POLITECNICO DI MILANO

Facoltà di Ingegneria dell'Informazione

Corso di Laurea Specialistica in Ingegneria Informatica

COMPACT AND PRIVACY PROTECTED PACKET ATTRIBUTION

Relatore: Prof. Stefano ZANERO

Tesi di Laurea di:

Giuseppe MACRI'

Matricola n. 712384

Anno Accademico 2009–2010

Alla mia famiglia

Acknowledgments

I would like to spend a few words for those people who supported me through my academic thesis work and personal growth. I am very thankful for my parents who give me the possibility to join at Politecnico di Milano first and at Univerisity of Illinois at Chicago, two amazing experiences, and because they are near to me in any circumstance. Thanks to their support I learn to never give up as student as well as person because I should learn from the past to reach my goals and have a better future. A special thank is dedicated to my brother, Antonio, he is my guideline for most of experience and problem I had.

An important part of last period of my life is my friends. I would thank them for supporting each other in our studies at both Politecnico di Mialno and University of Illinois at Chicago. I can spend a great spare time with them out of university.

Some special words are due to my advisors, Prof. Jakob Eriksson from UIC and Prof. Stefano Zanero from Politecnico of Milan. Since my thesis work was developed at UIC, I would like to thank Prof. Eriksson for who provided a guidance in all areas of my thesis and project and besides university he is a great person. Unfortunately I had no enough time to spend with Prof. Zanero, but every time I was in trouble with any kind of problem he is available to talk and give a good suggestion. I would thank him for coming over in Chicago for the thesis defense and for support he gave during the presetation. I hope sooner or later I can enjoy travel around the

world as he does. Definitely if I didn't get the job before graduating at Politecnico di Milano I will spend more time with him to improve my thesis work.

Thank you everyone

GM

Summary

Passive network monitoring systems are very useful to perform maintenance, protection and control of network. There are two important issues with that kind of systems; the main important issue is user privacy. In fact they are privacy implications subject to data protection laws.

Recent research articles [1] [2] confirm privacy-sensitive data not only payload within packets but also header information. It is possible to extract data from statistical analysis of network traffic. Many solutions to the privacy problem have been proposed based on static anonymization [3] [4]. A drawback of these solutions is the definition of policies to regulate the anonymization.

All networking monitoring systems use a lot of storage in order to create logs of the network users. This process is done to achieve a high security level. With this system everything done by the user is stored therefore the owner of the network is able to track any user. Having this level of control is very useful when there is abnormal behavior of a user; it is possible to retrieve the identity of the client and at what time they performed abnormal behavior.

Using this particular technique requires the system to use a considerable amount of memory space; considering the amount of users using the network, privacy is the bigger issue. It would be better to a find solution for both problems to avoid losing both security and storage space.

Two new methods have been implemented to change the NAT assign-

ment port policy in order to create an assignment procedure to improve privacy and storage efficiency.

Contents

Acknowledgments **iii**

List of Abbreviations **xi**

1 Introduction **1**

2 Background and related work **6**

2.1 Network Address Translations (NAT) 6

2.1.1 Port Address Translation (PAT) 7

2.2 Types of NAT 8

2.3 Netfilter 10

2.3.1 IPTables 10

2.3.2 Netfilter NAT 11

2.3.3 Connection Tracking 12

2.3.4 Conntrack-Tools 13

2.4 PRISM Framework 13

2.4.1 Architecture 13

2.4.2 PRISM Behavior 15

2.4.3 Drawbacks of PRISM framework 15

3 Methodology and Implementation **17**

3.1 Environment Setup 18

3.2 Methodologies 18

3.2.1 Fixed Port Approach 18
3.2.2 Flexible Port Allocation 20
3.2.3 Privacy 23

4 Experimental Validation **24**
4.1 Experiments 24
4.1.1 Manipulation and evaluation 24

5 Conclusions and Future work **27**
5.1 Conclusions 27
5.2 Future work 27

A Implementation **29**
A.1 Fixed port technique 29
A.2 Flexible port technique 31

B Scripting **35**

List of Figures

2.1 Full cone NAT, image courtesy of Christoph Sommer (Wikipedia) 8
2.2 Restricted cone NAT, image courtesy of Christoph Sommer (Wikipedia) 9
2.3 Port Restricted cone NAT, image courtesy of Christoph Sommer (Wikipedia) 9
2.4 Symmetric NAT, image courtesy of Christoph Sommer (Wikipedia) 9
2.5 Netfilter structure components, image courtesy of Jengelh (Wikipedia) 10
2.6 Iptables data flow, image courtesy of Guillermo Grandes (Wikipedia) 11
2.7 Packet flow paths, image courtesy of Martin A. Brown . . . 12
2.8 PRISM Architecture, image courtesy of Georgios V. Lioudakis 14

4.1 Number of users versus parallel connections 09/26/2001 . . 26
4.2 Number of users versus parallel connections 10/05/2001 . . 26
4.3 Number of users versus parallel connections 10/14/2001 . . 26

List of Tables

3.1 UIC System requirements 22

List of Abbreviations

NAT	Network Address Translation
PAT	Port Address Translation
NAPT	Network Address Port Translation
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IANA	Internet Assigned Numbers Authority
DHCP	Internet Assigned Numbers Authority
PPC	Privacy Preserving Controller
UIC	University of Illinois at Chicago
OSI	Open System Interconnection
HTTP	Hypertext Transfer Protocol
FTP	File Transfer Protocol

Chapter 1

Introduction

A very important aspect of network architecture is management. The network management is a set of actions which perform operations, maintenance and administration in a defined network. Software used for network management are also called network-monitoring systems. The set of available monitoring systems is very large and every network owner can choose a different package based on needs and network size. An important aspect of monitoring systems activity is network traffic measurement. Behind bandwidth analysis, traffic measurement performs users' network activity reports (e.g. Local and Remote IP addresses, Port number or protocol and user name) [5]. Traffic monitoring is an important tool to support operation and management of the network architecture, to menage anomalies, to defend the network and users from any kind of possible attacks, intrusions. Traffic monitoring is also required to perform information to legal authorities with an accurate log for investigations [6].

Privacy-sensitive information is not limited to the content of network packet. In fact considering the possible techniques used to end-to-end encryption, from a privacy point of view the protection of packet payload is trivial. More information can be extracted from protocols' header with statistical analysis of network traffic. Most of solutions proposed use anonymiza-

tion of the traffic. Besides ethical issues, the privacy problem protection domain has been growing within legislation area. Many countries have already adopted laws such that they can ensure users' privacy and regulate personal data collection, processing and dissemination. Regarding European Union, there are several rules to preserve the personal data from not legal processing. Other countries, as U.S.A, provide several laws to protect user information.

This document will focus on possible alternate solutions to tracking users avoiding privacy issues and the loss of storage space performance. All considerations and developed techniques are considered within a network with NATed (Network Address Translation) machine. The thesis will discuss: NAT background, current privacy preserving techniques and will propose two different approaches and possible advantages of proposed techniques.

The goal of the project is to implement a new kind of network monitoring system inside a router or an access point with a few hundred KBs of available storage. The current work is part of a project named Xenonets, developed by Bits Lab at UIC. The thesis, as discussed in the above paragraph, is focused on NAT network; the feature of such kind of network is to be able to use one global IP address serving many users with different private IP addresses, but chapter two will introduce a better overview of NAT machine. Particular attention during the project was on two different aspects:

- System Storage Efficiency
- Users' Privacy

As mentioned above a network monitoring system requires a large amount of storage space in order to track users, especially in a network with a NAT machine. Usually, monitoring systems store information as relation

database records [7]. The monitoring systems have to track each user's session (local and destination IP, local and destination port, exchanged byte, timestamp, etc). Considering the current monitoring systems used in networks, privacy is one of the principal issues to preserve. Network monitoring systems are very useful tools because it is possible to perform many kinds of operations in the network (management, planning and security). Besides technical issues, privacy also has legal implications.

Before introducing a possible solution to the problem we need to know which parameters are essential in order to track a user, as well as be more storage space efficient. Considering the monitoring system used at University of Illinois at Chicago (UIC), the stored parameters for each user's session are: source IP and port, destination IP and port, timestamps, packet length, payload bytes, interfaces. Many of these parameters are needed in order to perform many network measurements as bandwidth quotas. When the users are behind a Gateway, as UIC students, the real parameters we need in order to retrieve a user are the following: outgoing gateway port, timestamp and private IP address.

The new techniques are not based only about the smaller number of saved information. The second part of the technique is changing the NAT assignment port policy in both techniques. The NAT port policy assignment is usually sequential; each time a new connection comes from an private IP, the NAT tries to route the connection through a new available port using incremental order; if there are no new available ports, the NAT, using the outgoing port reuse feature, can route the connection in an already used port by other local IP address. One important constraint of both techniques is: one IP address connections can be routed through a single port at time. Because we need this constraint to ensure a unique association between local IP addresses and outgoing ports, we need to know how many ports a local user needs. Two different approaches have been considered in order

to provide an efficient solution to the problem. Both techniques focus on port policy assignment changes in the NAT system. The following points are an introduction of both techniques:

- Fixed Port Range: assigns a range port to each user inside the network with a mathematical function;
- Flexible Port Allocation: to provide more flexibility with large networks, the technique assigns port based on real users' needs.

Many data sources have been used to determine the minimum requirements for the new system. Two types of experiments have been conducted for different purposes:

- compute the necessary space to store all data trace for any user;
- analyze the average number of parallel connections a user needs.

The first experiment has been done to compute the possible number of parallel connections a user usually performs. The data source is the website: *http://www.crawdad.org* [8]. Crawdad provides a wide range of data source types.

Both implemented approaches evidence two main advantages:

- space saving in order to monitor the users;
- users' privacy.

Chapter 3 will discuss the details of both techniques, but we can introduce it with a brief description.

The first technique uses a mathematical function to assign outgoing ports; with this method we don't need to store any information (IP address, port, timestamp) because it is possible to compute the inverse function in order to retrieve the IP address.

The second technique tries to route all possible connections made by a single IP address through one outgoing port, while it is possible; with this method the gateway logs the relation between the IP address and the outgoing port; all possible connections through the same port need one record within the log file. Considering the privacy, both techniques don't store privacy sensitive information, e.g. destination, data exchanged.

Chapter 2 will introduce an overview of the used tools to conduct the experiment.

Chapter 2

Background and related work

In order to learn more on the topic, this section introduces NAT principles and the tools used to simulate a network with a client and a NAT machine.

2.1 Network Address Translations (NAT)

NAT (Network Address Translation or Network Address Translator) as a theoretical concept is the translation of an Internet Protocol address (IP address) used within one network to a different IP address known within another network.

NAT is also used to refer to the real machine in charge of translating addresses from one network to another one or to many different networks. In the entire document we use gateway to refer to the real machine. Using the NAT is possible, for example, to map many local IP addresses into one public IP. The NAT machine is often used when only one external IP address is available and many different computers need to be connected to the Internet. This helps ensure security since each outgoing or incoming request must go through a translation process that also offers the opportunity to qualify or authenticate the request or match it to a previous request.

The core of a NAT is the NAT table. The NAT table is a map that provides all possible allowed translations between internal IP addresses and the available external IP pool.

2.1.1 Port Address Translation (PAT)

An important feature of the NAT is the port address translation PAT. It allows having many hosts within the private network working with a single public IP. This feature is also known as NAT overload. Changes made by PAT concern both send's private IP and port number. Different from NAT, PAT, operating on both port numbers and IP address, works on layer 3 (network) and 4 (transport) of the Open System Interconnection (OSI) Model.

The PAT works essentially on two protocols (TCP, UDP) because both contain IP address and port numbers. The PAT uses a PAT table to route incoming packets from the public network. The PAT table stores information to keep track of public and private port pairs. A pair of IP address and port is named socket.

The address and port translation is transparent to both private and public hosts. If a private host want to initiate a connection with an external host, it sends a packet with the external host as the destination and its own IP address as the source; the PAT analyzes the packet, it changes the source address and finds one port from a pool of available ports to change with the original one in the packet header. It forwards the new packet to the receiver. Translating fields of the packet, the PAT creates an entry in the translation table (PAT table) containing many records and each record has: IP address, original source port, translated source port, original destination port. In this way it is possible to route many different connections through a single port with different record fields. When an incoming packet approaches the PAT, the PAT analyzes packet fields and it retrieves from the table the correct

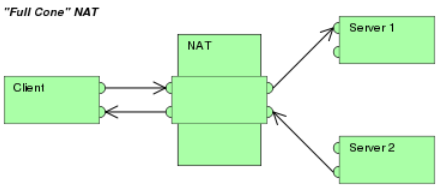

Figure 2.1: Full cone NAT, image courtesy of Christoph Sommer (Wikipedia)

IP address and port to forward the packet after replacing the original destination and port [9]. When a NAT uses the PAT feature is also called NAPT.

2.2 Types of NAT

A NAT can be classified in different way based on its own NAT table behavior:

- Full cone: an internal address and port is mapped to an external address and port, any packets from the same internal address and port will be sent through the same external address and port. Any external host can send packets to an internal address and port by sending packets to external mapped couple Figure 2.1.
- Restricted cone: an internal address and port is mapped to an external address and port, any packets from the same internal address and port will be sent through the same external address and port. An external host can communicate with an internal one only if the internal one has already initialized the connection with an port of the external address Figure 2.2.
- Port-Restricted cone: an internal address and port is mapped to an external address and port, any packets from the same internal address and port will be sent through the same external address and port.

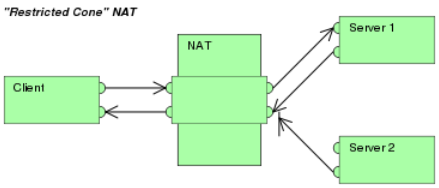

Figure 2.2: Restricted cone NAT, image courtesy of Christoph Sommer (Wikipedia)

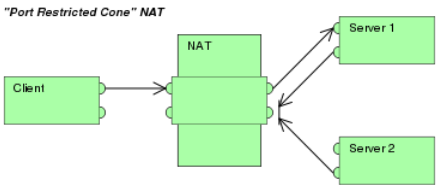

Figure 2.3: Port Restricted cone NAT, image courtesy of Christoph Sommer (Wikipedia)

A communication from an external host and port can exist only if an internal address has already sent a packet to the exact external address and port Figure 2.3.

- Symmetric: Each request from the same internal IP address and port to a specific destination IP address and port is mapped to a unique external source IP address and port. If the same internal host sends a packet even with the same source address and port but to a different destination, a different mapping is used. An external host can communicate only if the internal host has initiliazed the communication Figure 2.4.

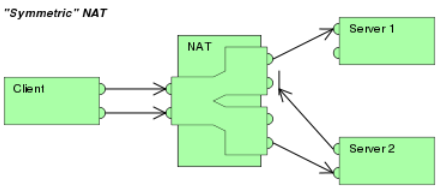

Figure 2.4: Symmetric NAT, image courtesy of Christoph Sommer (Wikipedia)

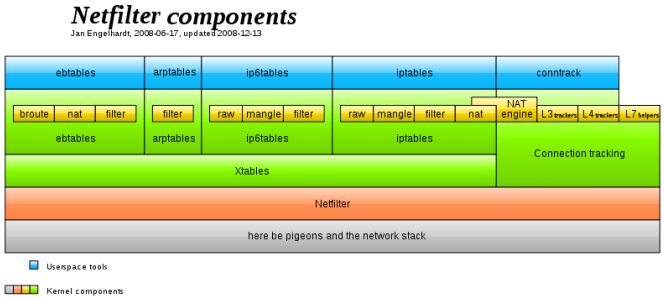

Figure 2.5: Netfilter structure components, image courtesy of Jengelh (Wikipedia)

2.3 Netfilter

As implementation of the NAT, the project is focused on Netfilter application. Netfilter is a framework able to intercept and manipulate network packets. Netfilter can create a firewall or NAT system within Linux machine. Netfilter is a modular framework able to extend itself with different and custom functions (hook). With this kind of approach it is possible to handle different protocol with different extended functions. In order to track connections between internal and external network, the main part of Netfilter is Connection Tracking [10].

Besides a complete stateful packet filtering, Netfilter can serve as NAT, perform port redirection, packet filtering and it can include rate limiting [11].

2.3.1 IPTables

Netfilter is able to handle many different tables from each different hook (modular function). All loaded tables have different features and serve different purposes.

Iptbles is a user space application for configuring Netfilter rules and packet filtering.

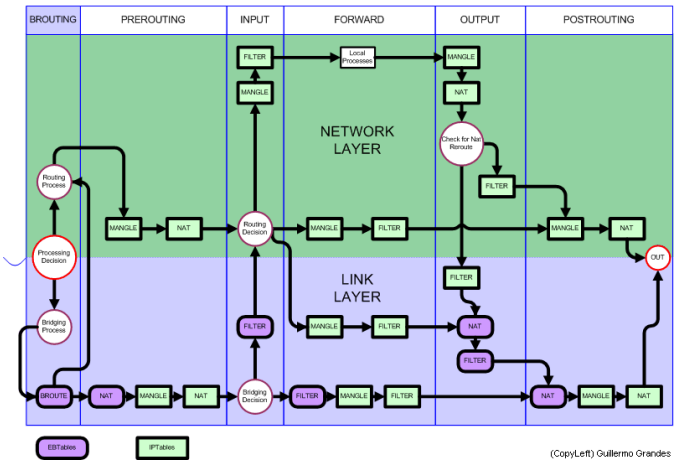

Figure 2.6: Iptables data flow, image courtesy of Guillermo Grandes (Wikipedia)

IpTables can be divided into 4 main modules as follows:

- Raw module: register a hook called from any other hook. It provides a table to filter packets before reaching Connection tracking;
- Mangle module: register a hook and mangle table after Connection tracking phase;
- NAT module: register two different modules, Destination and Source NAT. Destination transformations are applied before filter hook. Source transformations are applied after filter hook.

2.3.2 Netfilter NAT

Netfilter NAT is divided into two categories [12]:

- *Source*: NAT system alters the source address of the first packet of a connection. Source NAT is always performed post-routing, before the packets go out onto the wire. Special case is Masquerading.
- *Destination*: NAT system alters the destination address of the first packet of the connection. Destination NAT is always done before routing, when the packet first comes off the wire.

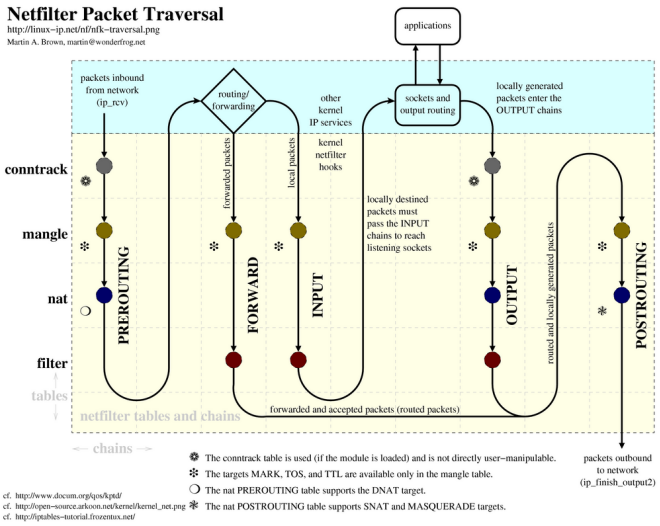

Figure 2.7: Packet flow paths, image courtesy of Martin A. Brown

2.3.3 Connection Tracking

One of the important features built on top of the Netfilter framework is connection track- ing. Connection tracking allows the kernel to keep track of all logical network connections or sessions, and thereby relate all of the packets which may make up that connection. NAT relies on this information to translate all related packets in the same way, and iptables can use this information to act as a stateful firewall.

The connection state however is completely independent of any upper-level state, such as TCP,Äôs or SCTP,Äôs state. Part of the reason for this is that when merely forwarding packets, i.e. no local delivery, the TCP engine may not necessarily be invoked at all. Even connectionless-mode transmissions such as UDP, IPsec (AH/ESP), GRE and other tunneling protocols have an, at least pseudo, connection state. The heuristic for such protocols is often based upon a preset timeout value for inactivity, after whose expiration a Netfilter connection is dropped.

2.3.4 Conntrack-Tools

The best way to interact with Netfilter information from userspace is the Conntrack-tools. This set of tools allows users to read and modify connection tracking entries and tables. The package includes a daemon tool conntrackd and a command line interface conntrack to handle connection tracking events. Conntrack-tools has been used to read information from NAT table and to log connections from internal to external host.

2.4 PRISM Framework

To find a solution to the privacy problem a new kind of framework has been developed by PRISM Consortium to handle both online and offline applications. The *PRISM* framework is two-tier framework architecture. The framework is the controller between source information and entities able to read data from network monitoring. Also PRISM provides a semantic model for access control policies able to work in real time and differently depending on the request type [13].

The access to privacy sensitive data is controlled with different levels of security. The architecture can handle different types of requests and directly contacts the data owner or send notifications to authorities. The framework gives the possibility to the users to be informed about data collection and processing. With a modular architecture, PRISM allows a high level of privacy but also a high level of security to access stored data.

2.4.1 Architecture

The main features for privacy-preserving system are the following:

- Real-time data protection (at the same time data is captured, it is preserved as well) on the online monitoring probes;

- Possibility to extend framework features with access and collecting data tool;
- Monitoring application customization to adapt the system to the entire network architecture;
- Entity separation to build a modular system in order do split data protection module from monitoring application module.

The architecture of the framework is not monolithic as the previous ones but is split in three different part as follows:

- Front-end: it captures and encrypts data;
- Back-end: it allows entities to access data;
- Privacy Preserving Controller (PPC): this controller is the Source of Authority to issue ACs to trusted holders. The PPC controls also the crypto keys to enforce the protection mechanism on Front-End and Back-End.

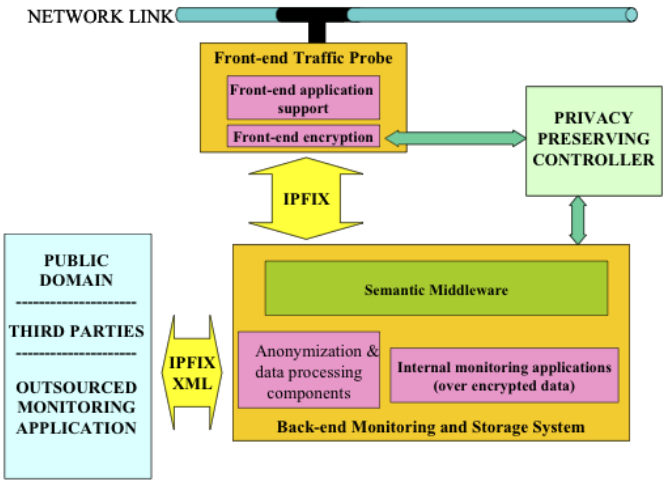

Figure 2.8: PRISM Architecture, image courtesy of Georgios V. Lioudakis

2.4.2 PRISM Behavior

The Front-end tier captures packets over networks links and the captured information is forwarded to the back-end tier already encrypted. The back-end tier provides access to data stored into a database. To access data each entity needs to submit a X.509 Role-assignment Attribute Certificate (AC).
The Privacy-Preserving Controller who provides certificates and implement active roles for different kinds authority controls the entire privilege system.

Data access

The Back-End tier provides the data access. It is in charge of controlling any data access (online and offline). All data is stored into a database to be available anytime an allowed user needs to read it. The Back-End tier can store information into the database using data already encrypted by the Front-end to ensure a high level of privacy.
No one can access directly data; because the data is already encrypted on the front-end tier it is also impossible for network operators to directly access the data.
To access data each entity needs to submit a X.509 Role-assignment Attribute Certificate (AC) to be allowed by the Back-end tier to read data [14].

2.4.3 Drawbacks of PRISM framework

PRISM framework is a good solution for users' privacy, but some features are not suitable to fit the goal of the project. We would implement the new system in a gateway router with a few hundreds of KBs of memory,

unfortunately the entire system requires a lot of storage space to run and as other network monitoring systems it requires a database to store a lot of information. The framework stores all information about users, even if data is not available to everyone, the privacy is not preserved.

The third section will discuss which kind of analysis has been conducted and the implementation in order to obtain desired goals.

Chapter 3

Methodology and Implementation

This chapter is focused on the methodology and implementation of the project. Before discussing the new developed techniques, we will introduce the basic concepts of the project.

The first basic concept is the outgoing port reuse; using this PAT feature we try to route all connections made by a single user through a single port, while it is possible. Unfortunately not all connections can be routed through a single port, e.g. parallel connections to the same destination (IP address and port). Because of that problem, one step of the project, the trace base study, is focused on learning how many parallel connections to the same host:port a user needs. With this study we can know how many outgoing ports a users needs.

The second basic concept is mapping IP addresses and outgoing ports. Because the goal of the project is to create a sort of monitoring system able to retrieve malicious users, we must know exactly which IP address is using a specific port in every moment. To handle this important requirement we impose each port has to be owned by a single IP address at a specific time.

The project is split into three parts:

1. trace-based study;
2. manipulation and evaluation;
3. implementation.

These new techniques will take advantage of the outgoing port reuse feature of the NAT. Compare the normal NAT behavior, routing many different users' connections to the same port, we route connections from only one IP address through a single port. This is a trade-off for our techniques but useful to realize a unique map between IP addresses and outgoing ports. Regarding these types of constraints it is important to maintain available ports for each user within the network.

3.1 Environment Setup

To test the new methods, a network with at least one client and the NAT machine is necessary. To avoid setting a real network with two machines, we use a virtualization software in order to set up a virtual network.
One of the virtual machines is the private host and the other the gateway machine. In the last machine IpTables is running in order to simulate the NAT [12].

3.2 Methodologies

In the project two methods have been implemented. Two different techniques have been used to satisfy different requirements and constraints.

3.2.1 Fixed Port Approach

The first implemented technique has been developed around a new assignment outgoing port policy of the NAT system. The main part of the technique is:

- NAT engine assigns a specific range of port per every user into the internal network.

The assigned range is defined by a mathematical function:

$$[x * y - 1, (x - 1) * y] \tag{3.1}$$

x are last IP number digits; y is the range size; to be sure that the range does not overlap ports from 0 to 1024, an offset has been added to the bounds.
As we introduced before, all possible connections made by an IP address are routed to a single outgoing port, except for parallel connections to the same host:port.
Not all ports have been assigned to internal users. The range from 0 to 1024 has been assigned to the NAT itself for communication reasons; an extra range of available ports, not assigned, has been computed for extra parallel connections needed by the users.
In such case, each time a port has been assigned within the additional range, a new record inside a log file has been created.
Retrieving an IP address from the outgoing gateway port needs to compute the reverse function of the above equation range equation.
The above system doesn't need to store information in order to retrieve a typical user, except the rare case that the NAT assigns a port within the additional range. The technique is efficient concerning storage space.
However the fixed port range method doesn't work well for networks with netmask less than *24 bits*.
Given a netmask of *16 bits*, it is possible to have 2^{16} different IP addresses and exactly 2^{16} outgoing available ports on NAT. In that case it is possible to assign a single port per user. This constraint is much too restrictive and this technique is useless in this type of network, for that reason we need to develop a more sophisticated technique.

3.2.2 Flexible Port Allocation

We propose a flexible port allocation technique, which handles with no problem, networks with a netmask smaller then *24 bits*. As a consequence of more flexibility the new technique is somewhat less efficient with respect to storage space.
Before going through the flexible method explanation, some terminology must be introduced:

- Connection Triple: source ip - destination - destination port;
- Outgoing Port Attributes: IP address - Timestamp.

The timestamp indicates the last time the port had been used and from which IP address. This technique, like the previous one, uses the outgoing port reuse feature of the NAT. The flexible method has been divided into the following parts:

- Initialization;
- Assigning outgoing ports;
- Port conflict resolution.

To retrieve a user with this technique the systems checks the log file to match timestamp and NAT port.

Initialization

At initialization time, when the NAT machine is starting, the NAT engine defines a limited outgoing port range. This range is available to internal IP addresses to create their own connections.

Assigning outgoing port

When a port is assigned to a user, the NAT stores the timestamp within the port attribute and a new record is created within the log file to track the

IP address with the current outgoing port.
Each time an IP address requests a new connection, the system tries to find a port already mapped with that IP address. If the connection parameters (connection triple) are different from the current one in that port, the new session can be routed through the previous port. When the connection is established through a port that has already been used by the same IP address the system doesn't record any information, because it already has information about the IP address and port association, but it updates the timestamp.

Port Conflict Resolution

The initial part of this method is to setup a small outgoing port range available to all users. The number of parallel connections to the same host and port or the number of IP addresses within networks can fill the available ranges. In the case that there aren't any available ports to route any new connections within the same triple or connection for a new IP address then there is a port conflict. There are two possible cases where a port conflict can exist:

- the available range size is smaller of the total number of available ports on the NAT;
- the available range size is equal to the total number of available ports on the NAT.

The first conflict case is very easy to handle; the NAT system doubles the range size in order to have additional available ports.
The second case is a little more complicated; if there aren't any available ports, the system tries to find the least recently used port. Once the system has located the port, it checks if there is a live connection. Netfilter can check connection status and also modify the status with the XTables struc-

ture. The Xtables contains the list of all connections through Netfilter. It is usually very rare to have a live connection through the least recently used port, therefore the system updates the port attributes (timestamp and IP address) and creates a new record within the log file.

Advantages

After complete implementation of a flexible technique, an evaluation has been conducted in order to understand the potential benefits. The simulation has been done on a tcpdump header file *010926.000002.packets* from [8] (716.9 MB). This file has been created from the University of Dartmouth during fall 2001 semester sniffing packets from 4 different building.
The study on the trace file counts how many parallel connections to the same host:port a user needs. Given that number we can compute how much space we need to track the user. Different from our system, the networking monitoring system of UIC needs to store all users' connections with more parameters.
3.1 shows the number of bytes the UIC system needs to store a single user's

Table 3.1: UIC System requirements

Field	Bytes
Source IP	4
Source Port	2
Destination IP	4
Destination Port	2
Timestamps	16
Headers + Payload Bytes	4
Payload Bytes	4
Interfaces (In,Out)	4

connection.

In the above file, we found 6055 users' session. Computing the total number of bytes to track a single user's session (40 bytes) times the number of sessions, the total amount of storage space needed is *337160 bytes*;
In our system, the fields needed are: IP address (4 bytes), Port number (2 bytes) and timestamp (4 bytes). With these parameters, the storage space needed for the same amount of sessions is *920 bytes*.
The storage savings for just one day is *99.727%*. If we consider more then one day trace, such that each IP address keeps the same port for a long period of time, the outgoing port reuse feature can arise the storage savings up *99.8%*

3.2.3 Privacy

As described both techniques track users without storing sensitive data, such as destination address, time and payload. Therefore these techniques ensure a high level of privacy for the users, while preserving the capability to identify the user responsible for any given packet.

Chapter 4

Experimental Validation

In order to determine how many ports a user needs, we study recorded traces from crawdad [8].

4.1 Experiments

We use the dataset dartmouth/campus (v. 2009-09-09). The dataset has been created at University of Dartmouth. All packets have been sniffed in three different periods: fall 2001 (four buildings), spring 2002 (five buildings) and fall 2003 (eighteen buildings). The data has been sanitized to remove information as Media Access Control address (MAC addresses). We study all samples from fall 2001 because more representative with less noise on the data.

4.1.1 Manipulation and evaluation

The goal of our trace-file analysis, as introduced above, is to learn how many parallel connections to the same host:port a user needs, such that we can know how many gateway outgoing ports a user needs.
Before analyze trace-file, we need to clean the data from unrequired fields and noise. Some of the trace files we got from crawdad were not complete,

e.g. synack with no finack and vice versa.
To clean the data we use t-shark such data we can print interesting packet header fields and analyze them with bash script.
The technique to count the above connections is the following:

- maintain a counter for each different triple *(source ip - destination ip : destination port)*;
- each time a *SYN ACK* with a particular triple is observed, increment the corresponding counter;
- each time a *FIN ACK* with a particular triple is observed, decrement the corresponding counter.

Figures 4,5 and 6 plot the number of parallel connections versus number of users on different days; we find that most of the users need one or two ports. This means that a typical user makes one or at most two parallel connections to the same host and port, so a user needs one or two outgoing gateway ports.
In all of figures we can see there are internal hosts using more than two ports, in that case the hosts are internal reachable from the external network (HTTP server and FTP).

Most of the users need one or two ports. Considering, the outgoing port usage that means a user makes one or at most two parallel connections to the same host and port.

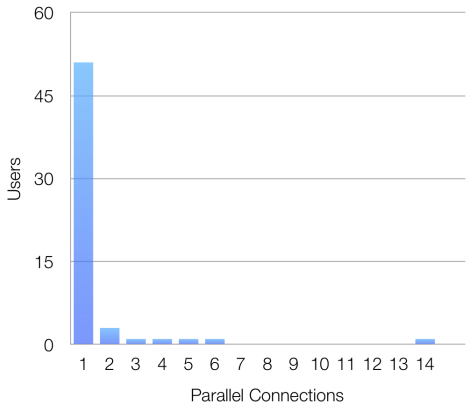

Figure 4.1: Number of users versus parallel connections 09/26/2001

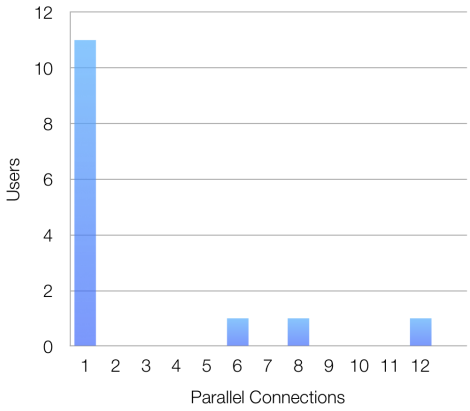

Figure 4.2: Number of users versus parallel connections 10/05/2001

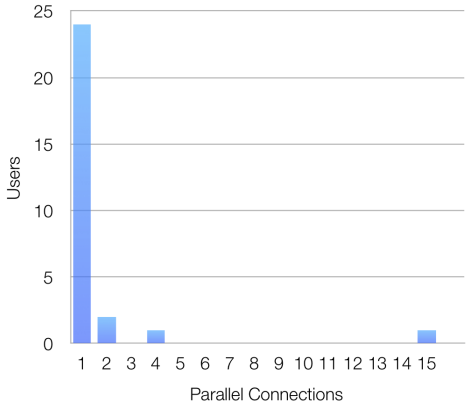

Figure 4.3: Number of users versus parallel connections 10/14/2001

Chapter 5

Conclusions and Future work

5.1 Conclusions

This thesis presented two techniques which allowed the monitoring of users with a high level of storage space efficiency; also they preserve user privacy through the network, and preserve a high level of privacy on users. The most flexible technique (useful with all network sizes) ensures storage savings at least of *99.727%* on our test trace.

Both techniques are transparent to clients within networks and are cheap to implement, due to the fact that they are simple source code modifications. It is possible to implement both techniques directly to an access point and/or a NAT.

5.2 Future work

The techniques have been implemented in Linux Ubuntu. The next step is the implementation into a real Gateway router box (e.g. *OpenWrt*). The software porting should be easy in this case because OpenWrt is a Linux based system with Netfilter implementation to provide a NAT system.
It is possible to integrate the complete system with the overall Xenonet ar-

chitecture under development at UIC.

Appendix A

Implementation

The Netfilter source code is built to handle either IPV4 and IPV6. The thesis focused on IPV4.

A.1 Fixed port technique

The *nef_nat_range* structure has been used in order to implement the assignment policy. The structure is defined into *net/netfilter/nf_nat.h*. With this structure it is possible to define the range in a dynamic way instead of the static way iptables does. The flags value specifies which type of range the NAT is dealing with. In order to have a port range with a specified IP and ports, the application has to set the flags value as follows:

```
1    flags = IP_NAT_RANGE_MAP_IPS | IP_NAT_RANGE_PROTO_SPECIFIED;
```

The values *min_ip* and *max_ip* is always the same; it is the IP address of the external NAT box.
The NAT engine has to modify to every user the value of *nf_conntrack_man_proto min, max*; these two fields define the minimum and the maximum port range. Using the equation 3.1, it is possible to determine the two values

concerning the IP address making the request.
In order to apply the range to the user, the source code file *net/nf_nat_core.c*, located in *net/ipv4/netfilter/*, has been modified. Focusing on function get_unique_tuple a new range has been created when the NAT modifies the packet, changing the source IP address and port.
Considering the client IP address of the internal network client is 192.168.0.2, the following steps have been added to the function:

- checking if the current packet belongs to TCP or UDP protocol and if the NAT is in the post-routing step;
- assign to the range the lower and upper port to define the range size.
- modify the range type with element flags assigning the value *IP_NAT_RANGE_MAP_IPS* OR (logic) *IP_NAT_RANGE_PROTO_SPECIFIED*.

The printk functions wasused to debug the program. The above changes are able to create and setup the range for the user. In order to use the policy: parallel connections to different hosts in the same port, modifying the function handling the port assignment is required. The assignment function is defined inside *net/ipv4/netfilter/nf_nat_proto_common.c*; the function name is nf_nat_proto_unique_tuple. The last part of the function is a for cycle to assign the correct port into the range. An element *unsigned *rover* is defined to enumerate all possible ports into the range. The element rover is a pointer to a *static u_int16_t tcp_port_rover;*. This value is defined inside *net/ipv4/netfilter/nf_nat_proto_tcp.c*. The rover stores the last offset of the used port. Instead of storing the last offset in the rover, the system resets that value every time to 0 such that the NAT routes the maximum number of possible parallel connections to different host in a single port.

A.2 Flexible port technique

The flexible technique requires to modify many files:

nf_nat_core.h

```
1  struct assignment {
2   unsigned int timestamp;
3   __be32 ip;
4   };
5   extern struct assignment *listPort;
6   extern unsigned availPorts;
7   extern unsigned offset;
8   extern __be32 current_ip;
9   extern bool limit;
```

The *struct assignment* contains the port attributes (timestamp and IP port); the *listport* is used to handle the different port range during the routing phase of the NAT; the *offset* is used to handle to avoid port overlapping between the port 0 and 1024. The variable *limit* is used to know when the number of available ports has reached 2^{16}.

nf_nat_core.c

```
1  /* Manipulate the tuple into the range given. For
       NF_INET_POST_ROUTING,
2   * we change the source to map into the range. For
       NF_INET_PRE_ROUTING
3   * and NF_INET_LOCAL_OUT, we change the destination to map
       into the
4   * range.  It might not be possible to get a unique tuple,
       but we try.
5   * At worst (or if we race), we will end up with a final
       duplicate in
6   * __ip_conntrack_confirm and drop the packet. */
```

```
7  static void
8  get_unique_tuple(struct nf_conntrack_tuple *tuple,
9                   const struct nf_conntrack_tuple
                        *orig_tuple,
10                  const struct nf_nat_range *range,
11                  struct nf_conn *ct,
12                  enum nf_nat_manip_type maniptype)
13 {
14  struct net *net = nf_ct_net(ct);
15  const struct nf_nat_protocol *proto;
16
17  .
18  .
19  .
20
21  bool response = proto->unique_tuple(tuple, range,
        maniptype, ct);
22
23  if( !response && maniptype == IP_NAT_MANIP_SRC) {
24
25   if(availPorts <= MAXPORT) {
26    struct assignment *tmp = kmalloc(sizeof(struct
          assignment)
27    * availPorts * 2, GFP_KERNEL);
28    if(!tmp) {
29         goto out;
30    }
31    memcpy(tmp, listPort, sizeof(struct assignment) *
          availPorts);
32    listPort = tmp;
33    availPorts *= 2;
34   }
35   else {
```

```
36          limit = true;
37
38    }
39  }
```

In this file the NAT tries to assign the port to the new connection. Input function parameters:

- tuple: modified tuple (source IP and port : destination IP and port) after NAT changes;
- orig_tuple: original tuple coming from the internal network;
- range: if defined into IpTables script, this parameter contains information about the range the new tuple should routed in.
- ct: this is connection tracking record inside Xtables;
- maniptype: specifies the manipulation type (destination o source manipulation).

If the value of *response* is true, the port has been associated, otherwise the NAT tries to double the available range if the available port number is less the the total outgoing ports otherwise the gateway tries to find the least used port always with the function *proto->unique_tuple*.

nf_nat_proto_common.c

```
1  if(limit) {
2   unsigned int min;
3   unsigned int index;
4   min = listPort[0].timestamp;
5   for(i = 1; i < availPorts - 1; i++) {
6    if (listPort[i].timestamp < min) {
```

```
7       index = i;
8     }
9   }
10  *portptr = htons(min + i);
11  if (nf_nat_used_tuple(tuple, ct)) {
12          return false;
13  }
14  return true;
15 }
16 for (i = 0; i < availPorts; i++) {
17  *portptr = htons(min + i);
18  /* Check if the port is used by the same user comparing
        the ip address */
19  if(listPort[i].ip == current_ip || listPort[i].ip == 0) {
20   if (nf_nat_used_tuple(tuple, ct)) {
21    continue;
22   }
23   listPort[i].ip = current_ip;
24   do_gettimeofday(&timestamp64);
25   listPort[i].timestamp = timestamp64.tv_sec;
26   return true;
27  }
28 }
```

In this part the systems check each available port owner, if the owner is the same current IP or the port is free, the NAT routes the connection through this port, it set the timestamp and the IP owner of the port and return true; otherwise if into the available range there is no free port, the system returns false. If the variable limit is set to true, the system finds the least used port and route the connection through this.

Appendix B

Scripting

To clean and analyze the trace data set and create charts, the following script are used:

```
1  mkdir report/AcadBldg16
2  for file `ls AcadBldg16`; do
3   tshark -nn -r $1 -T fields -e ip.src -e tcp.srcport -e
       ip.dst
4              -e tcp.dstport -e tcp.flags
5              '(tcp.flags.syn == 1 and tcp.flags.ack == 1)
6              or (tcp.flags.fin == 1 and tcp.flags.ack == 1)'
7              > report/AcadBldg16/$1.txt
8   awk '!x[$0]++' report/AcadBldg16/$1.txt >
       report/AcadBldg16/$1_clean.txt
9   awk '
10  /0x12/ {
11    syn_count++;
12    key=$1"-"$2"-"$3"-"$4;
13    syn[key] = 1;
14    chiave=$3"-"$1"-"$2;
15    par_conn[chiave]++;
```

```
16    print key"\tS "chiave" "par_conn[chiave];
17   }
18   /0x11/ || /0x19/ {
19   key=$1"-"$2"-"$3"-"$4;
20   if(syn[key] == 1  && fin[key] == 0) {
21    chiave=$3"-"$1"-"$2;
22    par_conn[chiave]--;
23    print key"\tF "chiave" "par_conn[chiave];
24    fin[key] = 1;
25   }
26   if(syn[invkey] == 1 && fin[key] == 0) {
27    chiave=$1"-"$3"-"$4;
28    par_conn[chiave]--;
29    print key"\tF "chiave" "par_conn[chiave];
30    fin[key] = 1;
31   }
32   }' clean.txt
33   report/AcadBldg16/$1_clean.txt | awk '{print $4}'
34                  | sort -n | uniq -c | awk '{print
                        $2"\t"$1}'
35                  > report/AcadBldg16/$1_graph.txt
```

The following is the IpTables script to run a NAT into a Linux machine:

```
1  #!/bin/sh
2  PATH=/usr/sbin:/sbin:/bin:/usr/bin
3  #delete all existing rules
4  iptables -F
5  iptables -t nat -F
6  iptables -t mangle -F
7  iptables -X
8  #Always accept loopback traffic
9  iptables -A INPUT -i lo -j ACCEPT
10 #Allow established connections,
```

```
11 #and those not coming from outside
12 iptables -A INPUT \
13                 -m state \
14                 --state ESTABLISHED,RELATED -j ACCEPT
15 iptables -A INPUT \
16                 -m state \
17                 --state NEW -i \
18                 ! eth0 -j ACCEPT
19 iptables -A FORWARD \
20                 -i eth0 -o eth1 \
21                 -m state \
22                 --state ESTABLISHED,RELATED -j ACCEPT
23 #Allow outgoing connections from the LAN side
24 iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
25 #Masquerade
26 iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
27 iptables -A FORWARD -i eth0 -o eth0 -j ACCEPT
28 #Enable routing
29 echo 1 > /proc/sys/net/ipv4/ip_forward
```

Bibliography

[1] M. Barbaro and T. Zeller Jr. A face is exposed for aol searcher no. 4417749. *The New York Times*, August 2006.

[2] S. Bellovin. A technique for counting natted hosts. *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement*, November 2002.

[3] D. Antoniades P. Trimintzios D. Koukis, S. Antonatos and E.P. Markatos. A generic anonymization framework for network traffic. *Proceedings of the 2006 IEEE International Conference on Communications (IEEE ICC 2006)*, June 2006.

[4] V. Paxson R. Pang, M. Allman and J. Lee. The devil and packet trace anonymization. *ACM SIGCOMM Computer Communication Review*, 36(1):29–38, January 2006.

[5] Wikipedia. Network traffic measurement. `http://en.wikipedia.org/wiki/Network_traffic_measurement`, June 2010.

[6] Francesca GAUDINO Lefteris KOUTSOLOUKAS George LIOUDAKIS Sathya RAO Fabio RICCIATO Carsten RICCIATO Felix STROHMEIER Giuseppe BIANCHI, Elisa BOSCHI. Privacy-preserving network monitoring: Challenges and solutions. 2008.

[7] Wikipedia. Comparison of network monitoring systems. `http://en.wikipedia.org/wiki/Comparison_of_network_monitoring_systems`, June 2010.

[8] David Kotz, Tristan Henderson, Ilya Abyzov, and Jihwang Yeo. CRAWDAD data set dartmouth/campus (v. 2009-09-09). Downloaded from http://crawdad.cs.dartmouth.edu/dartmouth/campus, September 2009.

[9] Wikipedia. Port address translation. http://crawdad.cs.dartmouth.edu/dartmouth/campus, June 2009.

[10] Harald Welte. Netfilter connection tracking and nat helper modules. `http://ftp.gnumonks.org/pub/doc/conntrack+nat.html`, June 2010.

[11] Wikipedia. Netfilter. `http://en.wikipedia.org/wiki/Netfilter`, June 2010.

[12] Rusty Russell and Harald Welte. Linux netfilter hacking howto. `http://www.iptables.org/documentation/HOWTO//netfilter-hacking-HOWTO.html`, June 2010.

[13] Anna Antonakopoulou Dimitra I. Kaklamani Iakovos S. Venieris Georgios V. Lioudakis, Fotios Gogoulos. Privacy protection in passive network monitoring: an access control approach. *International Conference on Advanced Information Networking and Applications Workshops*, 2009.

[14] International Telecommunication Union (ITU) Telecommunication Standardization Sector. *Information technology – open systems interconnection – the directory: public-key and attribute certificate frameworks*. ITU, August 2005.

[15] Georgios V. Lioudakis Aziz S. Mousas Dimitra I. Kaklamani Iakovos S. Venieris Fotios Gogoulos, Anna Antonakopoulou. Privacy-aware access control and authorization in passive network monitoring infrastructures. *2010 10th IEEE International Conference on Computer and Information Technology*, pages 1114–1121, 2010.

[16] Dirk GRUNWALD Paul OHM, Douglas SICKER. Legal issues surrounding monitoring during network research. *ACM SIGCOMM Computer Communication Review*, 2007.

[17] Francesca GAUDINO George LIOUDAKIS Dimitra L. KAKLAMANI Iakovos S. VENIERIS Giuseppe BIANCHI, Elisa BOSCHI. Legislation-aware privacy protection in passive network monitoring. *Information Communication Technology Law, Protection and Access Rights*, pages 363–383, 2010.

[18] European Parliament. Directive 95/46/ec of the european parliament and of the council on the protection of individuals with regard to the processing of personal data and on the free movement of such data. *Official Journal of European Communities*, (281):31–50, November 1995.

[19] Wikipedia. Network monitoring. `http://en.wikipedia.org/wiki/Network_monitoring`, June 2010.

[20] Wikipedia. Network management. `http://en.wikipedia.org/wiki/Network_management`, June 2010.

[21] Wikipedia. Network address translation. `http://en.wikipedia.org/wiki/Network_address_translation`, May 2010.

[22] andM.Pomposini G.Bianchi, S.Teofili. New directions in privacy-preserving anomaly detection for network traffic. *Proceedings of the 1st ACM Workshop on Network Data Anonymization (NDA 2008)*, October 2008.

[23] D. Sicker P. Ohm and D. Grunwald. Legal issues surrounding monitoring during network research. *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC '07)*, October 2007.

[24] Victor Castro. Roll your own firewall with netfilter. *Linux Journal*, October 2003.

[25] Nicolas Bouliane. Writing your own netfilter match. `http://www.linuxfocus.org/English/February2005/article367.shtml`, June 2010.

[26] Cisco Systems. How nat works. `http://www.cisco.com/application/pdf/paws/6450/nat-cisco.pdf`, June 2010.

[27] Netfilter Core Team. Netfilter. `http://www.netfilter.org`, June 2010.