normal and attack connections and they sniffed them with tcpdump (linux command). The experiment lasted 7 weeks of 5 days. The generated tcpdump files and ps outputs, log files... are known as Darpa98 database. (More information on the following website: `http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/index.html`).

After this simulation, they extracted connections from the tcpdump files and they represented them in a tabular dataset in UCI repository format. This way, instances of the dataset belong to connections. They extracted 41 attributes for each connection plus the class attribute. These attributes are divided in three main groups: intrinsic features (extracted from the headers' area of the network packets), content features (extracted from the contents area of the network packets), traffic features (extracted with information about previous connections). This dataset is known as KDDCup99. (More information on `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`).

We tried to follow the same steps followed to generate KDDCup99. With this aim, we processed tcpdump files with bro-ids (`http://www.bro-ids.org/`) and we got each connection with its attributes. Finally, we labeled each connection based on the connections-class files (tcpdump.list) that MIT provides.

C.2 Attributes

gureKDDCup (and gureKDDCup6percent) contains 41 attributes divided in 3 groups. The content of these attributes is described next:

C.2.1 Intrinsic attributes

These attributes are extracted from the headers' area of the network packets. See Table 42.

[1]This work has been done in the Automatic Classification and Parallelism (ALDAPA) group (`http://aldapa.eus`) of Computer Architecture and Technology (KAT/ACT) department of University of Basque Country (EHU/UPV). Contact email: Iñigo Perona `inigo.perona@ehu.es`

Num.	Name	Type	Description
1	duration	integer	duration of the connection
2	protocol_type	nominal	protocol type of the connection: TCP, UDP and ICMP
3	service	nominal	http, ftp, smtp, telnet... and other (if not much used service)
4	flag	nominal	connection status. The possible status are this: SF, S0, S1, S2, S3, OTH, REJ, RSTO, RSTOS0, SH, RSTRH, SHR
5	src_bytes	integer	bytes sent in one connection
6	dst_bytes	integer	bytes received in one connection
7	land	binary	if source and destination IP addresses and port numbers are equal then, this variable takes value 1 else 0
8	wrong_fragment	integer	sum of bad checksum packets in a connection
9	urgent	integer	sum of urgent packets in a connections. Urgent packets are packets with the urgent bit activated

Table 42: Intrinsic Attributes.

Num.	Name	Type	Description
10	hot	integer	sum of hot actions in a connection such as: entering a system directory, creating programs and executing programs
11	num_failed_logins	integer	number of incorrect logins in a connection
12	logged_in	binary	if the login is correct then 1 else 0
13	num_compromised	integer	sum of times appearance "not found" error in a connection
14	root_shell	binary	if the root gets the shell then 1 else 0
15	su_attempted	binary	if the su command has been used then 1 else 0
16	num_root	integer	sum of operations performed as root in a connection
17	num_file_creations	integer	sum of file creations in a connection
18	num_shells	integer	number of logins of normal users
19	num_access_files	integer	sum of operations in control files in a connection
20	num_outbound_cmds	integer	sum of outbound commands in a ftp session
21	is_hot_login	binary	if the user is accessing as root or adm
22	is_guest_login	binary	if the user is accessing as guest, anonymous or visitor

Table 43: Intrinsic Attributes.

C.2.2 Content attributes

These attributes are extracted from the contents area of the network packets based on expert person knowledge. See Table 43.

C.2.3 Traffic attributes

These attributes are calculated taking into account the previous connections. 9+10 attributes are divided into two groups: (1) time traffic features (2) machine traffic features. The difference between one group and the other is the mode to select the previous connections.

Time traffic attributes To calculate these attributes we considered the connections that occurred in the past 2 seconds. See Table 44.

Machine traffic attributes To calculate these attributes we took into account the previous 100 connections. See Table 45.

C.2.4 Class attribute

The 42nd attribute is the class attribute, it indicates which type of connections is each instance: normal or which attack. The values it can take are the following (view Table5): anomaly, dict, dict_simple, eject, eject-fail, ffb, ffb_clear, format, format_clear, format-fail, ftp-write, guest, imap, land, load_clear, loadmodule, multihop, perl_clear, perlmagic, phf, rootkit, spy, syslog, teardrop, warez,

Num.	Name	Type	Description
23	count	integer	sum of connections to the same destination IP address
24	srv_count	integer	sum of connections to the same destination port number
25	serror_rate	real	the percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in count (23)
26	srv_serror_rate	real	the percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in srv_count (24)
27	rerror_rate	real	the percentage of connections that have activated the flag (4) REJ, among the connections aggregated in count (23)
28	srv_error_rate	real	the percentage of connections that have activated the flag (4) REJ, among the connections aggregated in srv_count (24)
29	same_srv_rate	real	the percentage of connections that were to the same service, among the connections aggregated in count (23)
30	diff_srv_rate	real	the percentage of connections that were to different services, among the connections aggregated in count (23)
31	srv_diff_host_rate	real	the percentage of connections that were to different destination machines among the connections aggregated in srv_count (24)

Table 44: Time Traffic Attributes.

Num.	Name	Type	Description
32	dst_host_count	integer	sum of connections to the same destination IP address
33	dst_host_srv_count	integer	sum of connections to the same destination port number
34	dst_host_same_srv_rate	real	the percentage of connections that were to the same service, among the connections aggregated in dst_host_count (32)
35	dst_host_diff_srv_rate	real	the percentage of connections that were to different services, among the connections aggregated in dst_host_count (32)
36	dst_host_same_src_port_rate	real	the percentage of connections that were to the same source port, among the connections aggregated in dst_host_srv_count (33)
37	dst_host_srv_diff_host_rate	real	the percentage of connections that were to different destination machines, among the connections aggregated in dst_host_srv_count (33)
38	dst_host_serror_rate	real	the percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_count (32)
39	dst_host_srv_serror_rate	real	the percent of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_srv_count (33)
40	dst_host_rerror_rate	real	the percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_count (32)
41	dst_host_srv_error_rate	real	the percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_srv_count (33)

Table 45: Machine Traffic Attributes.

warezclient, warezmaster, pod, back, ipsweep, neptune, nmap, portsweep, satan, smurf and normal.

C.3 Payload

The payload is the content area of network packets, it is a sequence of ASCII characters. For each connection we generated 3 files with extensions: ".a", ".b" and ".c". Each of them contains accumulated payload sequences. In files ".a" we accumulated sent packets' payloads sorted by time. In files ".b" we accumulated received packets' payloads sorted by time. And finally in files with the extension ".c" we accumulated all packet payloads of the connection sorted by time.

C.4 The gureKddcup and gureKddcup6percent databases

The gureKdddcup database contains all the connections we matched with tcpdump.list: 2.759.494 connections. We provide information about these 2.759.494 matched connections. The tcpdump.list files contain 3.013.862 connections, therefore we matched the 92%.

The gureKddcup is too large to be used in any learning process. Most experiments with kddcup database are done based on the 10% of the database provided in UCI. The provided database contains 494.021 connections. It has flood, no-flood and normal connections. Analogously we decided to generate a reduced sample: gureKddcup6percent. The sample contains all no-flood attacks matched with tcpdump.list and a random subsample of normal connections matched with tcpdump.list. We disregard the pod attack because we haven't matched many connections of this type. Thus, at the end we have no-flood attacks and more or less 15% of the normal connections matched, selected randomly.

The distribution of attacks and normal connections in both databases is shown in the following table:

C.5 Structure of data

To better understanding of the next sections of this document, the connection_number format is explained: $XYN_5N_4N_3N_2N_1N_0$

- X: indicates the week from 1 to 7.
- Y: indicates the day from Monday (1) to Friday (5).
- $N_5N_4N_3N_2N_1N_0$: indicates the number of connection.

Therefore, when we download and decompress gureKddcup.tar.gz or when we download all parts of gureKddcup and we reassemble the Week4.tar.gz we get the following files: Week1.tar.gz, Week2.tar.gz, Week3.tar.gz, Week4.tar.gz, Week5.tar.gz, Week6.tar.gz and Week7.tar.gz.

In the same way, we obtain the following compressed files with gureKddcup-6percent: gureKddcup6percent.arff.tar.gz, Week1.tar.gz, Week2.tar.gz, Week3.-tar.gz, Week4.tar.gz, Week5.tar.gz, Week6.tar.gz and Week7.tar.gz.

	gureKddcup		gureKddcup6percent	
Connection type	Frequency	%	Frequency	%
normal	1,129,856	40.94432	174,873	97.798
attacks	1,629,638	59.05568	3,937	2.202
anomaly	9	0.00033	9	0.005
dict	879	0.03185	879	0.492
dict_simple	1	0.00004	1	0.001
eject	11	0.00040	11	0.006
eject-fail	1	0.00004	1	0.001
ffb	10	0.00036	10	0.006
ffb_clear	1	0.00004	1	0.001
format	6	0.00022	6	0.003
format_clear	1	0.00004	1	0.001
format-fail	1	0.00004	1	0.001
ftp-write	8	0.00029	8	0.004
guest	50	0.00181	50	0.028
imap	7	0.00025	7	0.004
land	35	0.00127	35	0.020
load_clear	1	0.00004	1	0.001
loadmodule	8	0.00029	8	0.004
multihop	9	0.00033	9	0.005
perl_clear	1	0.00004	1	0.001
perlmagic	4	0.00014	4	0.002
phf	5	0.00018	5	0.003
rootkit	29	0.00105	29	0.016
spy	2	0.00007	2	0.001
syslog	4	0.00014	4	0.002
teardrop	1,085	0.03932	1,085	0.607
warez	1	0.00004	1	0.001
warezclient	1,749	0.06338	1,749	0.978
warezmaster	19	0.00069	19	0.011
pod	5	0.00018		
back (flood)	2,248	0.08146		
ipsweep (flood)	15,760	0.57112		
neptune (flood)	1,526,643	55.32329		
nmap (flood)	1,995	0.07230		
portsweep (flood)	9,973	0.36141		
satan (flood)	31,411	1.13829		
smurf (flood)	37,666	1.36496		
TOTAL	2,759,494		178,810	

Table 46: Distribution of attacks and normal connections.

C.5.1 gureKddcup

The database gureKddcup contains 7 compressed files: Week1.tar.gz, Week2.-tar.gz, Week3.tar.gz, Week4.tar.gz, Week5.tar.gz, Week6.tar.gz and Week7.tar.-gz.

At the same time each of these compressed files contains 5 compressed files: Monday.tar.gz, Tuesday.tar.gz, Wednesday.tar.gz, Thursday.tar.gz and Friday.tar.gz. And at the end, within these files, we find 4 compressed files: a-matched.tar.gz, b-matched.tar.gz, c-matched.tar.gz and gureKddcup.list.tar.gz.

- gureKddcup.list.tar.gz: Contains information about every connection in the day. The first 6 attributes are: connection_number, start_time, orig_-port, resp_port, orig_ip, resp_ip (information to identify the connection). And the following attributes are the cited 41 attributes plus class. The file is in the "table" or tabular format.

- a-matched.tar.gz: There are accumulated sent packets' payloads of the connection sorted by time. These payloads match with gureKddcup.list connections. The name of each file is the connection_number with extension ".a".

- b-matched.tar.gz: There are accumulated received packets' payloads of the connection sorted by time. These payloads match with gureKddcup.list connections. The name of each file is the connection_number with extension ".b".

- c-matched.tar.gz: There are accumulated all packet payloads of the connection sorted by time. These payloads match with gureKddcup.list connections. The name of each file is the connection_number with extension ".c".

Thus, the three files of one connection_number belong to a specific connection in gureKddcup.list.

C.5.2 gureKddcup6percent

The sample gureKddcup6percent contains 8 files: gureKddcup6percent.arff.tar.-gz, Week2.tar.gz, Week3.tar.gz, Week4.tar.gz, Week5.tar.gz, Week6.tar.gz and Week7.tar.gz.

- Week1.tar.gz, gureKddcup-15percent.arff.tar.gz : Contains information about every connection. The first 6 attributes are: connection_number, start_time, orig_port, resp_port, orig_ip, resp_ip (information to identify the connection). And the following attributes are the cited 41 attributes plus class. The file is in the well-known Attribute-Relation File Format (ARFF).

- Week[1-7].tar.gz : Each week folder contains five compressed sub-folders: Monday.tar.gz, Tuesday.tar.gz, Wednesday.tar.gz, Thursday.tar.gz and Friday.tar.gz. And at the end, within these sub-folders, we find 3 compressed files: a-matched.tar.gz, b-matched.tar.gz, c-matched.tar.gz.

– a-matched.tar.gz: There are accumulated sent packets' payloads of the connection sorted by time. These payloads match with gureKddcup-15percent.arff connections. The name of each file is the connection_number with extension ".a".

– b-matched.tar.gz: There are accumulated received packets' payloads of the connection sorted by time. These payloads match with gureKddcup-15percent.arff connections. The name of each file is the connection_number with extension ".b".

– c-matched.tar.gz: There are accumulated all packet payloads of the connection sorted by time. These payloads match with gureKddcup-15percent.arff connections. The name of each file is the connection_number with extension ".c".

C.6 Downloading information

In this section we explain the structure, directories and files that we provide. If you fulfil the database request form in the website, we will send you the link where you can download gureKddcup database and gureKddcup6percent sample.

C.6.1 Complete database

We provide these databases in individual compressed files: gureKddcup.tar.gz (9.3GB) and gureKddcup6percent.tar.gz (4.3GB).

- The way to decompress files with the extension tar.gz is: tar xzf xxx.tar.gz

C.6.2 Divided database

Due to the large size of the data we also provide the databases in many split parts. As said before, the experiment of simulated network lasted 7 weeks and we analogously split the database following this approach.

Our structure of data is divided in 7 files, representing 7 weeks. Each file contains all payloads of a week. We considered downloading 1GB files is a reasonable size. In that sense, since the size of the 4 th week is more than 3GB, we split the Week4.tar.gz compressed file in 4 files running the command in Listing 51.

Listing 51: The command which splits a file into patrs of indicated sizes.

```
$ split -d -b 950M Week4.tar.gz Week4.tar.gz
```

We obtained the following files: Week4.tar.gz00, Week4.tar.gz01, Week4.tar.gz02, Week4.tar.gz03 of 950MBytes.

To reassemble these files to create Week4.tar.gz the following commands can be used:

In Linux (Listing 52):

Listing 52: Commands to join split files into a original file in Linux.

```
$ cat Week4.tar.gz* > Week4.tar.gz
or
$ cat Week4.tar.gz00 Week4.tar.gz01 Week4.tar.gz02 Week4.tar.gz03 >
Week4.tar.gz
```

In Windows (Listing 53):

Listing 53: Command to join split files into a original file in windows.

```
> copy /B Week4.tar.gz00 + /B Week4.tar.gz01 + /B Week4.tar.gz02 +
/B Week4.tar.gz03 Week4.tar.gz /B
```

C.6.3 Checking MD5 cryptographic hash function

You can check the integrity of the downloaded files to ensure that there has been no failure in the downloading process. With this aim, we provide MD5 cryptographic hash function for each file. Thus, when the files are downloaded, you can compare your file's MD5 with the provided one.

Options to obtain the MD5 of a file:

In Linux:

Listing 54: Commands to check the MD5 signature in Linux.

```
$ md5sum -b Week1.tar.gz or gureKddcup
or
$ openssl dgst -md5 Week1.tar.gz
```

In Windows:

Listing 55: Ways to check the MD5 signature in windows.

```
http://www.md5summer.org/
or
Firefox add-ons: https://addons.mozilla.org/eu/firefox/addon/12335
```