Lappeenranta University of Technology
Department of Information Technology

Thesis for the Degree of Master of Science in Technology

Forcing Usage Rules in Public Wireless LANs

Radek Spáčil

Lappeenranta, May 10, 2002

The subject has been approved by the Departmental Council of the Department of Information Technology on 10^{th} of April, 2002

Supervisor:	JOUNI IKONEN, PhD.
Examiner:	Professor JAN VORÁČEK

Contact information: Radek Spáčil
Karankokatu 4 C 18
53810 Lappeenranta, Finland
spacil@lut.fi

Abstract

Lappeenranta University of Technology
Department of Information Technology

RADEK SPÁČIL
Forcing Usage Rules in Public Wireless LANs

Thesis for the Degree of Master of Science in Technology
2002

With rapid increase of popularity of wireless LANs and employing of this technology in large scale networks, a necessity of forcing usage rules arise. This thesis describes how to force users to follow usage rules in Public Wireless LANs. Problems discussed in the thesis are methods of revealing rogue DHCP servers and users using their own IP addresses, which are not issued by official DHCP server. Every methods considers also stopping such users to violate the usage rules. Central logging as a mean of acquisition of information necessary for these tasks is also addressed. The proposed solutions are custom made for the test network, but general ideas are applicable in any wireless or wired environment.

Thesis contains 55 pages, 22 figures, 4 tables and 4 appendices
Supervisor: JOUNI IKONEN, PhD.
Examiner: Professor JAN VORÁČEK
Keywords: wireless networks, WLAN, usage rules, DHCP server, remote logging

Tiivistelmä

Lappeenrannan teknillinen korkeakoulu
Tietotekniikan osasto

RADEK SPÁČIL

Käyttösääntöjen valvonta julkisissa langatomissa lähiverkoissa

Diplomityö
2002

Langattomien lähiverkkojen yleistyessä nopeasti suurten verkkojen teknologiana käyttösääntöjen valvonta tulee tarpeelliseksi. Tässä työssä kuvataan, kuinka käyttäjät voidaan pakottaa noudattamaan käyttösääntöjä julkisissa WLAN-verkoissa. Työssä käsiteltävät ongelmat koskevat menetelmiä epäluotettavien DHCP-palvelinten paljastamiseksi sekä omia IP-osoitteita käyttävien käyttäjien paljastamiseksi tilanteissa, jolloin IP-osoite ei ole viralisen DHCP-palvelimen myöntämä. Jokaisen menetelmän kohdalla pohditaan, kuinka tällaisia käyttäjiä voidaan estää rikkomasta käyttösääntöjä. Lisäksi pohditaan keskitetyn tietojen keruun hyödyntämistä kuvattujen tehtävien suorittamiseksi. Esitetyt ratkaisut on erityisesti suunniteltu testiverkkoa varten, mutta yleiset ideat ovat toimivia missä tahansa langattomassa verkossa.

Työ sisältää 55 sivua, 22 kuvaa, 4 taulukkoa ja 4 liitettä
Ohjaaja: TkT. JOUNI IKONEN
Tarkastaja: Professori JAN VORÁČEK
Hakusanat: langaton verkko, WLAN, käyttösäännöt, DHCP-palvelin, tietojen keruu

Abstrakt

Lappeenranta University of Technology
Department of Information Technology

RADEK SPÁČIL
Kontrola dodržování pravidel použitísítě ve veřejných bezdrátových sítích

Diplomová práce
2002

S rychlým nárustem popularity bezdrátových sítí a používáním této technologie pro rozsáhlé sítě, vzrostla potřeba kontroly dodržování pravidel používání sítí. Tato práce popisuje jak přinutit uživatele chovat se v souladu s těmito pravidly, obzvláště v prostředí bezdrátových sítí. Popsány jsou metody vyhledávání falešných DHCP serverů a vyhledávání uživatelů používajících vlastních IP adres, nevydaných oficiálním DHCP serverem. Každá popisovaná metoda zahrnuje také možnosti jak zastavit těmto uživatelům v dalším porušování pravidel. Práce se zabývá také problémem centralizovaného logování, jako prostředku pro získávání informací nutných pro výše zmíněné metody. Navržená řešení jsou ušitá na míru testovacímu prostředí ačkoliv hlavní myšlenky jsou použitelné v jakémkoliv síťovém (bezdrátovém) prostředí.

Práce obsahuje 55 stran, 22 obrázků, 4 tabulky a 4 apendixy
Vedoucí diplomové práce: JOUNI IKONEN, PhD.
Zkoušející: Profesor JAN VORÁČEK
Klíčová slova: bezdrátová, síť, pravidla používání, DHCP, vzdálené logování

Contents

List of Figures

List of Tables

List of Abbreviations

ADSL Asynchronous Digital Subscriber's Line

AP Access Point

BOOTP Boot protocol

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance

CSMA/CD Carrier Sense Multiple Access with Collision Detection

DES Data Encryption Standard

DHCP Dynamic Host Configuration Protocol

DoS Denial of Service

DNS Domain Name Service

DSA Digital Signature Algorithm

DSL Digital Subscriber's Line

FTP File Transfer Protocol

HTTP Hypertext Transfer Protocol

IP Internet Protocol

IPSec IP Security

ISO International Organization for Standardization

ISP Internet Service Provider

IV Initialization Vector

LAN Local Area Network

MAC Media Access Control

OSI Open System Interconnect

RARP Reverse Address Resolution Protocol

RSA Rivest-Shamir-Adleman

SQL Standard Query Language

ssh Secure Shell

SSL Secure Sockets Layer

TCP Transport Control Protocol

UDP User Datagram Protocol

VPN Virtual Private Network

WEP Wired Equivalent Privacy

WLAN Wireless Local Area Network

WWW World Wide Web

Chapter 1

Introduction

Since 1997, when IEEE introduced 802.11 standard—often referred to as *WLAN*—this technology encountered small interest. Only in few last years it gained higher popularity, especially in small business and enterprise networks, and nowadays these networks are experiencing rapid growth.

1.1 Public Wireless Networks

Short time ago, so called *free networks* or *community networks* (according to [1]) of large—metropolitan—scale started to appear. They are basically build up from nothing, only by contribution of members of the community network. They are setting up their own access points and joining the community, building up kind of free metropolitan network. This networks are popular mainly in Northern America [1], but also in Australia or Europe [2], [3].

Opposing to this free attitude, when actually nobody is the owner of the network—it's possible to look at it as kind of *symbiosis* of particular community members—there is a concept of *public wireless networks*. There is one owner of the network who provides (either free or paid) access to it although users can join freely.

This thesis is written as part of wireless metropolitan test network project, run by Lappeen-

ranta University of Technology and City of Lappeenranta. This test network falls into category of public wireless networks, which means there is an owner of network—the university.

1.2 Motivation and Goals

As in every community or organization which aspires to something more than anarchy, some rules how to behave should be defined. All members should follow the rules to ensure smooth functioning of the community. This aspect is often neglected in articles or book describing wireless networks ([1], [4]).

This thesis describes how users can be forced to follow these rules with emphasis on open character of the wireless network. Similar work was done by Valian & Watson [5]. Their work focused mainly to administration of the network environment, but the aspect of following defined *Usage Rules* was only of their marginal interest. Another aspect is differences between wireless and wired technologies, although in higher network layers of OSI model there are only minor differences.

Wireless technology and especially of large, metropolitan scale is quite hot issue nowadays. This work aims to contribute to this hype by a topic which is often neglected.

1.3 Organization of the Thesis

First brief theoretical overview of the Public Wireless Networks is given in Chapter 2. The Usage Rules, IEEE 802.11b standard and DHCP service are discussed there. Following chapters are conceived more practically, starting with the Logging issue in Chapter 3. Continuing by description of rogue DHCP servers in Chapter 4, to tracking illegal IP addresses in the network in Chapter 5. All these practical chapters have their separate discussion section.

The chosen solutions finally implemented in the project are discussed in Chapter 6. The whole thesis is then concluded in Chapter 7.

Chapter 2

Public Wireless Networks

Wireless LANs are encountering rising popularity. They are used mainly by companies and home users to quickly set up small networks without necessity of having wires around. Networks based on standard 802.11b are also used on public places as hotels, airports, etc.

Only recently several projects were launched aiming ambitiously to extend these networks to large-scale metropolitan networks. Such networks can one day became cheap alternative to cable modems and (A)DSL connections.

We can divide these networks to private and public ones. Private networks are already mentioned company and home networks. Users are not allowed to join freely, but have to be members of the community (company, family, . . .).

Opposing to this are *Public Wireless Networks*. These are open networks covering either small, middle size or wide publicly accessible areas. Anyone can freely join the network and use all the advantages this network offers (some of the services can be paid though).

In this chapter Usage Rules are discussed—what they are and what they are necessary for. Then brief look at IEEE 802.11b standard is provided, continuing up on the OSI layer model to WEP security protocol and finally to DHCP protocol. The latter is not part of 802.11 standard, but is also important for this thesis.

2.1 Usage Rules

In closed—private—networks there is an administrator who takes care about the network and configuration of computers in this network. Since public networks are open an often of large scale the administrator is unable to take care of configuration of all computers. Users by themselves are "administrators" instead in this case.

From this reason some rules—*Usage Rules*—have to be created which must be followed by users to ensure smooth, collision free running of the network. Since the project is still in test phase, there is no complete set of usage rules defined. Thus for this work, only the most important rules are considered. These are:

All IP addresses must be issued by official DHCP server. Users are not allowed to enter their own, fixed IP address. If someone wants to have fixed IP address, this should be set up via DHCP server.

Users are not allowed to offer DHCP service to the network. Although users can set up their own DHCP server for e.g. small home network and connect it through a gateway to public WLAN. In this case DHCP server must not work on WLAN interface of the gateway.

Users must use transparent proxy server for accessing the Internet. Users are not allowed to avoid using this transparent proxy. This proxy is used for redirecting different groups of users to different service pages, thus allowing to deliver users different information according to their status.

Other rules are to be added during test period of the network.

2.2 IEEE 802.11b Standard

This section briefly describes the 802.11b standard, sometimes referred to as *WLAN*. Knowledge of information presented in this section should be sufficient to understand rest of this work. However interested reader can consult [1] and [4], where the information was taken from. Or directly consult the specification [6], which can be found on web pages of Institute for Electrical and Electronic Engineers (IEEE).

In 1997 IEEE developed standard for wireless networks number 802.11. This standard defines *Medium Access (MAC)* and *Physical Layer (PHY)* for wireless LAN. Later this standard was updated and extensions 802.11a and 802.11b were derived from the common ground. Both extensions work in different frequency band, however the main features of both extensions are the same. In Europe only 802.11b is used.

2.2.1 Topology

The basic topology unit is called *Basic Service Set (BSS)* shown in Figure 2.1a. It consists of *Access Point (AP)*, which is in fact bridge connecting wired and wireless parts of the network, and several wireless stations. Before stations are allowed to access the wired part of the network, they have to associate themselves with the access point.

Several BSS connected together—in other words several access points on the same network segment—is called *Extended Service Set (ESS)*. Since main advantage of wireless devices is their ability to move, the 802.11b standard defines seamless mobility inside BSS and also ESS. The access points can exchange information allowing mobile stations to roam between BSSs. However there is no seamless roaming guaranteed by the standard between ESSs.

Another arrangement supported by 802.11b standard is called *Independent Basic Service Set (IBSS)* shown in Figure 2.1b. This unit consist from several wireless stations which can communicate among themselves. No access point is needed in this arrangement, since one of the stations can provide gateway services. This arrangement is also called *add hoc* or *peer-to-peer* mode, because it may connect only the stations without actually being connected to the Internet or other network.

(a) (b)

Figure 2.1: Basic network units defined in IEEE 802.11b standard: *Basic Service Set* (a) and *Independent Basic Service Set* (b)

2.2.2 Logical Architecture

Logical architecture follows ISO/OSI model and defines the lower two layers, in 802.11 language called *Physical Layer (PHY)* and *Medium Access Layer (MAC)* (with logical link control layer). The comparison is shown in Figure 2.2.

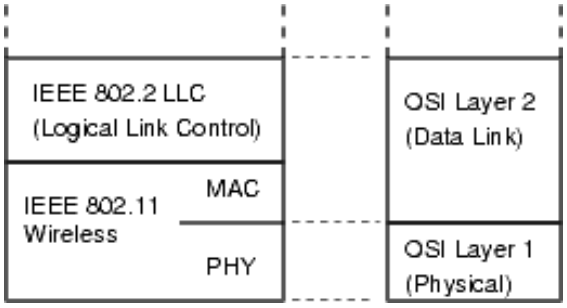

Figure 2.2: The IEEE 802.11 standard falls within the scope of layers 1 and 2 of the OSI Reference Model [4]

As seen in the figure, PHY corresponds to the physical layer whereas MAC corresponds to data link layer of ISO/OSI model. These two layers are further described in following paragraphs.

MAC layer. The goal of this layer is to provide access control functions for shared physical medium in support of higher layers (LLC). The 802.11 standard uses *CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance)* opposing the ethernet standard which uses *CSMA/CD (Carrier Sense Multiple Access with Collision Detection)*. It is not possible to receive and transmit on one channel at one time using radio transceivers, thus it is enough only to avoid collisions instead of detect them [4].

PHY layer. The 802.11b standard uses *Direct Sequence Spread Spectrum (DSSS)* as physical layer. DSSS encodes every bit by code consisting of 11 bits also known as *pseudo noise* or *chip* code as shown in Figure 2.3. Thus the transmitted signal has higher data-rate than the original data. This encoded signal has higher resistance to interference.

More detailed description of this technology as well as others used in different 802.11 specification can be found in [4].

Using DSSS network based on 802.11b can reach data-rates of 11 Mbps in 2.4 GHz band.

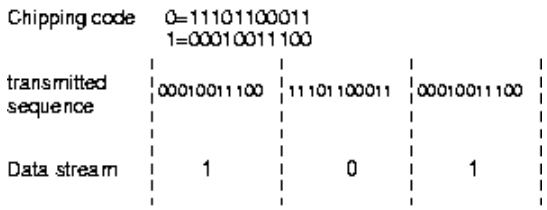

Figure 2.3: DSSS sends a specific string of bits for each data bit sent [4]

This frequency band is further divided into several channels as seen in table 2.1.

Table 2.1: 802.11b channel frequencies

Channel	1	2	3	4	5	6	7	8	9	10	11
Freq. [GHz]	2.412	2.417	2.422	2.427	2.432	2.437	2.442	2.447	2.452	2.457	2.462

To prevent interference, neighboring cells (BSSs) should use different channels. Actually, one channel uses 22 MHz of signal bandwidth, thus adjacent cells should be separated by at least five channels to avoid overlapping [1]. For example channels 1, 6 and 11 have no overlap. This is valid for ideal situation, in more crowded setting overlapping will became inevitable.

2.3 Differences Between WLAN and LAN from Security Point of View

The 802.11b standard employs *Wired Equivalent Privacy (WEP)* protocol to ensure similar level of security as in wired networks. The physical access to transmitting media (wires) in wired networks is limited—it is relatively difficult to connect physically to wired network. This involves entering the building and office with that network without being spotted by security guards or other employees.

In the case of wireless networks everyone have access to the radio transmissioned packets which are spreading often also outside of the intended coverage area. In case of public wireless networks this is even necessary—the publicly accessible areas should be covered by the signal. Unfortunately WEP does not ensure enough security—not even as much as wired networks have.

This is the main difference between LAN and WLAN and also the main source of problems

encountered in wireless environment. Recently there were published several studies of WEP protocol weaknesses ([7], [8] and [9])

WEP standard uses RC4 cipher to encrypt all packets going wireless and to provide authentication. The RC4 cipher uses *shared key* and *initialization vector (IV)* for encrypting packets. As the name *shared* implied the shared key has to be known to both sides—wireless station and the network itself (represented by access point). Using the shared key and the IV, RC4 generates *keystream*, which is then used to encode the packets (ordinary XOR function). To the result of encryption the IV is added in cleartext form and whole packet is sent. The receiver reads the IV, using shared key and just received IV generates the same RC4 keystream and decrypts the packet. This is possible because RC4 cipher generates always the same keystream provided the shared key and IV are the same. Whole process is illustrated in figure 2.4.

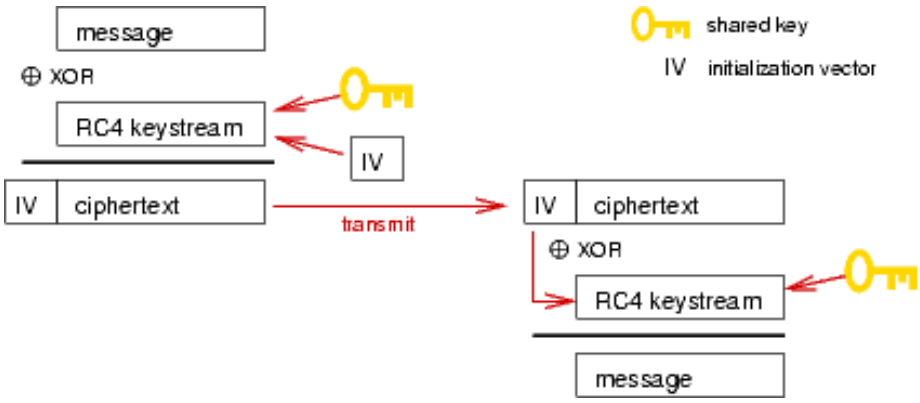

Figure 2.4: Process of encrypting and decrypting in WEP

Since the shared key does not change over long periods of time (the change requires to update the key in all stations using wireless network) possible attacker can passively listen on the network and collect packets with different IV. Then (s)he can use statistical methods to decrypt any packets on the network even without knowing the shared key. The detailed description of this procedure is described in [9].

Based on above mentioned technique it is not only possible to passively decrypt the traffic, but also to redirect packets or inject new packets into the network which will appear as authentic. It is clearly seen that standard security measures provided by WEP are not sufficient. Using of other security protocols as *SSL* (on application level) or *IPSec* and *VPN* technology (on lower layers) is strongly suggested.

2.4 Short Introduction to DHCP

This section describes *Dynamic Host Configuration Protocol (DHCP)*, which is essential tool in every mid to large-scale network. The knowledge of this protocol, at least at elementary level, is necessary to fully understand later part of this work.

DHCP is used for passing network configuration parameters to hosts on IP networks. It is used mainly for dynamic issuing IP addresses for the hosts. DHCP server keeps database of all issued and offered IP addresses. When new request came, server issues new IP or prolong the old one. The basic behavior is seen in Figure 2.5.

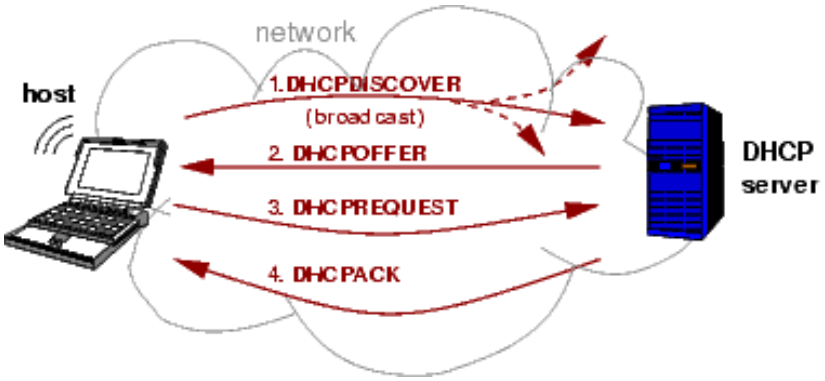

Figure 2.5: Basic DHCP communication

If a host does not have an IP address (e.g. after boot), it will send message: "I do not have an IP address, help me and give me some." (DHCPDISCOVER) to whole network as a broadcast. DHCP server answers with: "I am your DHCP server and I'm offering you IP address w.x.y.z." (DHCPOFFER). Host has to ask for this offered address: "I want w.x.y.z to be my IP address." (DHCPREQUEST) and finally DHCP server acknowledges this IP address: "All right, let w.x.y.z be your IP address for following time." (DHCPACK). This last message contains also time interval for which it is valid and other configuration parameters as default gateway, router, etc. This combination of issued IP address and time is called *lease*.

If client get more than one DHCPOFFER, it may choose the best[1]. Then subsequent DHCPREQUEST must be broadcasted in the same way as initial DHCPDISCOVER and it must be said which DHCP server client chose. This message is broadcasted to let the other DHCP servers know the client declined their offer and chose other one.

[1]The best for the client regardless on the server. Usually it is the first offer or the one client once already had.

Similar scenario applies if the host wants to renew its IP address. In this case the first two messages are omitted, and only DHCPREQUEST and DHCPACK are sent. In case that DHCP server is unable to serve the host with the same IP address it sends DHCPNAK which denies requested IP address and the host has to start the whole process once again from DHCPDISCOVER.

Here is a list of DHCP packets as defined in [10]:

DHCPDISCOVER client broadcasts to locate available servers

DHCPOFFER server to client in response to DHCPDISCOVER with offer of configuration parameters.

DHCPREQUEST Client message to servers either

1. requesting offered parameters from one server and implicitly declining offers from all the others,
2. confirming correctness of previously allocated address after e.g. system reboot, or
3. extending the lease on a particular network address.

DHCPACK Server to client with configuration parameters, including committed network address.

DHCPNAK Server to client indicating client's notion of network address is incorrect (e.g., client has moved to new subnet) or client's lease has expired.

DHCPDECLINE Client to server indicating network address is already in use.

DHCPRELEASE Client to server relinquishing network address and canceling remaining lease.

DHCPINFORM Client to server, asking only for local configuration parameters; client already has externally configured network address.

More detailed information about DHCP protocol can be found in [10].

Any "non-official" DHCP server, which can be assumed as rogue, can cause chaos in network IP space—some hosts will get IP from rogue DHCP server which can be out of "official" IP range, thus cause of network unavailability for such users. In worst case setting

up own rogue DHCP server can be origin of other attacks. Since DHCP can give also information about default gateways and routers, rogue DHCP server can denote attacker's computer as default router, and all traffic of affected hosts (those who have got IP from rogue DHCP server) will be routed to attacker's computer. Then it is easy to hijack the connection or act as man-in-the-middle which can lead to reveal of user's passwords.

Chapter 3

Logging

Even in the simplest computer or network there is a lot of different services and programs running which can generate lot of different messages: debug, warning or error messages, when someone is accessing the computer or in any occasion alike.

These messages are very important source of information about the computer or network. With these information it is possible to find out who or what caused problems on the computer, who did something unwanted or who braked into the database. Logging is essential part of this thesis, because methods described in later chapters are using the information collected by logging methods presented in this chapter.

It it impossible for administrator to watch all these messages in real time. Therefore usually the programs have interface to write these information into a file which is generally called *log* and the process of writing messages to this file is called *logging*. Administrator can look at logs later or use automated searching for patterns which look "suspicious".

Even if it will not be possible to stop the attacker to compromise the computer, all his steps will be logged in a file. On the other hand these log files are also possible to tamper in such a way it will look nobody did anything wrong. Thus it is not possible to rely completely on log files. Log files are very common target for tampering or deleting. There are methods to minimize this risk, but the threat will be still present. One of these methods is to store the log files in addition on different computer.

To have duplicated logs stored on independent server where it would be safer and where

it would be easy to backup them, a separate log computer should be dedicated for this purpose—*log server*.

3.1 Remote Logging—Log Server

The idea behind remote logging is to dedicate one computer as a *log server*. This computer should not run other services, ideally it should be "black hole" which is not transmitting any information, only receiving them[1]. This would make impossible to establish any TCP connection and break in using any protocol based on TCP. Unfortunately often TCP access to the log server is needed and thus other security measures has to be applied to withstand attacks.

Log server stores all the data which can help to identify possible network problems or attacks. Thus these data should be the most protected. If an attacker is able to tamper also logs stored on dedicated server, there is no other way how to reveal him. The network structure can look like in Figure 3.1.

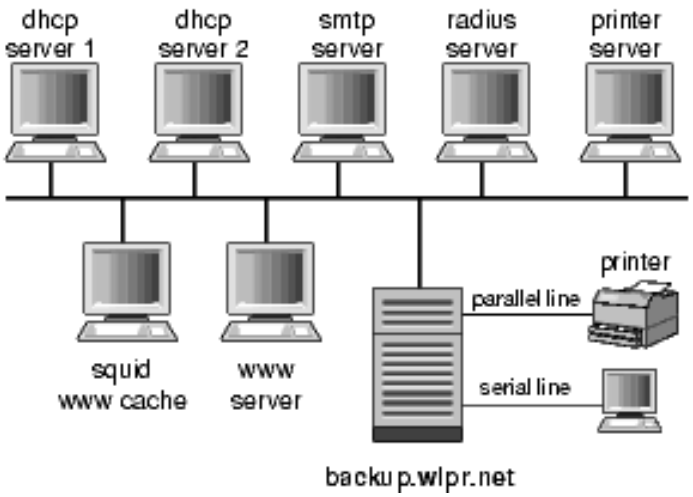

Figure 3.1: Network structure and dedicated log server

Even more effective is alternative with more than one log server or with logging onto another computer connected with log server via serial line as depicted on figure 3.1 and/or critical messages print on printer. The key issue is to have more then one copy of the logs but still have them centralized for easy administration.

[1]This is possible since *syslog* uses UDP which is connectionless, thus server does not need to answer.

3.2 *Syslog* Daemon

Syslog is daemon which listens on socket (standard */dev/log*) and writes messages coming there into a regular file, remote machine, terminal or console or send them to a named pipe. All this is based on configuration file (default */etc/syslog.conf*). The hostname, date and time are logged along with the message. The services are writing to this sockets by calling special system function `syslog()` or shell command *logger*.

Messages are divided into several facilities and levels. These are combined together and referred to as *facility.level* pair. The facilities list with description of programs they are used by is following [11]:

auth (or security—deprecated synonym) programs asking for login/password, authorization systems, e.g. *login*, *su*, *getty*, ...

authpriv private authentication messages,

cron messages from *cron* daemon,

daemon other daemons which doesn't fit into other categories,

kern reserved for messages from kernel,

lpr messages from printing system,

mail messages from mail system e.g. *sendmail*, *postfix*, *exim*, ...

news messages from news system,

syslog reserved for *syslog* itself,

user messages from normal users' processes,

uucp messages from UUCP system,

local0 to local7 reserved for local use.

Valid levels are (in descendant order according to the importance) [11]:

emerg (or panic—deprecated synonym) panic situation (normally broadcasted to all users),

alert conditions that should be corrected immediately, e.g. corrupted database,

crit critical conditions, e.g. hard device errors

err (or error—deprecated synonym) errors,

warning (or warn—deprecated synonym) warnings,

notice conditions which are not warnings, but should be handled specially,

info informational messages,

debug debug messages, used normally only when debugging programs,

none does not send messages of given facility; e.g. `*.debug; mail.none` will send all debug messages except coming from mail system.

When logging messages of particular level, also higher (more important) levels will be logged. For example logging `mail.notice` will cause to log not only messages from *mail* system of importance *notice*, but also all higher levels (`warning`, `err`, `...`, `emerg`).

Syslog reads this messages from three sources: */dev/log*—unix socket used by running programs, */dev/klog*—special device used by kernel for messages, and UDP port 514—network socket used for remote logging.

3.2.1 Services logging via *Syslog*

As stated above, different services are logging according the type of messages rather then according the service itself. Therefore administrator can setup *syslog* for storing authentication messages in one file while kernel messages to another.

To ensure maximum security is good to log remotely all levels from all facilities. The only limiting factor is disk space on log server and network bandwidth. If the volume of the logging messages needs to be cut down, some restrictive measures has to be applied.

To set up the *syslog* to provide remote logging facility, is enough to add following line into */etc/syslog.conf* file

```
*.*             @10.1.3.11
```

Note that it is possible to use host name instead of IP address. IP address is preferred since DNS attack will not affect it. After adding the line to the file restart *syslog*:

```
# /etc/init.d/syslog restart
```

Now *syslog* will log everything as before to different files according previous configuration and also to remote server, where it will further divided into different files (based on the log server configuration).

3.2.2 Other Services

Some of the services allows to log via *syslog* at least partially (e.g. error messages, while access logs are logged into file). In following paragraphs few of such services and their configurations are discussed. They were chosen according use in the test network and their relevance to wireless environment. This information was found in appropriate documentation.

Apache—www server

Apache is open source secure, efficient and extensible server ported to many platforms (including Unix and Windows NT) following current HTTP standards. It is the most popular web server in use since April 1996 [12].

Apache is able to log error messages via *syslog*. In fact it is able to log the same way also access logs[2], but this is not advisable because of possibly high volume of logged data.

To set up Apache to log error messages via *syslog* add following line to *httpd.conf* (usually located in */etc/httpd/conf*)

```
ErrorLog        syslog
```

This will cause *apache* to log via *syslog* facility *local7* by default. If the facility should be changed, the text `:facility` should be added to `syslog` in the configuration file (thus fi-

[2] *Apache* has ability to send access logs to named pipe, which can lead to *syslog*.

nally there will be `syslog:facility`), where facility is the one to be used. Original line can be deleted if logging only via *syslog* is required, otherwise *apache* will log in both ways.

The next step is to set up *syslog.conf* to log above mentioned facility to a file. If the above mentioned remote logging is in place this will be logged to remote server, but it is good idea to have logs stored also locally, thus to have two copies for comparison.

If also other information is necessary to log (e.g. access log) via *syslog*, consult [13]. There are described other methods *apache* can use for logging, e.g. logging through pipe, conditional logging and further customization. Description of these is beyond the scope of this work.

Squid—www proxy server

Squid is an open source full-featured Web proxy cache designed to run on Unix systems. It supports proxying and caching of HTTP, FTP and other protocols. Among others *squid* supports SSL and cache hierarchies. For more information consult [14].

With *squid* the situation is much simpler. It is enough to start *squid* with command line option *-s* [15].

```
# squid -s
```

Squid will start automatically log error messages to *syslog*. *Squid* does not allow to log access messages via *syslog* which is not wise anyway (due high volume of messages).

It is advisable to change start up scripts (usually located in */etc/init.d/* or */etc/rc.d/init.d*) so that *squid* will be always started with this parameter.

Radius server

Cistron Radius is according to its authors "an authentication and accounting server for terminal servers that speak the RADIUS protocol"[16]. Radius protocol is used for (centralized) authentication of users accessing servers where authentication is required.

Situation with *radius* is similar to *syslog*. It is possible to start *radius*[3] server so it will additionally log via *syslog*. The command line switch is *-g facility*, where facility is *syslog* facility which should be used

```
# radiusd -g auth
```

After this, *radius* will start to log also via *syslog*. This means that *radius* will log to its standard location and additionally via *syslog*.

It is advisable to change start up scripts (usually located in */etc/init.d/* or */etc/ rc.d/init.d*) so that *radius* will be always started with this parameter.

Orinoco Outdoor Routers 1000/1100

For the sake of simplicity it can be said that Outdoor Router is a sophisticated bridge (or access point) between wireless and wired parts of the network. On its wireless interface it has data rate 11 Mbps (or 5.5 Mbps). Wired interface is 10/100Base-T ethernet.

Orinoco Outdoor Routers (OR) have capability to log via *syslog* [17]. To enable this feature it is necessary to fill *Syslog Host Address* by IP address and *Syslog Host Facility* by number 0 to 7. These numbers corresponds with *syslog* facilities *local0–local7*.

These options can be set up in *OR Manager*—software administration tool shipped with Orinoco Outdoor Routers. The options are to be found either in *Setup→IP Setup* when IP routing *is not* enabled (bridge-only mode), or in *Setup→IP Router Setup* when IP routing *is* enabled. The former situation is shown in Figure 3.2.

3.3 Other Means of Logging

Several other services are performing logging in their own into special files not handled by *syslog*. In this case other methods of remote logging has to be applied. Since the logs are stored in normal text files, what has to be done is to transfer the files to the log server.

[3] This is valid for *Cistron Radius Daemon*, other implementation can use different way to achieve this or can lack this ability at all.

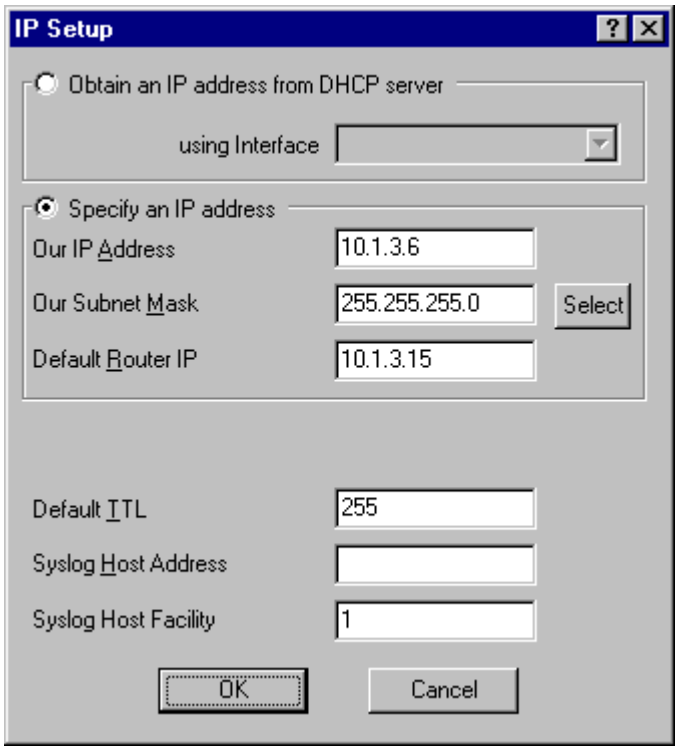

Figure 3.2: Orinoco Outdoor Router IP configuration

The ways how to transfer log to the log server I divided according initiator of the connection between two computers—*push* and *pull* mode and according the continuity of logging—*batch* and *real-time* mode, which in fact simulates *syslog* behavior.

3.3.1 Batch Versus Real-Time Mode

Logs on client[4] are usually rotated (*logrotate* is popular tool for this) for archival purposes—that means the log file is compressed and stored into archive file and the original file is emptied. Period of rotating depends on data volume in the particular log file and typically can be a day, a week or even a month—this is represented by timer T1 in figure 3.3. It would be easy to take compressed rotated log and send it to the log server whenever logs are being rotated. This could be done in *logrotate* configuration file. Unfortunately it is usually necessary to backup the logfile more often to minimize the time attacker would have to clean the traces, before sending logs (now cleared by attacker) to the log server—this period is represented by timer T2. To save bandwidth, *rsync*—which sends only differences

[4]By client in this sense is meant computer which sends its logs to the log server.

between two files—can be used.

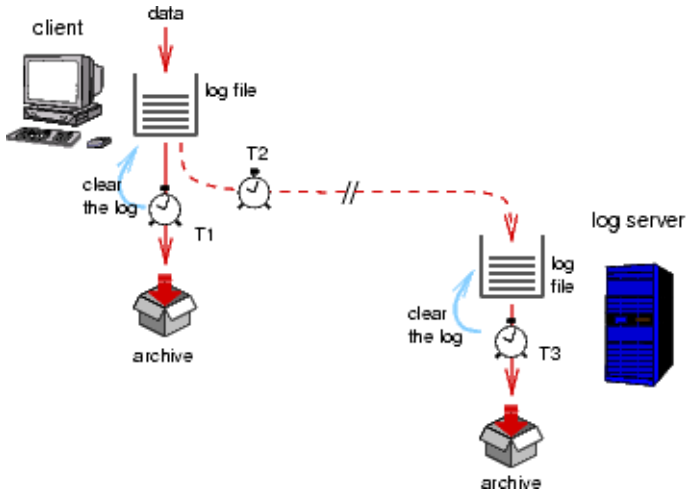

Figure 3.3: Batch mode of transferring log files

Problems arise when logs on client computer rotate. Then old logs are compressed and stored in different file and the original log file is emptied. With next *rsync* synchronization the remote log on the log server would be replaced by new, almost empty, file. Thus log files on the log server must be also rotated in the time between rotating on the client machine and next *rsync* synchronization—timer T3. This require synchronized timers T1, T2 and T3, thus synchronized time on both computers.

Opposing to batch mode is real-time mode. Log files are being watched and all updates are sent to the log server immediately through pipe or *ssh* channel. Since the data which is sent to the log server is not taken from file which is periodically being emptied, the timers T1 and T2 in figure 3.4 are independent and time does not need to be synchronized.

This require to have open connection all the time for each log file, moreover one connection per one log file (the reason for this will be shown later). This solution, particularly *ssh* version, have also other advantage—monitoring the computer. Possibly there is something suspicious on the client computer when the connection interrupts. Moreover *ssh* connection can be made easily non-interactive and still secure, with no need to enter password, using RSA keys.

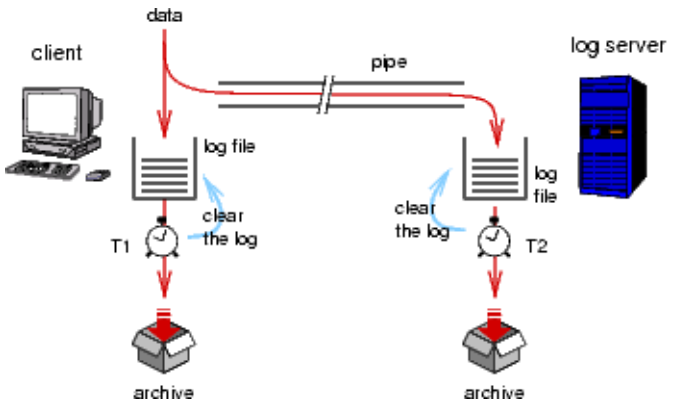

Figure 3.4: Real-time mode of transferring log files

3.3.2 Push Versus Pull Mode

Two methods which I developed are discussed here.

The comparison of these two approaches concerns more security issues then the previous one. Further is assumed *ssh* usage for creating secure connection between two computers. Thus dedicated user is created on both machines and ssh keys without passphrase are used. From this reason the user should have as few rights as possible—only rights that are necessary to write into backup directories and user's root directory should be changed to the top backup directory.

If the *push* mode is employed, the client computers connects to the log server to send the logs there. Since clients must have regular account on the log server (although very restricted), when client machine get compromised attacker has access also to the log server.

If the *pull* mode is employed, the log server connects to the clients to download the logs from them. Thus attacker compromising any of client computer will not have access to the log server, where all logins can be disabled (except e.g. a trusted machine for administration). On the other hand, if an attacker is able to compromise the log server, he/she can get access to all client computers. This is less probable since usually the log server is better protected than other, "normal" servers.

Further *real-time* mode will be assumed, thus having open connection encrypted by *ssh*. Special *backup* user is created with accounts on both machines—log server and client. *RSA authentication* is used as authentication method which allows to use RSA key without pass-

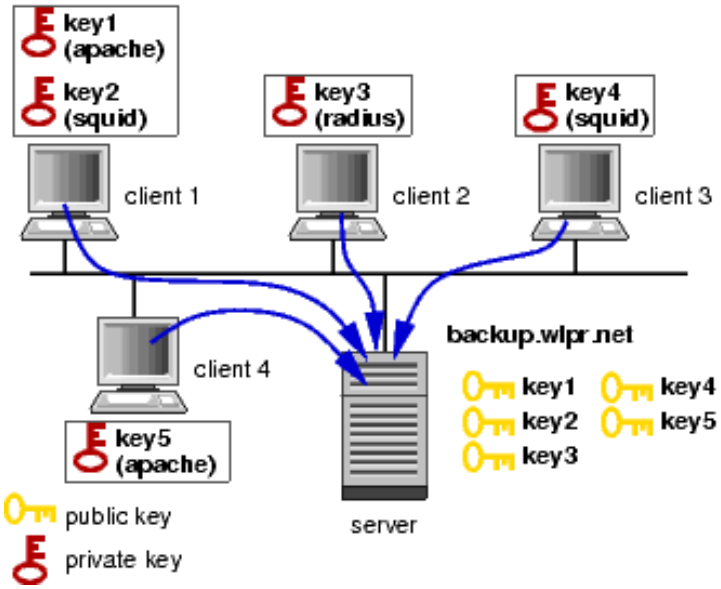

Figure 3.5: *Push* mode; The clients initiate connections and push the data to server. The arrows denotes direction of authentication

phrase, making logging non-interactive. Private and public keys are stored in *identity* files in user's *.ssh* directory. The public key has to be present also in the user's *.ssh* directory in file *authorized_keys* on the computer where this user is logged in. *Ssh* allows to set up single program bound to particular RSA key to be run after login, thus denying access to command-line and greatly decreasing possibility of breaking in. An attacker which captures RSA keys, is not able to gain access to command line.

Since usually multiple log files should be monitored and transferred, multiple connection should be established and therefore multiple keys should be generated. At this point another aspect comes into the play *push* versus *pull*—easy manageability of the keys. These two possible scenarios are depicted in Figures 3.5 and 3.6 respectively.

Realization of *Push* Mode

The *client1* computer (see Figures 3.5 and 3.7) running *apache* server will establish *ssh* connection with the log server using key *id_key_1*:

```
$ tail --follow=name /var/log/file.log | ssh -i ~/.ssh/id_key_1 \
       backup.wlan.net
```

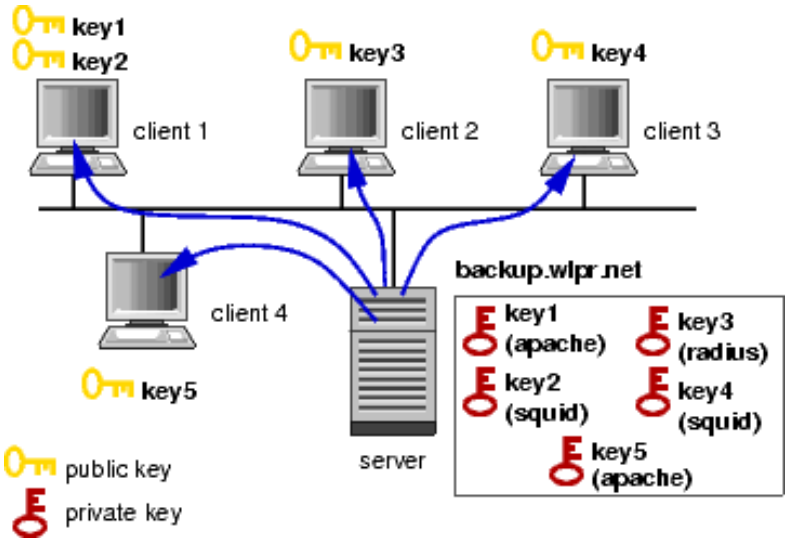

Figure 3.6: *Pull* mode; The server initiates connections and pulls the data from clients. The arrows denotes direction of authentication

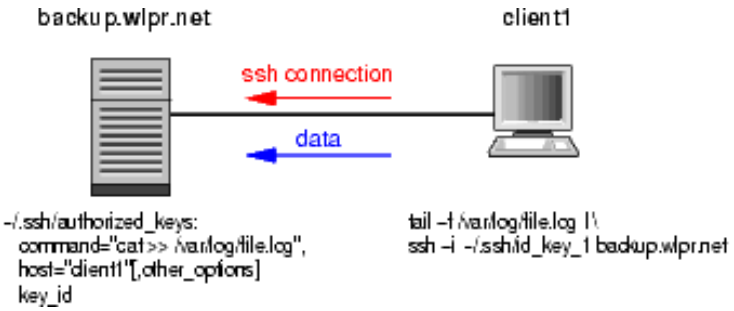

Figure 3.7: *Push* mode in detail

This will cause watching *file.log* and whenever new line is added, this is sent through *ssh* to the remote log server.

The relevant section of *authorized_keys* file on the *log server* will look as follows (everything must be on one line).

```
command="cat >> /var/log/file.log",host="client1",
no-X11-forwarding,no-pty,no-port-forwarding ssh-dss
3NzaC1k...[public key id]...OjCUU= [optional_comment]
```

This will forbid X forwarding, terminal emulation and TCP/IP forwarding. The effect of this command is simple adding of incoming data to *file.log*.

Realization of *Pull* Mode

The *pull* scenario (see figures 3.6 and 3.8) is opposite to the previous one.

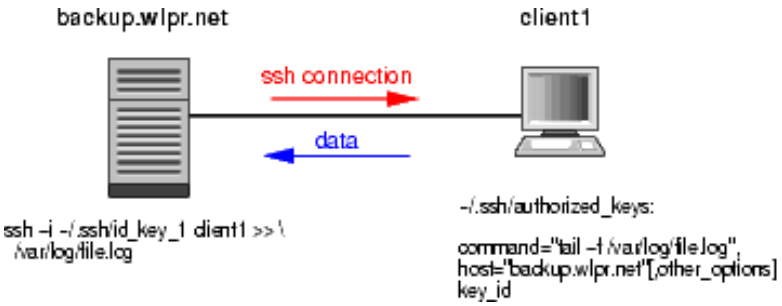

Figure 3.8: *Pull* mode in detail

The log server initiates *ssh* connection with following command:

```
$ ssh -i ~/.ssh/id_key_1 client1 >> /var/log/file.log
```

Sometimes characters *CR* and *LF* are not interpreted correctly. In such a case is necessary filter the data through a filter (e.g. *unix2dos*) to interpret these characters correctly. The above command will look as follows

```
$ ssh -i ~/.ssh/id_key_1 client1 | unix2dos >> /var/log/file.log
```

The relevant section of *authorized_keys* file on the *client computer* will look as follows (everything must be on one line).

```
command="tail --follow=name /var/log/file.log",host="client1",
no-X11-forwarding,no-pty,no-port-forwarding ssh-dss
3NzaC1k......[key id]......OjCUU= [optional_comment]
```

This will forbid X forwarding, terminal emulation and TCP/IP forwarding. The effect of this command is simple adding of incoming data to *file.log*.

3.4 Discussion

From above mentioned methods, logging as much as possible services via *syslog* to the central log server is suggested. This is standard means of logging in UNIX environment and it is widely supported. Services which does not use *syslog* as standard way of logging but allows to use it, should be set up so.

Other services should be transfered to the log server according the *pull* scenario. The reason why I chose this instead of *push* scenario is the centralized management of the *ssh* connections (all connections are initiated from one place) and keys—all secret keys, which is sensitive data, are stored on one place. One server is easier to secure than all of them.

On the other hand attacker which will gain access to the client computer will get not only public key—which is in fact useless—but will be able to modify the command in */.ssh/authorized_keys* which will be run when the special user will log in (using uncompromised secret key) next time. By this, even if attacker is not able to get secret key, which is safely stored on the server, will be possible to either stop the logging process or harm in other way. The question is if this matters once attacker have gained control over the computer. But even in this case (s)he will not be able to reach the server and his/her initial steps (of breaking into the client) will be logged safely on the server.

Opposite view can look at the attacker gaining access directly to the log server. Attacker will have access to all the logs stored on the server, but again will not be able to reach other clients, except the one command specified in */.ssh/authorized_keys*—if (s)he will not get direct access also to these stations. But in that case attacker would have under control whole network and nothing can help.

According to my opinion *pull* scenario provides higher security then *push*.

Possible future work on this topic include testing different implementation of *syslog* namely *syslog-ng* which can use TCP instead of UDP connection, can write messages into database and filter them by regular expressions. Another alternative is *modular syslog* from Core Security Technologies. This implementation can provide writing messages into database, filter them by regular expressions. *Modular syslog* can check integrity and/or encrypt the logs. This is extremely useful if we want to avoid tampering of the logs by attacker while they are sent to the log server.

Chapter 4

Revealing Rogue DHCP Servers

As mention in Section 2.1, according to *usage rules* it is not allowed to offer DHCP service by users. This can cause chaos in the network which can result in unavailability of network services for affected users. Presence of non-official DHCP server can be even preparation for attacks in the network. Since such a server is unwanted, it can be assumed as rogue.

In either case is necessary to find any such DHCP server and take additional measures to stop the service. Section 2.4 deals with basics of DHCP protocol which is necessary to fully understand methods described in this chapter. It is recommended to read that section first.

I suggested several methods of revealing rogue DHCP servers which are discussed in further sections. Basically there are two basic "external" ways to reveal rogue DHCP server: *passive* and *active*. The last possibility is to use improved DHCP protocol according to [18], which introduces new option for authentication of DHCP messages. Then DHCP server is authenticated to the client and thus any messages from rogue DHCP server would be ignored (provided attacker didn't break the authentication part of the protocol).

4.1 Active Method

This method employs idea of actively searching for DHCP servers on the network and then filtering out the official ones, leaving on the list only "illegal" or "rogue" servers.

It is necessary to have one computer which will monitor the network. This monitoring station can send DHCPDISCOVER packet, which is used for localizing DHCP server and asking for IP address. DHCP server should answer with DHCPOFFER packet, offering IP address. This address is then marked as "offered" and will not be used for the lease time by DHCP server. It will be released in short time if it will not be requested.

The major drawback of this method is in replying of the DHCP server. It can be configured to answer only to specific range of hosts or only to hosts on a list of allowed MAC addresses. This situation is shown in Figure 4.1. In such a case, our DHCPDISCOVER would be ignored by the rogue server.

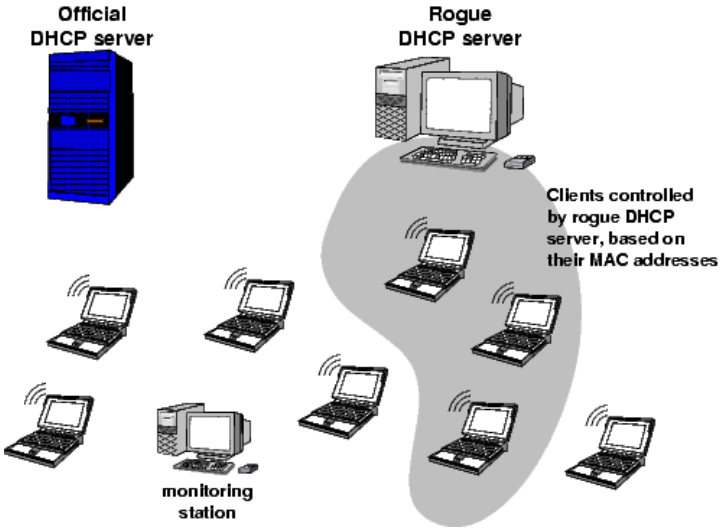

Figure 4.1: Rogue DHCP server can serve only to selected clients, thus to ignore monitoring station.

An advantage is that the DHCP packets can be relayed through possible subnetworks. DHCP relay is usually router, which captures broadcasted DHCP packets and rebroadcasts them to other subnetworks. DHCP protocol allows relays to rebroadcast packets or retransmit them directly to the requested destination. This feature DHCP inherited from BOOTP protocol [19]. If the packets are rebroadcasted depends on the configuration of the DHCP relay. Why it is important to rebroadcast the packets instead of only retransmit? One possible scenario where this make difference is depicted in Figure 4.2. Our monitoring station is in different subnetwork then rogue DHCP server, which is also in different subnetwork then official DHCP server. If the relays would be configured only for retransmitting, they would send the broadcast from our monitoring station directly to official DHCP server, since there is no relay between subnetworks 2 and 3. This means, that the broadcast packet would be "broadcasted" only in the subnetwork where the station is located—in this case

subnetwork 3. In subnetwork 1 it would be sent directly to the DHCP server by the relay "B" and not also to the rogue DHCP server in subnetwork 2. Unfortunately most of the DHCP relays are configured not to rebroadcast the packets, because it would be relatively easy to cause denial-of-service attack.

Such subnetworks and relays configurations prevents hosts in subnetwork 3 be fooled by rogue DHCP server located in subnetwork 2. Only host located in subnetworks 1 and 2 would be endangered.

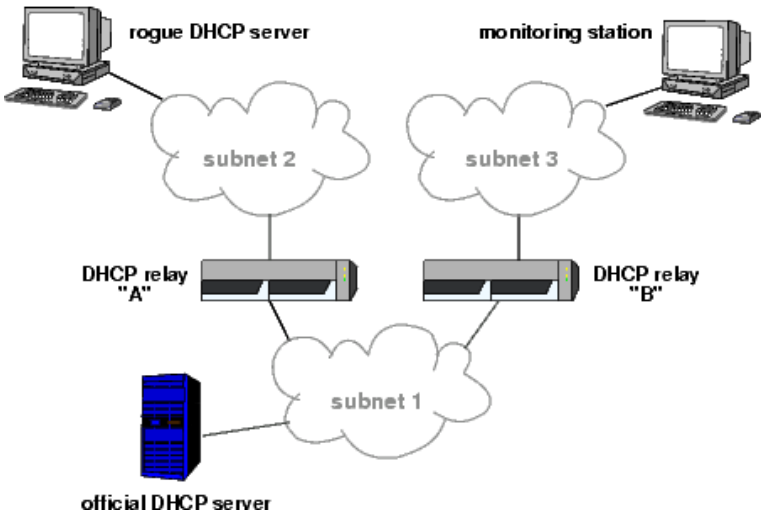

Figure 4.2: Subnetworks connected by DHCP relays

4.2 Passive Methods

Besides active searching for DHCP server by sending DHCPDISCOVER packets, it is possible to use passive methods to reveal DHCP servers. One possibility how to do it, is to sniff on the network. To do this it is enough to switch the network card to promiscuous mode, thus process all the packets going on the subnetwork (or segment in case of switching ethernet). By filtering out UDP packets going from port 67, which is used by DHCP on the server side, it is possible to get only packets from DHCP servers. Then it is easy to exclude packets coming from official servers. Any sniffing tool can be used for this purpose, e.g. *tcpdump* (www.tcpdump.org), *netl* (www.netl.org) or *ethereal* (www.ethereal.com).

Major drawback is, that this method does not work if the network is divided into several subnetworks or even if a switched technology is used. Unfortunately this is almost always

case in large networks. It would be necessary to have one sniffing computer in every subnetwork or segment.

Other option is to look for relevant information to the log of official DHCP server.

To find information about rogue DHCP servers in official DHCP server is necessary to have a closer look how communication between client and two DHCP servers looks like.

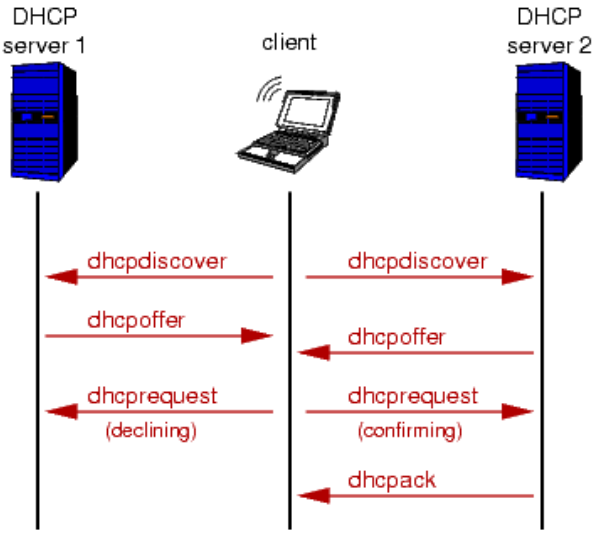

Figure 4.3: DHCP communication with two servers

As stated in Section 2.4 when a client get more then one DHCPOFFERs to its DHCPDISCOVER message, it must broadcast also following DHCPREQUEST in the same way as the first DHCPDISCOVER. This ensures that all DHCP servers which possibly answered the initial message will get also this DHCPREQUEST. In this message must be present "server identifier" option to indicate which server it has selected [10]. Thus even in the log file of server which has not been selected, a message appears that this server received DHCPREQUEST saying that its offer has been declined. This situation is described in Figure 4.3.

From above it can be found that looking for "right" DHCPREQUEST logged in the official DHCP sever log file reveal rogue DHCP server. Initially, looking for patterns of messages was done, but parsing the log and looking for only one message is faster and gives the same results.

It was difficult to find the right patterns which would not cause false alarms. There were several different patterns matching different scenarios: client is asking new lease after e.g.

reboot; client is asking to prolong the valid lease from rogue server; client is asking the same lease which was not longer valid; etc. Some of these patterns collide with situation when unregistered computer (with short lease) get registered. DHCP server in this situation refuse to prolong old lease and offer new one instead. This situation is similar to the one when client gets new lease from rogue server.

Finally was found that such complicated patterns are not necessary and matching one single line of DHCPREQUEST is sufficient. Such a line looks like

```
Mar 17 12:59:00 wlansrv1 dhcpd: DHCPREQUEST for 10.0.30.244
    (10.1.3.1) from 08:00:20:04:f5:df via eth2
```

This message means that client with MAC `08:00:20:04:f5:df` requested IP 10.0.30.244 from DHCP server at 10.1.3.1 (which was simulated rogue server). This request came to the server via eth2 interface.

This method has also one drawback. There is a situation when the rogue server remain hidden. This can happen if the official server will not be accessible for some period of time (e.g. part of the network lost the connection and became isolated or the server is down for a while) and the client will get lease from rogue server. Then this will not be logged into the official server logfile, but also consecutively client will try to keep the same lease and perhaps it will ask the rogue server directly to prolong its lease. This means till the time when the client will start the initialization process (to ask for a lease from beginning starting with DHCPDISCOVER), it can keep lease from rogue server and there will be no note in official server log.

4.3 Discussion

During the research several methods were tested. Each of them has its own advantages and disadvantages.

The active method will not reveal clever attacker which will first listen on the network or find some information about topology by other way. Then (s)he can choose to serve as DHCP server to only selected computer(s) proven to be only “harmless victim”.

Therefore I left the active solution in favor of passive one. Since the sniffing on the each subnetwork needs to install lot of monitoring stations I prefer the "parsing-the-log" version. This is easily implementable and moreover very elegant solution.

The main problem arises when it is necessary to reach the attacker and stop the rogue DHCP server. We must take into account that the rogue server can be e.g. only unintentional or forgotten DHCP server after fresh installation of an operating system or poor configuration of home network. It would be inappropriate to directly cut off the network connection for such a user.

Unfortunately there are not many possibilities to choose from. One is to cut off the connection for the user in access point (filtration by MAC address)—as said before, this is inappropriate solution. Other solution is similar to the one used in next chapter—to redirect the user using the IP of rogue DHCP server to special page, when (s)he will browse the web. To reach the outside Internet the user has to use transparent proxy server and at this point it is possible to force him/her a page explaining the situation (more detailed description will be presented in next chapter or interested reader can consult [20]). This solution requires the user accesses the outside Internet by web browser.

Real attacker will certainly not use the rogue DHCP server for browsing the web. And in this case there is no other means to stop him/her but to cut the connection off. In fact it is possible to find out which access point the attacker uses and limit the possible range of suspect users, but to completely track down the attacker is very difficult.

The best solution which is not yet available is to use improved DHCP according to *Authentication for DHCP Messages* [18]. Then all the major problems disappear and we all will live better lives. This requires improved server as well as clients. There is no implementation known to the author at the time of writing.

Chapter 5

Tracking of Invalid IP Addresses

In the wireless test network DHCP server is used to issue all IP addresses, users are not allowed to use their own "fixed" IP addresses. If user wants to have IP which will not change over a time, it is possible to set it up in DHCP server by network administrator.

DHCP is also used to deliver to clients' workstations important information about network as default routers, DNS servers, and netmask. For different categories of users, there are different configurations given by DHCP server. Users entering their own fixed IP address can have malicious intentions and intentionally spoof IP address—act as someone else. Thus it is important that users are using IP addresses issued by official DHCP server. Users using non-DHCP issued IP address should be found and instructed to use DHCP server in their network settings, as stated in Usage Rules.

Original idea or starting point for following methods was taken from *NetReg* project [5]. The solution they used in the project is to *ping* whole address space to fill up *arp*[1] table by MAC/IP pairs. Then to download this *arp* table from a router and compare this to DHCP server log.

Although the idea is nice, sometimes it is very difficult to realize. If the network has large address space it is impossible to send *ping* packets to every single IP address. This would either overload the network or it would take too long time. For illustration scanning of 10.0.0.0/16 network by *nmap* program took 961 seconds (16 minutes). There were 65536

[1] Address Resolution Protocol (ARP) is used to find MAC address to known IP address. Arp table then stores these MAC/IP combinations. This table is usually located in routers.

addresses scanned and only 21 hosts were up, thus the load of the network was minimal. *Nmap* was set up for sending particular *pings* as fast as possible without overloading the network or missing hosts. Since as default *nmap* uses ping and TCP ACK packets in parallel to see which hosts are up, additional condition was necessary to switch off TCP scanning.

This drawback is possible to minimize using straight *arp* requests instead of *ping*. *Arp* requests only ask MAC address corresponding to known IP address, while *ping* measures response time of the host (and *arp* request is part of it). Thus using *arp* request would speed the network scanning up.

Usually *arp* tables are not large enough to accommodate all the addresses. Thus splitting the whole process to smaller parts would be necessary. Besides this method is inefficient—trying to reach also hosts which are not running (or even do not exist). Therefore other methods were researched.

5.1 Using Access Point Bridge Table

Users are accessing the wireless network via *Access Points (AP)*. This is the best place to look for MAC addresses appearing on the network. Since AP is acting as a *bridge*, there is only *bridge table* present instead of whole *arp table*. The latter one has also IP addresses corresponding to MAC, while the former one stores only MAC addresses going through that particular access point. Therefore MAC addresses have to be used for user identification.

Since all users should use DHCP, the same MAC addresses as in AP should be present in DHCP leases file. Thus by comparing two lists of MAC addresses—one from AP and second one from DHCP server—it is possible to find users who have never used DHCP[2]. This situation is clearly seen in Figure 5.1a. Users which are not using DHCP will be present in the list from AP but not in the list from DHCP.

This comparison is only one part of the whole process. Second part should reveal also users which are using (either by accident or from previous sessions) IP address which was once issued by DHCP server but is not valid anymore.

DHCP server stores both MAC and also IP addresses, but these leases can be expired. More-

[2]DHCP server often stores also history. Therefore if user once used for a while DHCP issued IP address, (s)he will not be revealed by this way.

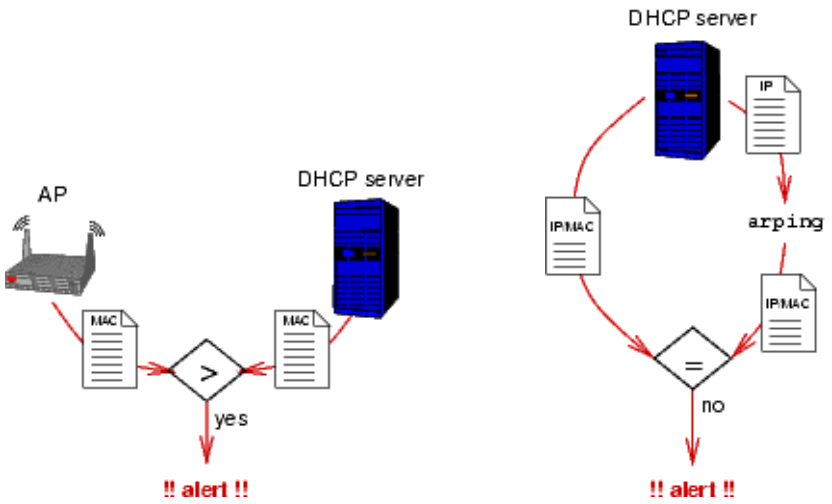

Figure 5.1: Revealing illegal IP addresses using AP bridge table: first find if given MAC has a lease (a), then find if the lease is valid (b)

over the MAC address appearing in AP can be present in DHCP leases file, but associated with different IP address. It is seen that also comparison of actual state with DHCP server is necessary. This comparison is schematically presented in Figure 5.1b. All the IP addresses from DHCP server are passed to *arping* program which sends, as name suggests *arp* queries for MAC address. Responses are then compared to DHCP leases file (combination of IP/MAC pairs and validity). Note that only existing IPs are checked thus reducing network load and saving time.

Table 5.1: Possible results of comparison of actual state with DHCP leases file

DHCP lease	*response from arping*	*status*
expired	no response	all right
expired	any response	alert
valid	the same MAC in response	all right
valid	different MAC in response	alert
valid	no response	no conclusion

Possible combination of *arping* responses and DHCP leases are listed in Table 5.1. First possible response represents situation when the lease is expired, therefore there should be no response—there should not be such a computer on the network. Second response represents exactly opposite situation—a computer is using expired lease and alert will be raised.

Third and fourth responses match situations of valid lease with correct response—the same MAC address was returned by *arping* query—and incorrect response. The last response represents situation when the computer with valid lease is switched off or did not responded from other reasons. In that case nothing can be concluded.

If the response is incorrect—MAC address is different—it means that for one IP address there are (at least) two MAC addresses. This can happen if an attacker will find out that the lease with this IP address is valid and enters this IP as fixed into his/her network configuration. Using this IP address the attacker can cause DoS for original user or even use his/her connection to Internet through proxy.

5.2 Using DHCP Only Database

The idea of this method came out from different thought. Initial question was: "*Where* is possible to stop the user and deliver him/her information about using DHCP server?" Certainly this is not inside the network which is free to use by everyone. Best place is the proxy server on boundary of the wireless network and the outside Internet (which access is paid). This proxy server is transparent and users usually are not aware of it. Thus the proxy server should be the place to track the users.

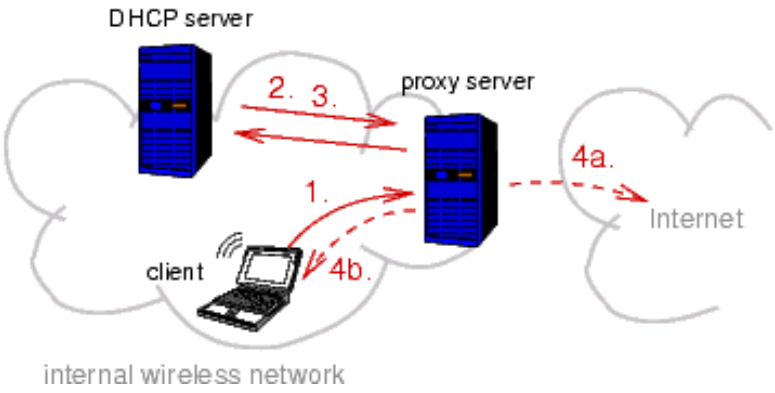

Figure 5.2: Stopping user at the proxy. For description see text

The idea is depicted in Figure 5.2. If a client wants to reach outside Internet by HTTP or FTP protocols, it has to go through the proxy server (in Figure 5.2 denoted as 1). At this point the source address of the request—which is the client's address—is checked if there is valid lease in DHCP leases file (2 and 3). Besides this the client's address is checked also for authentication, but this is irrelevant here. Based on the validity of the lease the client's

request is passed to the Internet (4a) or special page explaining necessity of using DHCP is returned to the client (4b). This effectively stops users using non-DHCP IP addresses from using outside network.

Since the process of acquiring information from DHCP server has to be as fast as possible to minimize the delay when requesting web pages, it is possible to keep copies of DHCP leases in database. This situation is shown in Figure 5.3. Currently author knows no implementation of DHCP server using directly database for storing the leases. Therefore in this case separate database should be used. The database should be updated immediately when the leases are issued by DHCP server. Thus there is a need for a script which will parse newly arriving lines to DHCP leases file and continuously update the database.

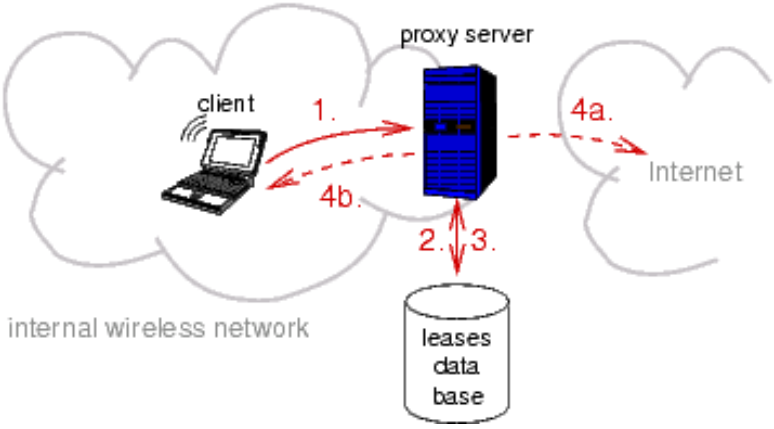

Figure 5.3: Stopping user at the proxy with database

5.3 Discussion

Although the first method—using AP bridge table—seems to lack a major, principal drawback there is one problem which disqualifies this for possible employing. At the end the MAC address will identify the user who is not using DHCP issued IP address. The problem arises when such a user should be contacted or blocked from using the network. Knowing only MAC address it is possible to set up access points to block it. This way of stopping user does not give any way to say the user what is wrong. Inexperienced user will wonder that "the network is not working". This would be possible to solve by ability of access point directly redirect such a user to special web pages explaining necessity of using DHCP server. Unfortunately at present, there are no access points with such a capability known to the author.

To find IP address matching MAC address is rather difficult problem. One of protocols—RARP (Reverse Address Resolution Protocol)—is old and not used anymore (it is not necessary nowadays since it was replaced by BOOTP and later by DHCP protocols). One possibility would be to scan whole network by *arping* to "build" own huge arp table. Then there arise problem with active presence of the computer etc.

The problem of contacting the user is addressed in second method. This method has also one drawback—user with non-DHCP IP address can use the inside network without limits. Only when (s)he will try to reach outside network (Internet) by HTTP (or FTP in browser) it is possible to stop him/her. Inside the network there is no possibility how to effectively stop the user, since (s)he will not pass any of network devices except access point. Only at the AP such user can be stopped, but as was mentioned above this solution is not preferred.

Comparing these two methods discussed in this chapter, the final results are similar. Even if it would be possible to find IP address matching MAC address, there is no other way how to stop the user. Personally I like the first solution more. It is more accurate and it will reveal more users using non-DHCP IP address.

On the other hand there is a question what user gains by using fixed IP address (not issued by DHCP server). To access even inside network (s)he needs to know default router, netmask and DNS servers. If (s)he wants to access the Internet (s)he has to know also address of proxy server. Netmask is easy to guess and the other parameters can be found from e.g. friends or from previous use of DHCP.

Then such a user can enter fixed IP address of someone else who exists on the network and possibly use his/her connection to connect to Internet. In this case the attacker can be revealed by all of the methods discussed earlier.

Chapter 6

Solution

This chapter discuss solutions to problems presented in previous chapters (3, 4 and 5). Reasons why the solution was chosen and implementation of the solution in the wireless test network is described.

6.1 Logging

Process of logging is important for other parts of the thesis, since it collect information which is used by methods discussed in later chapters. Fast and reliable logging is therefore key to success of revealing DHCP servers and invalid IP addresses. Solution employed in the test network is presented in this section.

6.1.1 Remote Logging

In correspondence with Chapter 3 all services able to log via syslog are doing so. This means they are set up to send logs also to the remote log server.

Services which are initially logging only to a file but are able to log via syslog are set up according Chapter 3. This concerns following services and devices: *apache*, *squid*, *radius*, *Orinoco outdoor routers*.

Set up of these services requires to select the *syslog* facility which it will use. To have all facilities consistent over the network, recommended ones are listed in Table 6.1.

Table 6.1: Recommended *syslog* facilities

service	*facility*
Access points (Orinoco)	local1
Apache server	local4
Radius server	local5
DHCP server	local6

6.1.2 Transferring Non-Syslog Log Files

For transferring other logfiles to the log server (backup server) the *pull* scenario was chosen. The main reason is centralized management of private keys which are in this case in one place—on the server (see figure 3.6). From security point of view the log server remains unaccessible for the special user, thus creating less possibilities to break in.

Dedicated user is used to connect from log server to the client via *ssh*, authorized by RSA key. This user is unable to run any commands except the one which is mentioned in */.ssh/authorized_keys* file [21]. Since this file is located on the client computer, attacker who would be able to log in as this dedicated user, can not run any other commands.

The command is *tail*, which is watching the specified log file and writing it to standard output, thus sending all the newly arriving lines directly through *"ssh pipe"* to the log server where it is stored and processed. Exact command syntax will be shown later.

Note that for one server and one log file being watched, one connection is necessary, thus one RSA key-pair. On each client public keys are stored, while private keys are stored *only* on the log server.

Step-by-step guide how to add one logfile to transfer follows. Further will be assumed that the file */var/log/httpd/access_log* from computer *turgon* is transferred to */var/log/transfer/turgon_apache.log* on the log server.

1. **Create *transfer* user.** To ensure maximum security special user is dedicated to transfer log files. Than it is easy to limit user's rights to only necessary ones. Create this user

on the log server and also on all "client" computers.

```
# adduser transfer
```

Password should be consistent (although this can be considered as security weakness) for easier maintenance—it is almost impossible to remember different passwords of this user on different computers.

2. **Generate ssh keys on the log server.** For every pair of *host/logfile* one pair of keys has to be generated. Utility *ssh-keygen* is used for this purpose.

   The following should be done as user *transfer*, in its home directory. First it is necessary to create directory where this keys will be stored (default is */.ssh*). Than generate the keys (see *man ssh-keygen* for more info).

   ```
   $ ssh-keygen -t rsa
   ```

   Switch -t denotes type of the key (in this case RSA, which is default for *ssh* version 2 protocol. Save key in file */home/transfer/.ssh/id_rsa_**turgon_apache*** and as passphrase enter nothing—this will allow non-interactive connection.

   Ssh will create two files—*id_rsa_turgon_apache* and *id_rsa_turgon_apache.pub*. The first one is private key while the second one is public.

3. **Copy public key to the client computer.** Now it is necessary to copy *public* key to the client computer into the file */.ssh/authorized_keys*. If this file exists simply add the new key to new line. Prepend this line with following code (without spaces after commas):

   ```
   command="tail --follow /var/log/httpd/access_log",
   no-X11-forwarding,no-pty,no-port-forwarding
   ```

   After this limiting commands the key should follow (on the same line). Above mentioned commands will ensure that after logging in, only command *tail –follow /var/log/httpd/access_log* will be executed and no terminal will be allocated. The standard output is redirected to the *ssh* connection, thus to the log server.

4. **Create startup scripts on the log server.** Create new script for starting this transfer session. Use "master" file in Appendix A. Place this script in */usr/local/sbin* directory.

   Then add this script to variable *LIST_TRANSFERS* in startup script in */etc/rc.d/init.d/transferd*—see Appendix B.

5. **Create log directory.** Create directory */var/log/transfer* and make sure that user *transfer* can write to it, or change *LOG_FILE* variable in application script (Appendix A) to requested value.

6. **Start the transfer daemon.** Now everything should be ready to start.

```
# /etc/init.d/transferd start
```

If everything is right, all the log files are being transferred from clients to the log server.

To stop the *transferd* daemon write:

```
# /etc/init.d/transferd start
```

This should stop all the logging. Unfortunately it kills only the shell, but *ssh* connections are still running. This is necessary to kill manually. This is known bug which will be solved later.

If it is necessary to stop only one (or several) services, kill process with PID noted in file */var/run/**turgon_apache**.pid*.

6.2 Revealing Rogue DHCP Servers

First I decided to use the active method. For sending DHCPDISCOVER packets short program was developed. This program is based on *dhcping* [22] program. My modified version can send only DHCPDISCOVER packets and listens to answers. Output of this program was improved to match further script processing.

The responses coming back to this program have "wrong" IP address. DHCP server thinks that the client does not have any IP yet, and sends the DHCPOFFER to right MAC address but set up IP address to the one being offered (not to the one the computer already has). This behavior completely corresponds with DHCP standard.

The DHCPOFFER packet has correct MAC address but "incorrect" IP address—from monitoring station point of view. This monitoring station is the destination according MAC address in the packet. But this host already has IP address, which is different then the new one offered by DHCP server in this packet. This packet will be processed by monitoring station on lower level of network subsystem (MAC address is correct), but will be dropped by IP layer, since the destination IP address does not match its own IP. Therefore *IP tables*,

which is network filter, should be set up that will pass UDP packets from port 67 and 68 (DHCP packets) to upper layers of operating system.

Following command can be used to set up *IP tables*[1]:

```
iptables -t nat -A PREROUTING -j REDIRECT -p udp --source-port 67:68
```

As a result of this was found easier, passive solution.

From the two passive methods mentioned in Chapter 4, I chose the latter one for implementation—parsing the DHCP log file. This method is fairly easy to implement especially with ready-to-use tools for parsing log files.

For implementation I chose *logsurfer* [23] for its support for looking for patterns of messages through several lines instead of testing only one line at a time as e.g. *swatch* program is doing. The information which the parser should look for is given as so called *regular expression*. Description of regular expressions is beyond the scope of this work. Interested reader should consult [24] or [25].

The regular expression used for searching of broadcasted DHCPREQUEST can be found in Appendix C.1. This expression will match exactly following part of the string (with optional addresses of course):

```
dhcpd: DHCPREQUEST for 11.0.30.30 (10.1.3.18) from 08:00:20:04:f5:df
```

Whole string can be found on page 36. This matched substring appears only in the situation when client is broadcasting its DHCPREQUEST to more than one DHCP server, thus declining one of the offered DHCP leases.

After that it is necessary to decide if the DHCP server which lease was declined is the official one or possibly rogue one. On the result this does not have big difference. In both cases there is one DHCP server more, but it is useful to know from whom client accepted the lease.

[1] This is valid for Linux kernels 2.4 series. In older series 2.2 and 2.0 similar program called *IP chains* was used. Consult your documentation how to set up *IP chains*.

The *logsurfer* allows to specify also regular expression which the line should not match, which is very useful for excluding official DHCP server from being logged. Configuration file for *logsurfer* can be found in Appendix C.2. Note that this exactly matches only when using DHCP implementation from Internet Software Consortium (www.isc.org), which is the most used implementation under Linux operating system.

To close such unwanted DHCP service is difficult task and needs further researching and testing. Implementation depends whether we know the user running the rogue server (we know its IP and MAC address) and whether we are able to contact the user. At present situation it seems that the only possibility is to stop this MAC address in access points.

6.3 Tracking of Invalid IP Addresses

I decided to implement the last method—using DHCP only database—because in the first one there is not solved effectively how to stop the user.

There is a *MySQL* database in the test network project used for radius authentication and for storing positions of users (in which AP they were located last time). One more table was added for storing DHCP leases, see table 6.2.

Table 6.2: MySQL database table for storing DHCP leases

name	MAC	IP	start	expire
type	CHAR(17)	CHAR(15)	TIMESTAMP	TIMESTAMP

This database is held up-to-date by special script *dhcptaild*, which was written as part of this work. The script can be found in Appendix D.

Following is valid only for HTTP requests or FTP requests done from web browsers, because web browsers are using transparent proxy server also for FTP requests. "Classical" FTP clients usually do not have proxy capabilities.

If HTTP request (or FTP) came from inside network to proxy, the source IP is checked in the database if it is authenticated by radius [20]. I added testing of DHCP lease. Thus if the source IP address matches a valid lease in the database, the request is allowed to pass. Otherwise user is redirected to special page explaining necessity of using DHCP issued IP addresses.

For redirecting, the capability of *squid* proxy was used. For more information about this proxy and redirecting consult [20] and [14].

The SQL query to the database is

```
SELECT IP, MAC FROM leases WHERE IP=ip AND
  (NOW()<=expiry OR expiry=0)
```

where *ip* is IP source address of user who sent the request. This will return all combinations of IP and MAC which has valid lease. Some leases in database has zero (0) expire time—these are fixed IP addresses issued by DHCP server. The server does not issue leases for these cases, the main reason why leases exists is time dependence, while fixed addresses have the same IP over the time, so there is no need for a lease. Therefore fixed IP addresses have to be added to the database manually.

6.4 Analysis of a Use Case: IP Address Hijacking

Testing is very difficult and can give false results—the times are dependent on start time of the test in relation to the start times of DHCP leases etc. From this reason analysis in form of use case is presented here. As use case, hijacking of IP address is chosen, since this encompasses IP address spoofing and the measures presented in this thesis are aimed to stop such abuse.

Description follows Figure 6.1. User A is regular user of the network. Whenever (s)he tries to access the network, the computer has to associate with the access point (AP1). Access point is checking radius server if the MAC address is allowed to join the network (1). Since the network is open to public, only blacklisted MAC address are denied. Then the computer asks for IP address from DHCP server (2). After that user can access the network (3) and also outside Internet (4), provided (s)he has agreement with an Internet Service Provider (ISP) who actually provides the outside connection.

Now another user (B or attacker) comes into the play. As a user he can sniff some time on the network and find that user A has IP address e.g. 10.1.2.3, and also that there is a Internet connection (which can be paid) for this user through the ISP. The attacker can use IP address of user A as his own, fixed IP address. Now both use the same IP, which can

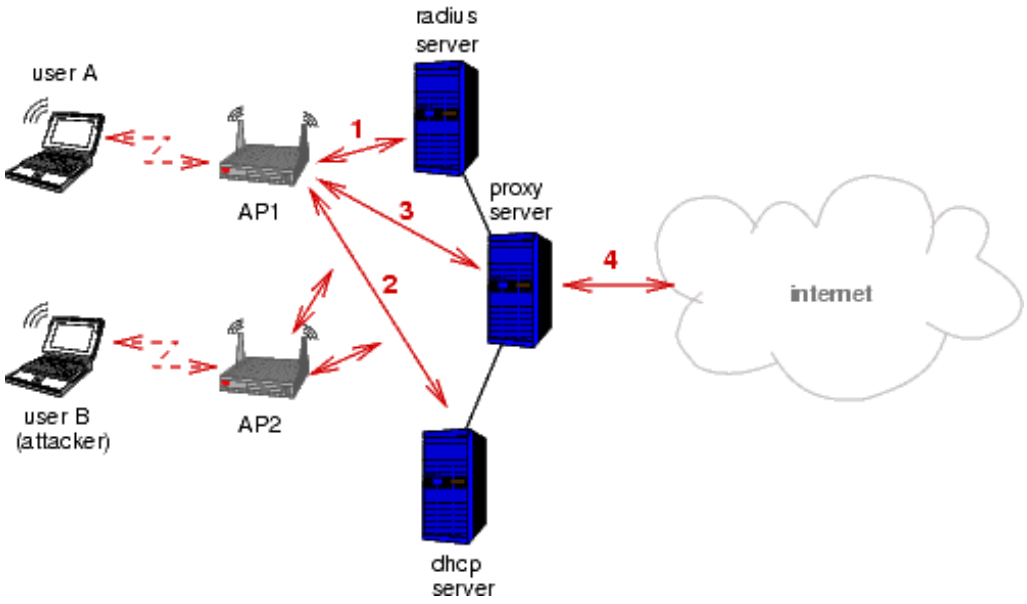

Figure 6.1: Use case: IP address hijacking

cause collisions. Attacker can also wait when user A will stop to use the network and thus to avoid the collisions.

Attacker now can pass the same procedure as earlier described (association with AP, AP checks MAC address in radius server) except obtaining IP address from DHCP server, because attacker has its own IP address already. Since attacker now uses IP address which *is* valid since there *is* valid lease for this IP in DHCP database, (s)he can pass through proxy server to outside Internet. If the ISP authentication is based on IP address (e.g. after secure authentication, *the IP* is allowed to pass), then the attacker can use Internet connection without paying—actually paid by user A.

How the measures presented in this thesis can reveal and stop such attacker? The time in which is possible to find and stop him depends on several factors. There are several time limits and it depends which will expire first. I will take all the possibilities in order.

1. **Blacklist in radius server.** Every time a computer is associating with an AP, the AP is asking radius server if the MAC is allowed to enter the network. Radius server allows to specify blacklist of MAC addresses which are not allowed to enter the network. If the attacker's MAC is known it can be added to the blacklist. Once the computer is associated with the AP, next re-association can take place in several hours, unless the computer is moved to another AP or lost the connection for a while. Even in the

case the computer will not associate longer than certain time, the AP can renew all the information from radius database in regular periods. In our test environment this time period is set up for 12 hours.

2. **Blacklist directly in AP.** It is possible to add the MAC address into a blacklist directly in the AP. In this case the AP, through which the attacker is connected, has to be known (this information is in radius database). This takes effect immediately but it requires to reboot the AP after the blacklist was updated, which is not desirable. Moreover this affects only the one AP.

3. **DHCP lease expiration time.** On the proxy server, the presence of valid lease for all passing IP addresses is checked. If the lease expires during some time, the attacker is unable to access outside network anymore. Expiration time for registered users is set up for 24 hours in our test network.

4. **ISP authentication expiration time.** Authentication on ISP side is expiring every 24 hours. This arrangement is temporary and can be changed in future. After expiration of this time, the attacker is unable to access outside network.

Summed all the possibilities up, the maximum time attacker is able to access the Internet illegally, is the time necessary to reveal him and stop him. This is necessary to do system-wide, otherwise attacker would be able to move to another AP and continue using the network. The maximum time can be up to 12 hours, which is the period of updating AP information from radius server. If the AP would be updated directly, not via radius, the blacklist will take effect immediately after updating the AP. These cases are valid if the attacker's MAC address is known. This can be found by method described in Section 5.1 and shown in Figure 5.1a.

If the MAC address is not known, then the maximum time increases up to 24 hours. Note that these are *maximum* times in case the attacker will spoof its IP address exactly after the renewing information in AP from radius server, issuing new lease or logging in to ISP system respectively. Also the times mentioned here are illustrative only, taken from experimental test network. However the times can be improved e.g by shortening update time interval of AP from radius server.

Chapter 7

Conclusion

This work describes often neglected problem of forcing users to follow network Usage Rules. To reveal the violations against the Usage Rules, information stored in central log server is used. Remote logging methods are used to collect this information.

The remote logging is solved by using *syslog* daemon and by creating *ssh pipes* for transferring bigger data rates. Violation against Usage Rules which are discussed in this thesis—using own DHCP servers and using own IP address (not issued by official server)—are divided into two parts: revealing and stopping of such users. Although revealing is quite successfully solved mainly by passive approach—parsing log files generated by DHCP and other services—stopping of the users is less successful. Users can be stopped only when they will access the Internet (i.e. outside the metropolitan network), thus going through a transparent proxy where they can be caught.

Although this thesis is primarily focused on wireless environment, its results can be used in wired networks also. All the implementation is done in higher layers of OSI model, which are the same in both environments. Probably small tuning will be necessary as for any particular case, but the general ideas remain the same.

References

[1] FLICKENGER R.: *Building Wireless Community Networks*, Sebastopol, CA: O'Reily & Associates Inc, 2002.

[2] FREENETWORKS COMMUNITY: *Wireless Networking Projects around the world*, available: http://www.freenetworks.org/moin/index.cgi/WirelessNetworkingProjects (May 8, 2002).

[3] TOASTER.NET COMMUNITY: *802.11b Community Network List*, available: http://www.toaster.net/wireless/community.html (May 8, 2002).

[4] GEIER J.: *Wireless LANs–Implementing Interoperable Networks*, Macmillan Technical Publishing, 1999.

[5] VALIAN P., WATSON K. T.: *NetReg: An Automated DHCP Registration System*, LISA XIII conference proceedings, pages 137-145, November 7-12, 1999.

[6] 802.11 IEEE GROUP: *IEEE Standard for Wireless LAN Medium Access (MAC) and Physical Layer (PHY) Specifications*, IEEE, 1997.

[7] BORISOV N., GOLDBERG I., WAGNER D.: *Intercepting Mobile Communications: The insecurity of 802.11*, available: http://www.isaac.cs.berkeley.edu/isaac/wep-draft.pdf (May 8, 2002).

[8] ARBAUGH W.A., SHANKAR N., JUSTIN WAN Y.C.: *Your 802.11 Wireless Network has No Clothes*, http://www.cs.umd.edu/~waa/wireless.pdf (May 8, 2002).

[9] FLUHRER S., MANTIN I., SHAMIR A.: *Weaknesses in the Key Scheduling Algorithm of RC4*, August 2001, available: http://www.cs.umd.edu/~waa/class-pubs/rc4_ksaproc.ps (May 8, 2002).

[10] DROMS, R.: *Dynamic Host Configuration Protocol*, RFC 2131, IETF, 1997.

[11] GARFINKEL, S., SPAFFORD, G.: *Practical UNIX & Internet Security*, Sebastopol, CA: O'Reilly & Associates Inc, 1996, Czech translation Praha: Computer Press, 1998.

[12] APACHE SOFTWARE FOUNDATION: *Apache HTTP Server Project*, available: http://httpd.apache.org (May 8, 2002).

[13] APACHE SOFTWARE FOUNDATION: *Apache HTTP Server Documentation*, available: http://httpd.apache.org/docs/logs.html (May 8, 2002).

[14] WESSELS D. ET AL: *Squid Web Proxy Cache*, available: http://www.squid-cache.org (May 8, 2002).

[15] WESSELS D. ET AL: *Squid documentation*, available http://www.squid-cache.org/Doc/FAQ/FAQ-3.html#ss3.8 (May 8, 2002).

[16] SMOORENBURG VAN M.: *Cistron Radius Server*, available: http://www.radius.cistron.nl (May 8, 2002).

[17] LUCENT TECHNOLOGIES: *ORINOCO OR Manager, User's Guide for Outdoor Router 1000/1100*, Nieuwegein, Nederlands: Lucent Technologies Inc., 2000.

[18] DROMS R., ARBAUGH B.: *Authentication for DHCP Messages*, RFC3118, IETF, 2001.

[19] CROFT, B., GILMORE, G.: *Bootstrap Protocol*, RFC 951, IETF, 1985.

[20] KURZ V.: *Delivery system for location based information in wireless IP networks*, Master's Thesis, Lappeenranta University of Technology, 2002.

[21] OPEN BSD: *Ssh manual page*, available http://www.openbsd.org/cgi-bin/man.cgi?query=sshd (May 8, 2002).

[22] GROOTHUIS, E.: *Dhcping*, available: http://www.mavetju.org/unix/general.php (May 8, 2002).

[23] LEY W., ELLERMAN U.: *Logsurfer*, available: http://www.cert.dfn.de/eng/logsurf/ (May 8, 2002).

[24] FRIEDL J.E.F.: *Mastering Regular Expressions*, Sebastopol, CA: O'Reily & Associates Inc, 1997.

[25] WALL L., CHRISTIANSEN T., ORWANT J.: *Pattern Matching*, in *Programming Perl* (pp. 139-216), 3rd edition, Sebastopol, CA: O'Reily & Associates Inc, 2000.

Appendix A

Application script

```
#!/bin/sh
#
# 2002 Radek Spacil,
# (C) Wireless Lappeenranta project, http://www.wlpr.net/
# Lappeenranta University of Technology
#
# script "watching" log file on remote machine
# and downloading it throug "ssh pipe"

# you shouldn't need to change following variables
USER=transfer

# if copied as master file to create new logging service
# you need to change these variables:
MACHINE=10.1.3.18               # imp2
KEY_ID=/home/$USER/.ssh/id_rsa_turgon_apache
LOG_FILE=/var/log/transfer/turgon_apache.log

ssh $MACHINE -l $USER -i $KEY_ID >> $LOG_FILE
```

Appendix B

Startup script

```
#!/bin/sh
#
# 2002 Radek Spacil,
# (C) Wireless Lappeenranta project, http://www.wlpr.net/
# Lappeenranta University of Technology
#
# Startup script for transfering files to the log server
#

# description: This script is used to run different "ssh pipes"
#              to transfer log files from different computer
#              to the log server (here)
#
# chkconfig: 345 85 15

# Source function library.
. /etc/rc.d/init.d/functions

USER=transfer
LIST_TRANSFERS="turgon-syslog imp2-apache"
PID_DIR=/var/run/
DIR=/usr/local/sbin/
SYSLOG_LEVEL=daemon.err
```

```
#DAEMON=/usr/local/sbin/$NAME


start () {
        echo -n "Starting $1: "
        if status $1 >/dev/null; then
                failure $"$1 startup"
                echo
                return 1
        fi
        su --command $DIR$1 --login $USER 2>&1 |\
           logger -t $0 -p $SYSLOG_LEVEL &

        echo $! > $PID_DIR$1.pid
        success $"$1 startup"
        echo
        return 0
}

stop() {
        echo -n "Shutting down $1: "
        killproc $1
        RETVAL=$?
        echo
        return $RETVAL
}

case "$1" in
    start)
        for I in $LIST_TRANSFERS; do
            start $I
        done
        ;;
    stop)
        for I in $LIST_TRANSFERS; do
            stop $I
        done
```

```
        ;;
    status)
        for I in $LIST_TRANSFERS; do
            status $1
        done
        ;;
    *)
        echo "Usage: $0 {start|stop|status}"
        exit 1
        ;;
esac
exit $?
```

Appendix C

Data for revealing rogue DHCP servers

C.1 Regular expression for searching DHCPREQUESTs

```
dhcpd: DHCPREQUEST for (([0-9]{1,3}.){3}[0-9]{1,3})[ \t]*
\((([0-9]{1,3}.){3}[0-9]{1,3})\)[ \t]*from (([0-9a-fA-F]{2}:)
{5}[0-9a-fA-F]{2})
```

C.2 Configuration file for *logsurfer*

```
'dhcpd: DHCPREQUEST for (([0-9]{1,3}.){3}[0-9]{1,3})[ \t]*
        \((([0-9]{1,3}.){3}[0-9]{1,3})\)[ \t]*from (([0-9a-fA-F]{2}:)
        {5}[0-9a-fA-F]{2})' '\(10.1.3.1\)' - - 0
    exec "/bin/echo DHCPREQUEST to suspicious (rogue) DHCP server
        at $4 (ip requested: $2, from mac: $6)"
```

Appendix D

dhcptaild

```
#! /usr/bin/perl
#
# dhcptaild2.pl
#
# 2002 Radek Spacil,
# (C) Wireless Lappeenranta project, http://www.wlpr.net/
# Lappeenranta University of Technology
#
# This script tails DHCPd leases and writes all IPs, MACs and lease
# times into MySQL database. The database is used by location-based
# advertisement redirector and for controlling access for subscribed
# users.

use DBI;
use Time::Local;
use Sys::Syslog;
use POSIX qw(strftime);

# Not buffered I/O
$|=1;

# Some definitions
#DHCPD_LEASES=/var/state/dhcp/dhcpd.leases
```

```
$DHCPD_LEASES = dhcpd.leases;
$MySQL_USER = dhcp;
$MySQL_PASS = 'df7-43.:23d';
#$MySQL_HOST = imp2.wlpr.net;
#$MySQL_DB = wlan_adv;
$SYSLOG_FACILITY = daemon;
$SYSLOG_LEVEL = err;

sub log_and_die {
  my ($message) = @_;

  openlog($0, 'cons,pid', $SYSLOG_FACILITY);
  syslog($SYSLOG_LEVEL, '%s', $message);
  closelog;

  $sth_replace->finish;
  $dbh->disconnect;

  die $message;
}

# Connection to MySQL server
$dbh = DBI->connect("DBI:mysql:database=WLPR;host=imp2", $MySQL_USER,
    $MySQL_PASS) || log_and_die($DBI::errstr);
$sth_replace = $dbh->prepare("REPLACE INTO leases (IP, MAC, start, expiry)
    VALUES (?,?,?,?)") || log_and_die($dbh->errstr);

# read the lease
LINE: while ($lease = <STDIN>) {
  next LINE unless defined $lease;
  next LINE if $lease =~ m/^#/;               # discard comments

  # find beginning of lease entity
  if ($lease =~ /lease (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}) \{/) {
    $ip=''; $line=''; $mac=''; $starttime=''; $endtime='';
    $ip = $1;

    do {                    # read lines till end, which is line="}"
```

```
  $line = <STDIN>;    # read next line
  # save information to match into variables
  SWITCH: {
    # START TIME
    $line =~ m/starts/ && do {
      ($startyear, $startmonth, $startday, $starthour, $startmin,
        $startsec) = ($line =~ /(\d{4})\/(\d{2})\/(\d{2})
        (\d{2}):(\d{2}):(\d{2})/);

      #convert time&date into sec since epoch for easy computing
      $starttime = timegm($startsec,$startmin,$starthour,$startday,
        $startmonth-1,$startyear);
      $startt = strftime "%Y%m%d%H%M%S", localtime($starttime);
      last SWITCH;
    };
    # END TIME
    $line =~ m/ends/ && do{
      ($endyear, $endmonth, $endday, $endhour, $endmin, $endsec) =
        ($line =~ /(\d{4})\/(\d{2})\/(\d{2}) (\d{2}):(\d{2}):(\d{2})/);
      #convert time&date into sec since epoch for easy computing
      $endtime = timegm($endsec,$endmin,$endhour,$endday,$endmonth-1,
        $endyear);
      $endt = strftime "%Y%m%d%H%M%S", localtime($endtime);
      last SWITCH;
    };
    # MAC
    $line =~ m/ethernet/ && do{
      ($mac) = ($line =~ /(..:..:..:..:..:..)/);
      last SWITCH;
    };
  };  # end of SWITCH
} until ($line =~ m/^}/) || eof;       # until end of lease entity

# now all neccesery info is in variables (for this particular lease),

#check if IP and MAC are not empty
if ($ip && $mac) {
  $sth_replace->execute($ip, $mac, $startt, $endt) ||
```

```
      log_and_die($sth_location->errstr);
  };

  };          # end of 'if' match 'lease x.x.x.x {'
};           # end of 'while' - parsing file till the end

$sth_replace->finish;
$dbh->disconnect;
```