Managing User Level Networking-Personal IP networks

Alessandro Pira Enrico Tassi Renzo Davoli
Department of Computer Science
University of Bologna – Italy
{pira,tassi,davoli}@cs.unibo.it

Author responsible for correspondence : Renzo Davoli, Department of Computer Science, University of Bologna. Mura Anteo Zamboni, 7. I-40127 Bologna. Italy. Phone +39 051 2094501, FAX +39 051 2094510, email renzo@cs.unibo.it.

Abstract

When a user logs in a multi user machine using User Level Networking-Personal IP he/she get assigned his/her own IP address or no address at all. In a multiuser multitasking operating system where several user are operating at the same time, each user has his/her own personalized access to the network. In this way the system and network administrator can assign access rights, traffic shaping, routing on a user-by-user basis. In case of user abuses on the network there is a direct mapping between the IP address and his/her user id; it is sufficient to read a single log file. In this paper we discuss also the problems related to design and manage ULN networks.

1 Introduction

User Level Networking, Personal IP, a.k.a. ULN [11], has the main purpose of giving the opportunity of assigning different IP addresses to different users also when they are working concurrently on the same multiuser computer.

The original idea of TCP-IP addressing scheme is to map one-to-one between IP addresses and network interfaces. This rigid mapping has had many exceptions: e.g. some net interfaces have several addresses for virtual hosting or the same IP address is round-robin assigned by the DNS to several servers for load balancing.

In ULN IP addresses are assigned also to users. A multiuser machine has its own IP addresses (normally one for each interface) used by all the machine level services (e.g. by daemons) and IP addresses assigned to users (normally one per user). Users cannot access the machine-level addresses: all the servers (bind), client (connect), or datagram traffic of a user on the network is originated at his/her own personal IP address (or at one of the addresses assigned to that specific user). Each user has the rights of using only addresses that have been assigned to him/her through ULN.

The basic idea of ULN consists of giving the user the opportunity to obtain an IP address through a DHCP-like process which requires authentication. The assigned address is added to the addresses of the NIC: with IPv4 [8] a new virtual interface (i.e. like ethX:Y) is created, while with IPv6 [2] addresses the assigned address is simply added to the list of IPs of the interface.

A similar approach has been implemented on DHCP to assign personal IP addresses to single-user personal computer, e.g. the Framed-IP-Address attribute in Radius authentication [9] or other DHCP authentication methods [3, 5]. The purpose is similar but in this latter case the mapping user-machine is trivially solved by the structure of the operating system: only one user at a time can access the computer. Up to our knowledge there are no other similar projects on multiuser machines.

Some efforts to give differentiated access to the network can be seen in the linux kernel development. The packet filtering architecture includes a owner match support option: it is possible to create iptables access lists to limit the access to the net using the process owner's UID as a parameter. This approach, with respect to ULN, is hard to manage and the rules are related to each single computer. To implement something similar to ULN there must be an access-list rule per user, per service and per computer (a total of no. of users times no. services times no. computer configuration lines), in case of

policy change the iptables of all the computers must be changed. ULN address assignment policies are on a single server. Different classes of users can be assigned different ranges of IP addresses and then access and routing policies can be defined on routers once for each different range.

So, a better control both over user network privileges and over the opportunity of tracking user actions over the network can be achieved,too.
The investigation on network abuses on the net is quite simple by using ULN: there is direct map between the address and the responsible user on the log of the ULN address assignment server. Currently the only way to look for responsabilities on multiusers hosts is to keep a very detailed log of command execution and port assignments. It is clear that a so detailed log creates also privacy issues.

Here are some examples of what ULN makes possible:

- deny network access to an user;
- assign statically, even in a LAN, the same IP address to the same user, no matter which computer he/she is using, in this case the user can operate on one computer at a time;
- assign statically, even in a LAN, to each user an IP address from a set of pre-assigned addresses for that user no matter which computer he/she is using, in this case the user can operate on multiple computer at a time;
- assign dynamically IP addresses to users: each user get assigned an address for an address space allocated for a class of users sharing the same network permissions, the mapping user-address is registered in a log file;
- different IP addresses can be used for virtual hosting; currently the only way to give IP level differentiated services on a single computer is the use of virtual machine: ULN is much lighter in terms of resource use;
- it's possible to have a privilege separation of "normal use" and "network access", each user can turn on and off his/her network interface. The user could also decide between several networking scenarios among those permitted by the configuration but this feature has not been implemented yet.

To summarize, ULN is suitable for the network administrator who needs an authentication mechanism for the network access.

Current prototype of ULN has been developed under GNU-Linux, and it is made by both a kernel patch[1] and code which must be executed at user-level.

All multiuser operating systems suffer from the problems solved by ULN, not only those Linux-Unix based. Services like remote execution or remote terminal in Windows NT/2000/XP allow multiple users to operate concurrently sharing the same network addresses and access. ULN can be used also as a security feature for network services daemons. Foreign code gets executed on a server in many cases: for effect of foreign agents, configuration code or marshalling code upload, or simply for bugs. In other cases local code gets executed under remote inputs, e.g. servlets or in general network services. Deamons try to limit the effects of the use and misuse ot that code by using sandboxes or file system cages. If the deamon runs as a service user (like inn news server or apache web server do, for example) ULN can be used as a cage for the network access implemented at kernel layer.

ULN is a way to prevent anonymous use of local resources but it is not a threat against privacy. Privacy must be not confused with insecurity. Each user must have all his/her privacy: it is possible to implement user privacy preserving protocols [1] over ULN and thus the user is anonymous on the Internet. On the contrary the system and network administrator must be able to locate the user that was using a specific address in case of misuse notices. Surprisingly, the use of ULN increases the level of privacy of users: each address is leased to a single user at a time. There is no need of keeping track of all the activities of each single user to locate which program originated a specific pattern of traffic. Using shared IP addresses and this latter method is the only procedure that can be used to find who was responsible for a network connection.

Clearly ULN generate the need of a larger address space of IP addresses. In IPv4 it can be used only by organizations having large class A or B address spaces or inside NAT/IP Masquerading domains. When ULN is used in a NAT/IP Masquerading internal network it loses some of its pros. For example when the network administrator receives a mail message from a collegue notifying an abuse it is not possible to map directly the responsability using a single local log file because from outside the network connections appears to be generated at the firewall. This problems are related to the scarcity of IPv4 32

[1] We originally used the stable tree of the Linux kernel: version 2.4.21; currently a 2.6.3 patch is available through CVS on sourceforge [11]

bit addresses. Using IPv6 all the problems related to the addressing space gets obsolete. Our prototype is already fully IPv6 compliant.

2 Architecture

The purpose of this section is to analyze how ULN works. The two main components of ULN are the kernel patch and the userspace utility suite, which includes client and server software.

The mechanism which inhibits the normal use of the network is obviously placed inside the kernel of the client machine, while the mechanism used to distribute IP addresses to the users is all made in user space and is composed by a client and server binary.

The userspace utility suite implements the DHCP style IP request mechanism, that is divided into two main modules, the remote server and the local client.

While the server, described in 2.2, must be installed only on one machine, the client must be installed on each computer the users have access to. We now describe how the user can ask for an IP address and how the client utility interacts with him.

2.1 uln-client

The only binary the end user must be aware of is *uln-client* for IPv4 and *uln-client6* for IPv6. We will use only *uln-client* in our description, since they work exactly in the same manner. We'll refer to a *network session* as the interval of time in which a user has the right to access the network with his personal IP address.

When the user wants to start a network session, he simply executes *uln-client* with no parameters, and a new IP is bound to him. *uln-client* life follows these steps:

1. contact the server and ask for a new IP address for the current user
2. inform the kernel that the current user is allowed to use this IP address
3. put on a virtual NIC like eth0:0[2] for the new IP address
4. wait until the user hits Ctrl-C
5. inform the server that the user ended his network session
6. inform the kernel that the user is no more allowed to use the IP address
7. bring down the virtual NIC [3]

Now we will describe how *uln-client* performs each step.

The first step seems problematic, since the server is remote. In a common network environment the user can contact a server and ask for a service, but with the new kernel semantic a user has no chances to open a socket, since he has no personal IP address to be redirected[4] to. The only user that can still use the real NIC IP address is root. *uln-client* is a setuid binary, and uses root's privileges in all steps except number four. With root's privileges it forks and executes *ssh* as the child and reads back from a pipe the personal IP address the child has negotiated with the server.

To ask the kernel to bind this IP address to the current user (step two), the client binary executes *uln-ifown* or *uln-ifown6* with root privileges. The same procedure is followed in step six.

Step three is simply done calling *ifconfig* with root privileges, and the same strategy is adopted for step seven.

Step five is completely made in the server side, since the server ends a network session when the connection falls.

2.1.1 Configuration file

uln-client has no runtime options, but reads a configuration file usually placed in `/etc/uln-client/uln-client.conf`. The configuration file has exactly this structure

```
SERVERNAME = 192.168.1.10
SERVEREXEC = /usr/bin/uln-server
IFACEPREFIX = eth0:
LOCKFILENAME = /var/lock/uln-client.lock
```

[2] *uln-client6* simply adds the new IP address to an existing NIC, since a network interface can have more than one IPv6 address assigned to it

[3] *uln-client6* simply removes the IP address from the real NIC

[4] Here redirected is used as a source redirection, since the user's sockets will be bound to his personal IP address, but the destination of a connect will not be altered

The first line is the only one the system administrator must change. It can be a name or an IP address and must identify the machine on wich *uln-server* is installed. `SERVEREXEC` is the command ssh will execute on the server machine, and should be left untouched. The name of the lock file can be altered, but the default setting is usually good. `IFACEPREFIX` is tipical of each computer, but most of client machines have only one NIC, called eth0.

2.1.2 Usage

uln-client takes no parameters, and must be started by the user who wants to begin a network session. The user may be prompted to enter his passord on the server machine. Since ssh supports public key authenticatin model, you can simply generate a couple of keys and let ssh do the authentication automatically. To end the network session the user must send a Ctrl-C to *uln-client*.

2.2 uln-server

The *uln-server* program has the function of managing the IP addresses. *uln-server*, which usually gives to the client an IPv4 address, can be used with the *-6* parameter to get an IPv6 address.

uln-server is executed, through ssh, from *uln-client*, In this way, the protocol security is left to ssh. Besides, *uln-server* runs with the privileges of the user who, on the client computer, executes *uln-client*. But, since *uln-server* needs to access the filesystem in a spool directory (which for security reason should not be accessible by everybody), the binary executable must be installed *setuid*, and it runs with the privileges of the special user *uln*.

2.2.1 Compile-time options

Some parameters of the server can be configured at compile time. These parameters are:

- the path to the configuration file (described in 2.2.2; the default value is /etc/uln-server/uln-server.conf
- the path to the shared data file; the default value is /var/spool/uln/uln-server.shared_data
- the log level which will be passed to syslogd; the default value is `LOG_LOCAL5`;
- the log options which will be used in the syslog() call; by default, only `LOG_PID` is used, but it's possible to have all logs printed also to *stderr* by specifiyng `LOG_PID|LOG_PERROR`.

Those parameters must be specified in the *Makefile*. Obviously, you'll have to recompile *uln-server* to modify those default values; it's not possible to do that with just the uln binary installation.

2.2.2 Configuration file

The run-time options can be specified by modifying the configuation file.

In the configuration file, the server can find the list of IP addresses and relative rights. Through this file the network administrator can configure both the IP addresses which are available for assignment and the users who have the necessary rights to access those addresses.

This is a sample configuration file:

```
BEGIN
   IP4_ADDR = 192.168.1.0/24
   USER_DENY = 1-400,badguy
END

BEGIN
   IP4_ADDR = 192.168.2.0/255.255.255.0
   USER_DENY = 1-400
END

BEGIN
   IP6_ADDR = fec0:000a::/64
   USER_DENY = ALL
   USER_ALLOW = ipv6user
END
```

Every section is delimited by a `BEGIN` line and a `END` line.

In every section there **must** be a line which specifies a list of IPv4 or IPv6 addresses. Those are addresses which will be available to the clients.

Optionally you can specify also a USER_DENY and/or an USER_ALLOW line. Those lines can be used to specify a list of users which can or cannot access the IPs specified in the relative section. In both lines the special token ALL can be specified to indicate all the users. Besides, more than one user can also be specified by using numeric values and minus to indicate a user range (i.e. 1-400 indicates all user IDs from 1 to 400 included), or comma (without space) to separate user names.

By default, an user is allowed to access an IP address. This means that all the users which should not get their IP address should appear USER_DENY list (this includes also "system" user like nobody or daemon).

In brief, an user is allowed if:

- he does **not** appear in any of these list

or

- he appears in the USER_ALLOW list

3 Examples

In this section we will present some scenarios, and solve the hilighted problems using ULN.

3.1 Computer Science Laboratory

Consider a computer science laboratory where all the computers are connected together through a local network, and are all visible from the internet, since they have a public IP adrress. All computers are usualy accessed both from the outside (a students that works at home may use an ssh connection to read his mail, check for news, talk with friends...) and from the inside (each computer has a keyboard too).

A laboratory like this one may work fine, except when there is an exam. The teacher, in this case, needs to prevent students to reach external resources, like a friend that has already passed the exam and is reachable trough the internet, and to communicate with each other through the LAN.

The easiest solution can be cutting out from the net all the computers by turning off the network hub/switch, but this may cause some problems, since every computer may be used by users logged in from the outside.

With ULN is possible to cut out from the network only the students involved in the exam, and let all the other users to continue working.

In order to do that, the system administrator should:

- prepare a script that will be executed on the machine running uln-server just before the exam starts; this script should send a SIGTERM to all the instances of *uln-server* owned by the students who are scheduled for the exam; this will revoke all the assigned IP addresses to the students;
- prepare a new uln-server configuration file, which will be used for the duration of the exam; in this file the students scheduled for the exam should appear in every USER_DENY list and be deleted from every USER_ALLOW list.

All computers will continue working regularly for all users, except for the students who are scheduled for the exam.

3.2 Malicious use of network resources

An university can have thousands of students, and some of them may abuse of the resources the university gives to them. Ususally these resources are enough to easily cause network attacks, for example a Denial of Service, and the university must respond of these acts to the victim. The main problem is to identify the author of the misuse. Commonly used techniques, like bsd process accounting, are usually enough, but with ULN it is easyer.

If an user commits an abuse from a ULN managed network, the network administrator just needs the IP address from which the abuse has been committed and the time in which the abuse has been committed. The couple (*IP address*,*time*) is enough to find out the misuser, since in a ULN network there is only one user that owns an IP at each time.

All the network administrator has to do is to open the system logs of the computer on which uln-server is in-

stalled, and search for the log line identifying the assignment of the IP address, starting from the time of the abuse and going backward. In the log line are specified both the username and the computer used to committ the abuse.

3.3 Free shell accounts

A company or university which wants to provide free shell accounts has to be very careful about the actions that users may perform with those accounts. In this case, denying network access to "free shell" users can be a good security policy. In this way, users cannot commit abuse through the network.

ULN can be used to implement that policy. With ULN, the system administrator can also define two class of users: "free shell" users, which don't have access to the network, and trusted users (like for example the system administrator himself).

In order to do that, the system administrator should setup an ULN server on a second computer[5], and install ULN in the computer used for free shell account.

The ULN server must be configured to deny every IP address to untrusted users, and to grant one to trusted users.

In this way, the machine providing free shells can be online and reachable, and at the same time untrusted users cannot use the machine to access the network.

3.4 Service-based traffic shaping

If, for some reason, a network administrator has the need to limit the network traffic produced by a determinate service, he/she can use ULN to do that in a more effective way.

Since now, in all the examples, a user was a real human being, but you can assing personal IPs to special users too. For example a system administrator may create a user *ftp*, that has a statically bound personal ip address. This gives the administrator some advantages.

[5] a different computer should be used for security reasons, but the *uln-server* software can also be installed on the "free account" computer

First, traffic shaping policy can be easily implemented at network level, since each service may have his own IP address. There is no need of inspecting the network layer of a packet to make decisions over it. This can be used both to simplify shaping rules and to improve performances of network filters.
This may help in virtualizing the network too. For example there is no easy way of installing more than one service of the same kind on a single machine. The only way is to use different ports (like starting two apache web servers on port 80 and 8080) , but your users must know this peculiarity to use the service. With ULN a single computer may hosts more than one service on their standard ports with their reserved IP addresses.

3.5 User-level traffic shaping

A network administrator may be asked to implement a network shaping policy based on users and not services. Some users may require a fast network network connection. This may be a good idea to optimize network utilization, discriminating users who really need an internet connection from users who may be pleased of having it. A personal IP can be used to implement this kind of policy using standard shaping softwares. Without ULN you have to discriminate fast and slow connections judging just the machine responsible of the conection, while with ULN the discrimination can be based on users.

3.6 Two (or more) virtual networks

A quite interesting networking scenario can be obtained with two different uln-server available on the same network (see Fig. 1).

The two server can be hosted on two different machine, or even on the same one. If the same computer will host both the server, each one should have its own configuration file, and its shared data file.

Besides, every machine on the LAN should have two different configuration file for *uln-client*, so there should also be two different binaries of *uln-client* on every computer. Each one will allow the user to obtain an IP address from one server.

In this scenario, two different virtual LANs can be configured. Each one can have its own configuration and can be managed separately.

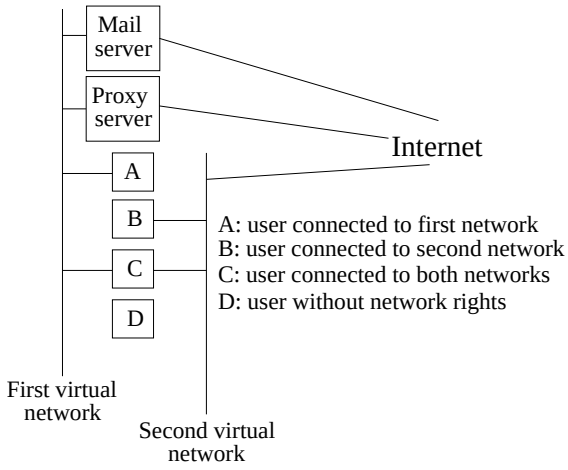

Figure 1: Two virtual networks example

The network administrator can grant access to user to only one network, none, or both.

This can lead for example to a scenario with four class of users:

- users which don't have network access rights;
- users which can access only the first-level network, which provides them a mail server and proxy server to surf the web; no direct access to computers outside of the LAN is possible;
- users which can access the second-level network, which grants a direct connection to internet, but no mail or proxy server;
- users which can access both networks: those users can use the mail and proxy server of the first network, and they can also have a direct connection to internet through the second network.

4 Security

In this section we want to analyze some architectural aspects of ULN that can lead to security risks. Main points are

- authentication
- denial of service
- local exploits on the client machine

The first security risk is authentication. This process is completely left to the *ssh*[10] package, that is a reliable and usually up to date package in every well administrated cluster.

A denial of service can be caused by both malicious and inattenting users. A user may request all available IP addresses if the server is not well configured. A distracted user may leave the computer, without killing *uln-client*. This can lead to the same denial of service, but a crontab script called *uln-client_maintenance*[6] can automatically drop unused virtual NICs. *uln-client_maintenance* checks for user's scheduled tasks, letting a user leave up his virtual NIC if he has some batch jobs to run.

As said before *uln-client* is a setuid binary owned by root. This is commonly considered a security risk, but the small size of the code and the impossibility for the user to directly interact with it should make the sysadmin sleep all the night long.

5 Conclusions and future developments

The prototype need an extensive testing phase to evaluate the usability, the user acceptance and the performance. From the first informal tests we are confident that there will be almost no loose in performace. We have also to test scalability: the kernel code has never been tested with hundrends of interfaces. The behavior of standard configuration commands like ifconfig or ip can be unsuitable for a so large number of interfaces. A hierarchical organization of network interfaces (maybe a virtual filesystem as /dev/if/eth0, /dev/if/uln/bill) could be useful. Moreover in this way standard file system related access control methods (chown, chgrp, chmod) can be used also for network interfaces.

The packaging of ULN for standard distributions is also a development of our project to make easier the installation for sysadm. We have already completed the .deb package for Debian [4].

The IP autoconfiguration can be another option to be included in the prototype: the user level IP address could be computed by a combination of an host-id and the

[6] the version for IPv6 is called *uln-client6_maintenance*

user-id. This method is easier to install than a central address assignment server but however there are several cons: the user-id or at least a group-id is encoded in the address and thus is readable from the net (privacy issue), change in user permissions must be operated host by host or the polocy must be included in a NIS or LDAP map.

6 Bibliography

References

[1] M. Tortonesi R. Davoli. User untraceability in the next-generation internet: A proposal. In *Proc. of Communications and Computer Networks (CCN)*, pages 177–182. IASTED, November 2002.

[2] S. Deering R. Hinden. Rfc 2460 - internet protocol, version 6 (ipv6) specification.

[3] T. Komori T. Saito. The secure dhcp system with user authentication. In *Proc. of 27th Annual IEEE Conference on Local Computer Networks (LCN'02)*, pages 123–131, November 2002.

[4] Debian web site. http://www.debian.org/.

[5] II Joseph W. Graham. Authenticating public access networking. In *Proc. of 30th annual ACM SIGUCCS fall conference on User services conference*, pages 247–248, 2002.

[6] King enzo ark programs. http://www.bononia.it/˜renzo/keap/.

[7] Homepage of the linux nis/nis+ projects. http://www.linux-nis.org/.

[8] Information Sciences Institute University of Southern California. Rfc 791 - internet protocol.

[9] Advanced radius user manual: Authentication and authorization. http://advancedradius.com/on_line_doc/Authentication_n_Authorization.htm.

[10] Openssh. http://openssh.org/.

[11] The uln web site. http://uln.sourceforge.net/.