Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Deploying open source routers

A low-cost return on investment

R. Carbone
DRDC Valcartier

Defence R&D Canada – Valcartier
Technical Note
DRDC Valcartier TN 2006-630
November 2006

Canada

Deploying open source routers

A low-cost return on investment

R. Carbone
DRDC Valcartier

Defence R&D Canada – Valcartier

Technical Note
DRDC Valcartier TN 2006-630
November 2006

Principal Author

Richard Carbone

Programmer/Analyst

Approved by

Yves van Chestein

Head/Information and Knowledge Management

Abstract

Network routing can be a difficult concept to grasp as can the configuration and deployment of routers. As with any specialized equipment, time and understanding are required. Specialized equipment also tends to be expensive. Under the auspice of the Urban Operations project, due to budgetary constraints, a custom Linux-based router was built for interconnecting multiple networks together. While routing with Linux is not any more difficult than with any other operating system, the problem was establishing how to implement and deploy it with three or more interconnected networks. All of the existing documentation was written for two networks and dial-up connections. Through trial and error, one method among many other possibilities was discovered. This method is described in this document and focuses on the inherent capabilities of the Linux operating system. In sharing this work with others it becomes possible for them to expand on it and make better use of it. Linux is a powerful and flexible operating system that is ready for C2 application and this document demonstrates different ways in which it can be deployment for military purposes.

This page intentionally left blank.

Executive summary

Deploying open source routers: A low-cost return on investment: A low-cost return on investment

Carbone, R.; DRDC Valcartier TN 2006-630; Defence R&D Canada – Valcartier; October 2006.

While a commercial router was preferred for interconnecting three disparate networks together, budgetary constraints dictated otherwise. Thus, a PC-based Linux router was implemented in order to provide the necessary network connectivity for the Urban Operations project. A development lab, a contractor's private network, and the corporate network had to be linked together. However, security was paramount and flow control of the data and network stream was imperative.

After researching how this could be done, it was found that a valid manner to do so would have to be different from the methods proposed in the various consulted documentation. The documentation was nevertheless an excellent starting point. While many operating systems were examined, Linux was the easier due to in-house experience with it.

The result was a PC-based router that could be extended for connecting as many physical networks as there were physical interfaces on the router. While commercial routers can also do the same, the price is in the thousands. Linux, while free, has all of the necessary tools required to provide firewalling, routing, and web caching.

Following the procedure developed in this text, it becomes possible for others to create their own similarly configured router. Sections can be skipped, depending on the requirements. While networking and routing concepts, as well as TCP/IP, can be difficult to grasp, knowing their underlying precepts are not required. Anyone familiar with the basic of networking and routing could easily create a powerful and versatile routing system. Thus, it is now possible to create an *N*-interface routing system. The possibility has always been there and has no doubt been used before; the catch is that no existing publicly available documentation was available for it. It is now possible to deploy Linux-based routers that are suitable in C2 environments.

The router described in this document is no longer in use in Urban Operations due to changes in the needs of the contractor and his network and software lab. However, the Linux router is being fielded in the JCDS21 TDP.

This page intentionally left blank.

Table of contents

List of figures

List of tables

Acknowledgements

The author would like to thank René Gagnon and Steve Jarvis, two co-op students who helped make this work possible while completing their internship.

The author would like to thank Martin Salois for his hard work in revising this document to ensure its accuracy and relevancy.

Disclaimer

While open source technology is useful and can aid in solving many technological problems, it is not currently approved for implementation on DWAN, the Defence Wide Area Network, nor is it approved for CNET, the Classified Network. Any implementation of Linux, Linux routers, or any other open source technology must approved by the appropriate individuals/supervisors and/or the appropriate persons in the chain of command. By no means does the author specifically endorse or advise that Linux-based/open source-based routers are better suited or more /less secure than their commercial equivalents.

These networks also require a strict change management process and re-accreditation. Thus, changes to these networks are generally neither a trivial nor a simple task. That is why changes must be made with care and have been well thought out.

Finally, the purpose of this document is not to advocate the usage of these technologies but rather to advise, explain, and demonstrate their usage and potential capabilities so that those required to make technological decisions are better able to do so.

1. Implementation

1.1 Introduction

For the Urban Operations project, due to the needs of the contractor, it was required to interconnect three disparate networks together. It was required to connect the contractor's software development lab, a private network, and the corporate network. Due to the nature of the software being developed and tested in the software lab, it was necessary that it be isolated.

While commercial solutions were available, their cost was prohibitive. From the software lab, the contractor required access to the corporate network in order to access certain pre-existing network-based services such as file storage, Internet access, *CVS*, *Oracle* database, and backup storage systems. This could have been done using a very simple router. Shortly thereafter, the contractor required access to the software lab from his own private network. While two simple network routing appliances would have been sufficient, it was nevertheless considered best to perform all network routing operations from a single system. Thus, it was decided to build the router using Linux on an existing older computer.

Configuring Linux for dual-networking router is simple; configuring it for more than two interfaces was not. The majority of the documentation consulted for ascertaining how to do so came from The Linux Documentation Project (TLDP). Here, many high-quality documents on Linux and network specific configuration-oriented tasks could be found. However, nothing could be found on three or more interface-based routers. The router was built essentially by trial and error, and the experiences and procedures have been documented here to help others perform the same task. The available documentation found on TLDP was nevertheless a good starting point.

Much of the job had been given to two Winter-term co-op students; while they were able to complete much of the work, unfortunately, due to their lack of familiarity with the Linux environment some of the final portions had to be completed by the author. Despite their lack of familiarity with the operating system, Linux was the operating system of choice due to much in-house experience with it.

1.2 Requirements

While building a three-interface router was difficult enough, to complicate matters further, network isolation would have to be implemented via a firewall on the router in order to block any access to the corporate network from the contractor's lab, apart from the three services described in the next paragraph.

The main resource to be accessed on the corporate network was a UNIX server that provided file storage, network authentication, backup storage facilities, and *CVS*. There was also a lesser used Linux server that was also needed. Internet access was also required.

The router must be the central point in the three-network system. The contractor must be able to connect to the software lab from his network, but must not be able to connect to the corporate network at all. The software lab must connect to either the contractor's lab or the corporate

network, and certain systems from the corporate network may connect to the software lab, but no corporate system must connect to the contractor's lab.

Because each network was using a different class of network, no special routing schemes had to be implemented. The contractor's lab, a standard Class C network was using an addressing scheme of 192.168.1.0/255.255.255.0. The software lab, a standard Class A network was using an addressing scheme of 10.0.0.0/255.0.0.0. The corporate network, a Class B network with subnetting network was using an addressing scheme of 142.145.32.0/255.255.240.0.

It is imperative that DHCP addresses not be used on the Linux router on any of its interfaces. Doing so will adversely affect the system hostname, NAT, web caching and other proxy-related services.

Thus, the Linux router would also use a firewall and implement Network Address Translation (NAT) to successfully fulfill the requirements.

1.3 Installing Linux

For the Linux installation, Red Hat 9 (RH9) Professional was chosen. While older, it was found to be stable, robust, and well supported by many of the TLDP documents.

The RIP (Routing Information Protocol) protocol was chosen rather than OSPF (Open Shortest Path First) due to its simplicity. RIP is static and works well for networks that do not change very much, such as was the case for the three networks to be connected together. Furthermore, RIP is easier to configure than OSPF. Other valid choices could have involved the use of *Zebra*, *Gated*, or *IP Route*.

The *GRUB* boot loader was used rather than *LILO* and a *GRUB* password was implemented. Before starting the installation, it is a good idea to place a password on the computer's BIOS. Booting from CD-ROM, a text-based installation was chosen. Mouse and keyboard configuration will vary according to the language and device type. Time zones will also vary. For this installation, an English installation was selected, as was the EST5EDT time zone.

Specifically, RH9 was installed onto a dual-Pentium Pro 200 MHz system with 256 MB RAM. Three disks were available in the system and they were partitioned as follows using *Disk DruidI* and formatted using the Ext3 filesystem.

Table 1: Disk layouts for partitioning the system during installation

Disk #	**Disk Size**	**Mount Point**	**Partition Size**
Disk 1 (IDE):	1.2 GB	/boot	512 MB
		swap	675 MB
Disk 2 (SCSI):	4.0 GB	/	3500 MB
		swap	500 MB

Disk 3 (SCSI):	4.0 GB	/usr	2000 MB
		/var	750 MB
		swap	250 MB

Networking was implemented using the TCP/IP information provided in Section 1.2. This information can be found in Table 2. The system was given a hostname of "router." The default installation RH9 firewall was not implemented. For system authentication, local system authentication was selected along with the use of MD5 and shadow passwords. The full distribution of packages was installed, containing about 1400 installed packages and just over 4.0 GB in size.

Table 2: Network interface configuration

Network Interface	**Activate on Boot**	**Configuration Information**
Eth0	Yes	IP Address: 142.145.32.165 Netmask: 255.255.240.0 Gateway: 142.145.32.1 Primary DNS: 142.145.32.2
Eth1	Yes	IP Address: 10.0.0.1 Netmask: 255.0.0.0
Eth2	Yes	IP Address: 192.168.1.20 Netmask: 255.255.255.0

The appropriate graphics card and monitor were configured. Once the installation was complete, the system was rebooted. The installation of RH9 is rather basic. It is outlined in more detail in [1].

To deny rebooting the system from the keyboard Ctrl-Alt-Del hotkey, the following line from the file */etc/inittab* must be commented out by placing the symbol # at the beginning of the line:

```
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
```

To prevent users from gaining root access by booting into single-user mode, in the same file, immediately after the aforementioned line, add this line:

```
~~:S:wait:/sbin/sulogin
```

1.4 Network configuration and routing tables

Under Linux, as in many other UNIX operating systems, network configurations can always be changed at any time. The tool of choice is the command line tool *ifconfig* and it is widely regarded as one of the most powerful of the UNIX system administration commands. This command requires root access for making changes to the system.

Linux, like UNIX, treats devices as files. Ethernet-based interfaces under Linux are recognized as the file */dev/eth0*, */dev/eth1*, and */dev/eth2*, respectively. The devices are logical devices[1] and represent the physical Ethernet network devices with respect to their position on the physical PCI bus. Other operating systems such as Solaris 8 (and higher) would recognize Ethernet interfaces as */dev/hme0*, and */dev/hme1*, etc. Logical operating system devices are always stored under the directory */dev*. Under UNIX, the general rule of thumb is that the first device in a suite of other similar devices ends the filename with a "0" or an "a" (sometimes both are used), and progressively move to higher and/or combinations.

The command *ifconfig* allows for modification of the networking parameters as related to the logical networking devices. The networking information should already have been specified during installation (Section 1.3); however, requirements can change and so can network addresses.

Because each interface will route packets emanating from other interfaces on the router, these interfaces, at the very minimum, require gateway addresses defined for each network card. These gateways specify where internal packets from one card are to be sent to when leaving that interface. For this router configuration, each network interface points to itself, thus each network packet from a given network knows which address provides a link outside of the immediate interface. Put differently, each interface is the in and out for each network connected to it. While during installation, each network card was given an IP address, only *eth0* was given a gateway. In order for routing to work, at the very least, all the network cards must have a gateway defined.

Thus, whether to change or modify the interface's network information, the *ifconfig* command will allow for any of these changes to be made. Using various methods[2], it is possible to calculate any missing pieces for the networking information required for each interface. The command *ifconfig* requires as much information as can be possibly provided. The following commands, in this specific order, are required to correctly configure the system before routing can be enabled:

For *eth0*:

```
$       ifconfig eth0 142.145.32.165 netmask 255.255.240.0 broadcast 142.145.47.255

$       route add –net 142.145.32.0 netmask 255.255.240.0 gw 142.145.32.165

$       route add default gw 142.145.32.1
```

[1] To the machine itself, these devices are physical, but to the operating system, these devices are logically denoted and addressed.

[2] The Internet offers many different web sites that are able to perform these calculations; all that is required is some basic pieces of information and all the rest of the data is automatically calculated. http://www.subnetmask.info is just such a web site.

For *eth1*:

$ ifconfig eth1 10.0.0.1 netmask 255.0.0.0 broadcast 10.255.255.255

$ route add –net 10.0.0.0 netmask 255.0.0.0 gw 10.0.0.1

For *eth2*:

$ ifconfig eth2 192.168.1.20 netmask 255.255.255.0 broadcast 192.168.1.255

$ route add –net 192.168.1.0 netmask 255.255.255.0 gw 192.168.1.20

While it has not yet been explicitly mentioned, because Internet access will occur from the corporate network, to which interface *eth0* is connected to, in order for any other of the router's interfaces to be able to access the Internet, *eth0* must have a default gateway defined. This is the purpose of the third line for the *eth0* commands. The software lab will be attaining Internet access from *eth0*. It is perfectly valid for a network interface to have multiple gateways; however, only one gateway can be the system's default gateway. Because it is too unrealistic to manually define the static points between the router and the Internet, the default gateway takes care of this issue. This could also be done using certain traffic shaping tools[3] included with Linux.

To make these seven commands permanent, it is required that they be added to the system post-boot initialization file */etc/rc.local* by appending them to the bottom of the file.

To enable routing under Linux, two items must be present. The first, the Linux kernel must be configured for routing. RH9's kernel has already been pre-compiled with this option. This option is known as *IP Forwarding*. The second item is to enable the kernel's routing structure by performing the following command:

$ echo 1 > /proc/sys/net/ipv4/ip_forward

The */proc* directory is where all the kernel-related structures exist. The structures are recreated at boot time from various system configuration files. To avoid performing this task at each system boot, the command can be appended to the file */etc/rc.local*.

Routing requires the use of routing tables. These tables define where data comes in from and where it goes out to. Furthermore, they define how this occurs. Under RH9's, networking information can be found under the directory */etc/sysconfig/networking/devices.* There, six files contain all of the necessary network information, including routing tables. They are:

eth0.route

eth1.route

eth2.route

[3] Certain forms of traffic shaping and redirection can be accomplished using firewalls and network address translation.

ifcfg-eth0

ifcfg-eth1

ifcfg-eth2

The **.route* files contain the static routing information. The *ifcfg-** files contain the network configuration information. Thus, each network interface requires two configuration files: one file to set the interface's network parameters and the other to set up the static route. The contents of the two files are presented in Table 3 and Table 4.

In these tables, it is important to notice the similarities between the static routes of each network interface. They have been colour coded to make this task easier. All three files and thus all three interfaces share the same basic three routes. Notice however, that the file *eth0.route* has one additional route that the others do not; this is represented by a colour not shared by the other files. These files recreate the seven commands entered in earlier. These files could also be modified to make the changes made on the command line earlier.

Table 3: Static routing configuration files

File eth0.route	**File eth1.route**	**File eth2oute**
GATEWAY2=192.168.1.20	GATEWAY2=142.145.32.165	GATEWAY2=142.145.32.165
NETMASK2=255.255.255.0	NETMASK2=255.255.240.0	NETMASK2=255.255.240.0
ADDRESS2=192.168.1.0	ADDRESS2=142.145.32.0	ADDRESS2=142.145.32.0
GATEWAY1=10.0.0.1	GATEWAY1=192.168.1.20	GATEWAY1=192.168.1.20
NETMASK1=255.0.0.0	NETMASK1=255.255.255.0	NETMASK1=255.255.255.0
ADDRESS1=10.0.0.0	ADDRESS1=192.168.1.0	ADDRESS1=192.168.1.0
GATEWAY0=142.145.32.165	GATEWAY0=10.0.0.1	GATEWAY0=10.0.0.1
NETMASK0=255.255.240.0	NETMASK0=255.0.0.0	NETMASK0=255.0.0.0
ADDRESS0=142.145.32.0	ADDRESS0=10.0.0.0	ADDRESS0=10.0.0.0
GATEWAY=142.145.32.1		
NETMASK3=255.255.240.0		
ADDRESS3=142.145.32.165		

Shown in Table 4 are the contents of the *ifcfg-** files. These files contain the necessary start-up parameters to correctly initialize all of the network interfaces. It should now be possible to understand how to manually modify these files to accomplish the necessary routing features.

Table 4: Network interface parameter configuration

File ifcfg-eth0	**File ifcfg-eth1**	**File ifcfg-eth2**
DEVICE=eth0	DEVICE=eth1	USERCTL=no

BOOTPROTO=none	ONBOOT=yes	PEERDNS=yes
ONBOOT=yes	BOOTPROTO=none	TYPE=Ethernet
IPADDR=142.145.32.165	IPADDR=10.0.0.1	DEVICE=eth2
NETMASK=255.255.240.0	NETMASK=255.0.0.0	BOOTPROTO=none
USERCTL=no	USERCTL=no	NETMASK=255.255.255.0
PEERDNS=yes	PEERDNS=no	ONBOOT=yes
TYPE=Ethernet	TYPE=Ethernet	IPADDR=192.168.1.20
NETWORK=142.145.32.0	NETWORK=10.0.0.0	NETWORK=192.168.1.0
BROADCAST=142.145.47.255	BROADCAST=10.255.255.255	BROADCAST=192.168.1.255
GATEWAY=142.145.34.165	GATEWAY=10.0.0.1	GATEWAY=192.168.1.20

Therefore, it is through the creation of the static routing tables that it is possible to route between three and more interfaces. This is a concept that was not found in any of the consulted documentation at TLDP.

The static routing tables are now complete. Obviously, they will vary from system to system and from requirement to requirement. What is important is to know where to make the necessary changes and what types of changes to make to achieve the routing objective. Certainly, there are other ways to create routing tables. Unfortunately, these require the use of more sophisticated routing tools and protocols that is outside the scope of this document.

Routing at this point is not quite yet operational. The routing daemon must be enabled. To do so, run the following command:

$ /etc/init.d/routed start

This will start the Linux RIP routing daemon. It is also possible to run the OSPF routing daemon, but again, this is far more complex than is needed by the current requirements. To avoid typing this command every time after start-up, it can be appended to the file */etc/rc.local*. The daemon can also be enabled by configuring the system's current runlevel services. RH9 provides two runlevel service configuration tools: *ntsysv* and *chkconfig*. The former is a command line tool while the latter is an X Windows enabled tool.

The final issue that is of interest is to examine the system's routing table. This can be done by running the:

$ *netstat –rn*

This will print out the various network routes as shown in Table 5. Notice the bold text. This shows the results of the seven commands used to configure the network interfaces at the beginning of this section. Gateways are denoted by "UG." The data presented below is representative of the files found in */etc/sysconfig/networking/devices*.

Table 5: Kernel IP routing table

Destination	Gateway	Genmask	Flags	MSS Window	irtt	Iface
192.168.1.0	**192.168.1.20**	**255.255.255.0**	**UG**	**0 0**	**0**	**eth2**
192.168.1.0	0.0.0.0	255.255.255.0	U	0 0	0	eth2
142.145.32.0	**142.145.32.165**	**255.255.240.0**	**UG**	**0 0**	**0**	**eth0**
142.145.32.0	0.0.0.0	255.255.240.0	U	0 0	0	eth0
10.0.0.0	**10.0.0.1**	**255.0.0.0**	**UG**	**0 0**	**0**	**eth1**
10.0.0.0	0.0.0.0	255.0.0.0	U	0 0	0	eth1
127.0.0.0	0.0.0.0	255.0.0.0	U	0 0	0	lo
0.0.0.0	**142.145.32.1**	**0.0.0.0**	**UG**	**0 0**	**0**	**eth0**
0.0.0.0	**192.168.1.20**	**0.0.0.0**	**UG**	**0 0**	**0**	**eth2**

Routing is now configured and working correctly. At this point, the physical networks can be connected to the router; however, no network routing security has yet been enabled.

1.5 Configuring the network clients

After connecting the networks together, network connectivity tests must be performed to verify if the various networks and their systems are accessible. This was done by placing a test client machine on each network. Client 1 is connected to the software lab with IP address 10.0.0.2/255.0.0.0 and is a Windows 2000 workstation. Client 2 is connected to the contractor's network with IP address 192.168.1.21/255.255.255.0 and is a Linux workstation. Client 3 is connected to the corporate network with IP address 142.145.32.55/255.255.240.0 and is a Windows 2000 workstation.

The appropriate gateway information had to be configured for each workstation. Clients 1, 2, and 3 used gateways 10.0.0.1, 192.168.1.20, and 142.145.32.165 (the router's three network interfaces), respectively. Client 1 could ping all the router's interfaces but could not succeed in pinging Client 2 or 3 (192.168.1.21 and 142.145.32.55). The same was true of systems Clients 2 and 3. Furthermore, all these systems required a reconfiguration of their network settings. Gateway and subnet information had to be setup correctly. The systems also had their DNS configured to point to 142.145.32.150, the local DNS server on the corporate network. The workstations were then rebooted. Upon reboot, they could successfully ping each other and all of the router's three interfaces.

To further test connectivity, a web server was enabled on Client 2 using *Apache*. The Telnet service was enabled on clients 1 and 3. Once all three clients were appropriately configured, Client 2 could successfully *Telnet* to clients 1 and 3, and they could both access Client 2's default web page. *Thus, it is ascertained that routing does in fact work.*

However, for clients 1 and 2, communicating with systems on the corporate network other than Client 3 was another matter altogether. This is because packets leaving the router's corporate network interface could not be sent back to this interface because the destination systems knew nothing about its existence. The systems on the corporate network instead had a different default gateway that was the address of the network's main router used to provide interconnectivity throughout the corporate network and provide Internet access. Using a packet-sniffing tool, it was confirmed that packets from clients 1 and 2 were reaching their destinations; the packets just did not know how to get back to their source. In order for communications to be correctly established from the two internal networks, the client systems would have to have their static routing tables adjusted. Because only two systems were required on the corporate network for various services (a Linux PC and a Sun datacentre, 142.145.32.74 and 142.145.32.150, respectively), their routing tables would have to be modified. The Sun system is also the DNS server. While it was confirmed by its system logs that DNS requests had been received from clients 1 and 2, the responses had never been received.

Client 3 could communicate with systems on the local subnet only. Systems on other subnets of the corporate network, including Internet access did not work.

Therefore, the routing tables of systems 142.145.32.74 and 142.145.32.150 had to be modified to configure two-way communications. This is done by running the following commands:

Linux PC (142.145.32.74):

```
$       route add -net 10.0.0.0 netmask 255.0.0.0 gw 142.145.32.165

$       route add -net 192.168.1.0 netmask 255.255.255.0 gw 142.145.32.165

$       route add -host 142.145.32.55 gw 142.145.32.165
```

Sun (142.145.32.150):

```
$       route add -net 10.0.0.0  142.145.32.165

$       route add -net 192.168.1.0  142.145.32.165

$       route add -host 142.145.32.55 gw 142.145.32.165
```

These commands tell the Linux PC and the Sun datacentre that there are other networks that are to be connected to via an alternative gateway on the existing corporate network. From the consoles of both these systems it was then possible to ping the router's three interfaces (142.145.32.165, 10.0.0.1, and 192.168.1.20) and successfully access the web server on Client 2 (192.168.1.21) and access the Telnet services running on clients 1 and 3 (10.0.0.2 and 142.142.32.55). To make these routing changes permanent, these commands could be entered into the files */etc/rc.local* (Linux) and */etc/init.d/inetsvc* (Sun).

Without any network filtering on the Linux router yet implemented, and since, at this time, security is not yet important, it must be understood that by configuring the various clients/servers on the corporate network with additional routing tables opens up holes in network security.

These systems now know about the existence of the secondary gateway and the networks behind the Linux router. This allows the contractor's network to access the corporate network which goes against the security requirements stated in Section 1.2; this issue will be dealt with in Section 1.9.

However, since it could occur that a Windows-based system on the corporate network may eventually need access to the private networks behind the Linux router, it is important to know how to configure the Windows systems for communications with a secondary gateway. Thus to provide access to both the router's internal networks (software lab and contractor's network) as well as to Client 3, the following commands are required:

C:\> route add 10.0.0.0 mask 255.0.0.0 142.145.34.165

C:\> route add 192.168.1.0 mask 255.255.255.0 142.145.34.165

C:\> route add 142.145.32.55 142.145.34.165

For the first two commands, the local system is instructed to communicate with the designated secondary gateways in order to establish communications with those networks behind the gateways. As for Client 3, while it resides on the corporate network, communications must be sent to its gateway in order to correctly establish the communication stream with it. Once the Windows system was configured with these three commands, all three client machines could be pinged and their services (web and Telnet) were accessible. To make routes permanent under Windows, use the *route –p* command.

1.6 Case for using proxy-based technology and NAT

While network connectivity was established, Internet access for the clients did not work because the Linux router was unable to communicate with the subsequent routers found on the corporate network that establish the link to the Internet.

Internet access was not possible for two reasons. The first reason is due to the routers in place between the Linux router and the final relay to the Internet found along the corporate network. The second reason is that even if the network routers were configured to accept connections from the Linux router, IP addresses that were not specific to the corporate network would ultimately be rejected.

Two relatively simple and commonplace technologies exist to get around this network limitation. They are web caching via a proxy server and NAT. These technologies will make it possible to access the Internet without having to implement more complex network technologies such as Proxy ARP.

In both instances, whether using a proxy server or NAT, the rest of the corporate network must be led to believe that the connections to the Internet are coming from the Linux router and not from systems behind it or connected to it.

1.7 Configuring the proxy server

For a web caching proxy server, there exists a powerful open source utility, *Squid*. It is a free, high performance web caching server that is available for (and compiles on) many different platforms (including Windows). It can be used to fulfill a variety of requirements. Depending on these requirements, it can be use to cache requests for subnets or entire networks. It can also be used to cache not only DNS and FTP requests, but also Gopher, SSL, and SNMP requests. It can even help speed up Internet requests, an advantage for organizations with slower Internet connections.

Due to its flexibility, *Squid* can be used in many places; however, it can also be difficult to setup and configure. However, *Squid* is easier to configure than firewalling or NAT. For the Linux router, *Squid* was configured as a *Reverse Proxy Web Caching* server. This type of proxy system acts as a proxy server for the actual web server. Thus, from the viewpoint of a client system, reverse proxy acts more like a web server than a proxy server. This causes the destination URL to be routed through the reverse proxy where web content is then streamed back to the client. This is also known as a *Web Server Accelerator* because it frees up the "real" web server to handle dynamic content while the caching server feeds the static content to the clients. Because the *Reverse Proxy Web Cache* server is on the Linux router, it can now also act as a gateway.

However, there are other types of proxy servers. Another form is the *Standard Proxy Web Cache*. It is different from a *Reverse Proxy Web Cache* in that its task is to cache web pages and content only after a first request has been made by a client. The first request fetches the real data off a real web server, and the network stream is simultaneously downloaded to both the client and the proxy server. The other form is a *Transparent Proxy Web Caching* server, which as its name implies is completely transparent. These systems are generally found on the bulk of network gateways and routers. These systems require no changes to the client's browser because they automatically filter and intercept all web-based traffic. Under Linux, a transparent proxy can be created using the *Ipchains* or *Iptables* firewalling tools.

A reverse proxy web cache requires that a client system point its browser to the Linux router specifying the network port where *Squid* is listening. The proxy server listens for incoming connections on a specific network port; there the various connections are performed on behalf of the client. The network connection is then transmitted from the proxy server to the client system. It is not difficult to configure modern clients systems for this; almost all modern browsers support this feature. Furthermore, all requests done by the proxy server for a client system is completely transparent to the web server; it has no indication that anything is out of the ordinary.

Squid comes pre-packaged with many different Linux operating systems. If not included, it will compile on almost every UNIX-like operating system. It is suggested to run *Squid* as a non-root user to avoid a system from being compromised. Under RH9, a user and group "squid" has already been created for this purpose.

The default location for the caching directory is under */var/spool/squid*. Its main configuration file is */etc/squid/squid.conf*.

The initial configuration step is to verify if the proxy server's hostname is equal to its DNS record. If they are not the same, then *Squid* will complain with the following error message:

"FATAL: Could not determine fully qualified hostname. Please set 'visible hostname'."

This can be fixed by modifying the following entry in the *Squid* configuration file:

from:

visible_hostname

to:

visible_hostname *localhost_name*

"Localhost_name" is the system name given to the system during installation. It can also be found in the system configuration file */etc/hosts*.

The complete *Squid* configuration file as configured for the Linux router can be found in Annex B.1.

Squid is started up by running the command:

$ squid -N -d 1 -f /etc/squid/squid.conf

Its output will vary depending on the configuration options used. In Annex B.2, a sample output of *Squid* can be found. The system script */etc/init.d/squid* can also be configured for execution at start-up using *ntsysv* and *chkconfig*.

To configure the browser on various clients, set the client to use a proxy server by specifying the IP address of the proxy server and the listening port. This is shown in Figure 1. While Figure 1 is from a Windows-based system, it equally applies to other browsers.

Once the browsers had been configured on the client systems (clients 1, 2, and 3), they were able to access the Internet without trouble. As far as the corporate network is concerned, all requests appear to be coming from the Linux router itself (142.145.32.165). The clients were tested out using web sites with combinations of static and dynamic content. There was an apparent slowdown of around 10% in all web-based transactions. Many factors can influence *Squid's* speed: the speed and power of the Linux router; the amount of dedicated cache memory (physical RAM); the amount of additional swap space for the cache, and the speed of the disks used for writing the cache to disk.

The web server on Client 2 (192.168.1.21) was re-enabled and from both clients 1 and 3 (10.0.0.2 and 142.145.32.55) the web server was found to be available, and the request was cached by *Squid*. The speed was virtually instantaneous and all HTTP requests were correctly cached.

In today's world of heightened security requirements, it is reassuring to know that *Squid* can handle SSL requests. Various SSL sites on the Internet were tested and worked without incident. Thus, interoperability between *Squid* and SSL is not an issue.

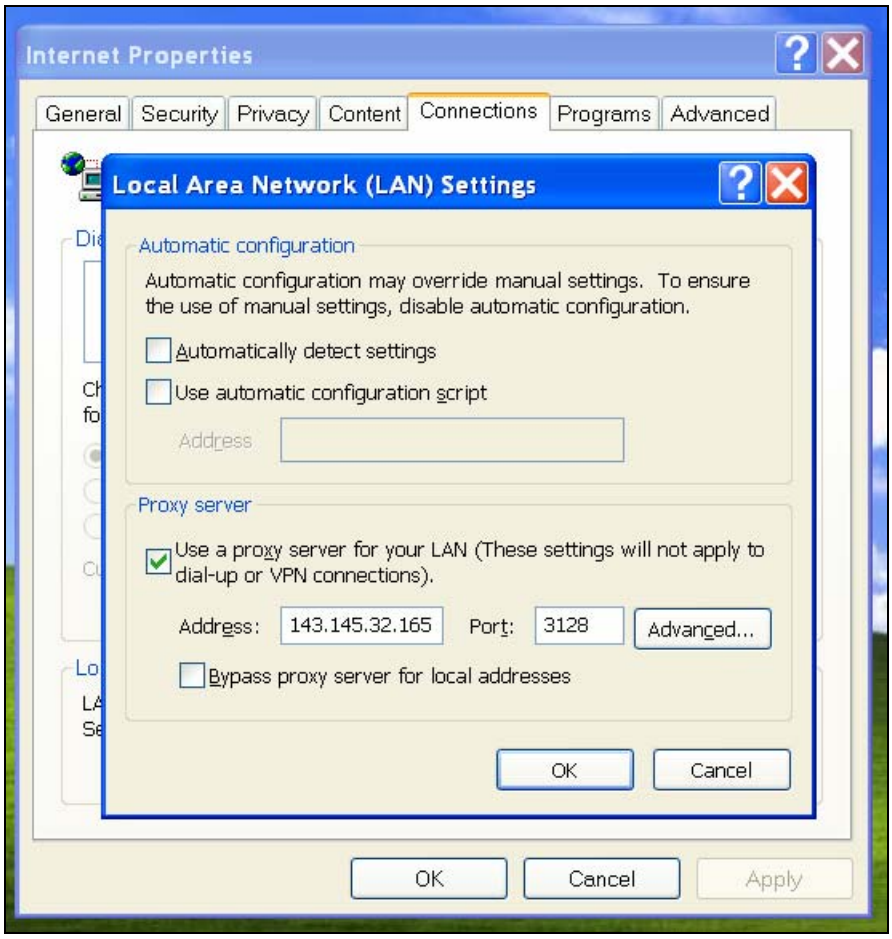

Figure 1: Configuring Internet Explorer's proxy server feature

Unfortunately, however, it was found that certain FTP requests made outside the corporate network on the Internet did not work correctly. This was due to the way the main Internet-connecting firewall on the corporate network was configured to handle FTP streams. The problem could be bypassed by pointing a browser to the FTP address and replacing FTP in the URL with HTTP as seen in this example:

Original URL:

ftp://ftp.redhat.com

New URL:

http://ftp.redhat.com

However, this causes the browser to open up a FTP connection using the HTTP protocol. In addition, not all FTP sites actually support this feature. Thus, in order to provide seamless FTP capabilities, it became necessary to download and install an additional tool. This tool is an open

source program called *Frox* that can relay and cache FTP requests. Simply put, it is a FTP relaying tool. It can also be used to relay HTTP requests; however, this is best left to *Squid*.

The package is downloaded from http://frox.sourceforge.net, copied to */tmp*, and compiled as follows:

$ ./configure --enable-transparent-data --enable-local-cache --enable-libiptc

$ make install

This compiles *Frox* with the options necessary to perform FTP relaying. Once compiled and installed, it is necessary to create a temporary directory where *Frox* will store its cached data; directory */var/spool/frox* was used for this purpose. It is necessary to change the permissions and ownership of the directory using these commands:

$ chown –R squid:squid /var/spool/frox

$ chmod 775 /var/spool/frox

The *Frox* configuration file *frox.conf*, found in the source subdirectory *src/*, is copied to */usr/local/etc*. The *Frox* configuration file used for the Linux router can be found in Annex B.3.

Frox requires that IP relaying be configured. This is done by relaying all incoming/outgoing FTP requests on port 21. The requests to/from port 21 are converted to 2121 where *Frox* will be able to intercept all forms of this traffic via the kernel's TCP/IP stack. This can be done using *Iptables* and NAT. This is done on a per client basis (although entire networks could be specified instead).

Before enabling FTP relaying, the NAT rules buffer on the kernel's TCP/IP stack should be cleared. This is done using the following command:

$ iptables --flush

$ iptables -t nat --flush

To enable FTP relaying for clients 1, 2, and 3, the *Frox* documentation [2] details these commands as:

$ iptables -t nat -A PREROUTING -p tcp -s 192.168.1.21 --dport 21 -j REDIRECT --to 2121

$ iptables -t nat -A PREROUTING -p tcp -s 10.0.0.2 --dport 21 -j REDIRECT --to 2121

$ iptables -t nat -A PREROUTING -p tcp -s 142.142.32.55 --dport 21 -j REDIRECT --to 2121

To make these commands permanent after start-up, they can be added to the file */etc/rc.local* (See Annex A.1). These three commands, when combined together, form an easy to use and easy to understand firewall script. It is important to remember that under Linux NAT is achieved using the standard firewall tools.

Testing various FTP connections both external and internal to the corporate network, they were all found to work correctly. Furthermore, performance was not noticeably different. These tests were made with clients 1, 2, and 3 (10.0.0.2, 192.168.1.21, and 142.142.32.55). Both in downloading and uploading files via FTP no discernable issues could be ascertained.

1.8 Squid and Microsoft Exchange

It was found that HTTP-based requests to a MS (Microsoft) *Exchange* server did not work. MS *Exchange* supports HTTP-based connections for use in VPN environments so that users can access their e-mail through an intuitive GUI. As is, however, *Squid* is unable to provide the connection. This is due to the way that NTLM, a Windows-based authentication protocol, has been written. There are third-party source codes that can be compiled against *Squid* to provide most of the necessary functionality, it can also be easily provided by implementing NAT. This is examined in the next section.

1.9 Configuring NAT

This section is by no means an all-inclusive tutorial. For detailed information, a thorough consultation of [3] is required. Furthermore, [3] is considered as the authoritative reference on NAT and Linux firewalling.

NAT can be used to provide seamless access to other networks' resources, including Internet access, without the need for gateway-based software. It can also be used for creating network ACL's (access control list). NAT can also eliminate NTLM-based connectivity issues with systems such as MS *Exchange*. Through NAT, it is possible to do away with caching and other relaying technology altogether. However, implementing NAT, like any firewall rule-based mechanism, is not a trivial matter. Before proceeding, *Squid* and *Frox* must be disabled, as well as the *Iptables* rules used by *Frox* that may have been added to the file */etc/rc.local*. If these entries do exist, they should be removed before proceeding else they will conflict with the new ACL's. While using *Squid* and *Frox*, it was not possible to cache or relay other services such as *CVS* or *Oracle*; however, using SNAT, it became possible. Various other services accessible on the Linux PC and Sun datacentre (142.145.32.74 and 142.145.32.150) will now be fully accessible.

NAT is a very flexible and powerful mechanism found in the Linux firewall toolkit. Today, *Iptables* is the primary tool; however, *Ipchains* has been kept around for compatibility. The *Ipchains* tool is far less powerful than the more modern *Iptables*, and it should only be used if necessary. Furthermore, *Iptables* is more flexible and it can be used to build very complex firewalls to provide access to, modify, and block many additional network services and resources. More information on firewalls is available from [3] and the TLDP.

There are two forms of NAT: Source NAT (SNAT) and Destination NAT (DNAT). For the Linux router, SNAT will be used. SNAT is used when the IP addresses of internally connected systems are converted to the IP address of a router's external interface, or the interface that will provide access to a desired resource on an external network. In SNAT (and DNAT), the allowed/unauthorized ports must be explicitly defined. For simplicity, on the Linux router, it was decided to make all the NAT rules all-inclusive for the entire range of permissible IP ports; the

standard range is ports 1 to 65535. Whether this constitutes a security issue is open to interpretation and it must be defined by the specific requirements. The specific blocking of ports was not desired; instead, the blocking of IP addresses was required.

DNAT is essentially the inverse of SNAT. All incoming packets from the external interface are converted to the IP address of the other internal interfaces so that to the internal clients it appears as if the remote system(s) were on the same network segment.

Unfortunately, however, firewall rules in general are difficult to implement and become increasingly difficult as they are tasked to do more.

The required ACL's are such that the contractor's lab should never be able to access the corporate network, and vice versa. However, the software lab should be able to access either of these two networks. The contractor's lab also requires access to the software lab. SNAT (as compared to DNAT) also has the added inherent advantage of blocking IP addresses from external networks (i.e. the corporate network) from communicating directly with the internal network(s)[4] [3]. Thus, the ACL's for communications initiated from the corporate network do not have to be written; only the set for communications coming from the contractor's network. However, depending on requirements, two-way communication between these two types of networks may be necessary. In this case, neither SNAT nor DNAT will be suitable.

Before having implemented SNAT, it was possible for systems on the corporate network to communicate with systems connected to the Linux router's private networks. Those systems only had to have their routing tables adjusted to reflect a secondary gateway on the corporate subnet. However, SNAT changed that; this ability is no longer available because of it. Thus, SNAT has changed the Linux router from a mere router to a firewall[5].

Remember that the software lab (network 10.0.0.0) was connected to the Linux router's *eth1* interface and that the contractor's lab (network 192.168.1.0) was connected to the router's *eth2* interface. Start by flushing any existing firewall and NAT rules [3]:

Step 1:

```
$       iptables --flush

$       iptables -t nat --flush
```

The required SNAT rules for defining access are defined in [3] as follows:

Step 2:

```
$       iptables -t nat -A POSTROUTING  -o eth0 -j SNAT --to 142.145.32.165
```

[4] There are ways around this and thus for safety-critical systems this should not be taken as an absolute. Network-probing tools such as *Nmap* have been known to penetrate the inherent SNAT security measure.
[5] Firewalls are often routers with the ability to block/filter network communications.

```
$       iptables -t nat -A POSTROUTING  -p tcp -o eth0 -j SNAT \
                --to 142.145.32.165:1-65535
$       iptables -t nat -A POSTROUTING  -p udp -o eth0 -j SNAT \
                 --to 142.145.32.165:1-65535
```

These rules, when read together, are instructing the *Iptables* mechanism to load the necessary modules (if not already done) into the kernel space, convert all TCP and UDP-based packets coming in from interfaces *eth1* and *eth2*, and assign them an IP address of 142.145.32.165, the same as the Linux router (interface *eth0*). Furthermore, these rules allow for all possible service ports (ports 1 - 65535) to be made available through the router. Through trial and error, it was found that certain applications would otherwise[6] not work correctly. These rules should be added to the system post-start-up file */etc/rc.local* and the rules for *Frox* (Section 1.7) should be removed.

While SNAT blocks systems from the corporate network from accessing both the software lab and the contractor's networks, nothing is currently stopping the contractor's network from accessing the corporate network. Thus, a set of ACL's will be required. These ACL's can be written to block either specific IP addresses or entire ranges such as a subnet.

To block bi-directional access to/from the contractor's network (it is isolated from the software lab and corporate network) altogether, in addition to the commands in steps 1 and 2, this command suffices [3]:

Step 3:

```
$       include Step 1
$       include Step 2
$       iptables -i INPUT -i eth2 -s 192.168.1.0/255.255.255.0 -j DROP
```

It is important to understand that individual IP addresses within the blocked networked cannot be unblocked. The only method of unblocking is to unblock the entire network; however, this can only be done when the entire network has been blocked. But, if the systems on the network had been blocked individually, IP address by IP address, then using the following commands, it would be possible to unblock various addresses [3]:

Step 4:

```
$       include Step 1
$       include Step 2
```

[6] It is important to note that under UNIX-based operating systems only the root user can use ports 1-1023. This is why many connections may work for the root user but not work for regular users; regular users must use service ports higher than 1023.

$ iptables -i INPUT -i eth2 -s 192.168.1.21 -j DROP

… And all the IP addresses in between that exist …

$ iptables -i INPUT -i eth2 -s 192.168.1.40 -j DROP

Moreover, if a given blocked IP address requires access to the software lab (or the corporate network), then to unblock that address, the following command can be used [3]:

Step 5:

$ iptables -D INPUT -i eth2 -s 192.168.1.35 -j DROP

Or any other valid IP address that should be unblocked

Unfortunately, the method of blocking the contractor's network from the corporate network while at the same time allowing it full access to the software lab could not be determined [at this time]. While it is believed that a method for doing exists, none could be found in the various sources of documentation. Since the method could not be determined it cannot be listed here.

This causes the unfortunate inconvenience of an all or nothing approach. Either the contractor's lab is blocked altogether or it is left open so that it can connect to both of the other networks (software lab and corporate network). A reasonable work around is to individually block only the IP addresses of the system's on the contractor's network that should not have access to either of these two networks and not block the addresses of the systems that require legitimate access to either of the two other networks. However, whether this workaround can be permitted will be up to local network/IT security support staff.

The advantage is that it is relatively simple to administrate the systems on the contractor's network that do and do not have access to the software lab and corporate network. The disadvantage is that if a system should only have access to the software lab it will unfortunately also have access to the corporate network.

The problem arises from the complexities inherent in SNAT. This is due to the conversion of the IP addresses (from the contractor's network/software lab) to that of *eth0*'s through NAT. Therefore, it is best if the number of system's on the contractor's network that will be granted access to both networks is limited to the fewest number possible to avoid any unnecessary security issues.

2. Conclusion

Through various steps and procedures it has been successfully demonstrated how to build a simple Linux router, and from there, configure it for added capability. The topics of web-based proxy caching, FTP proxy caching, NAT, NAT redirection, as well as basic and more advanced firewall ACL's have been explored.

Via the brief discussion in annexes A.1 and A.2, various other uses for a Linux-based router have also been explored; as well, the various tools that can be used to both monitor a network managed by a Linux router and collect statistics.

By better understanding how Linux performs routing and how its routing tables can be manipulated, new forms of interconnection and internetworking can be implemented. While only RIP was examined, it is a sufficient routing protocol for the majority of use cases. Unfortunately, the available documentation from the TLDP as well as [1] [2] [3] did not provide any meaningful insight into building a multi-homed router with more than two Ethernet interfaces.

The Linux *Iptables* command provides a powerful and versatile facility for creating firewall rules and implementing NAT. It is a tool that should not be dismissed. While this report has only examined the packet filtering capabilities of *Iptables*, its state-based filtering capabilities should not be dismissed. The type of filtering will depend largely on the requirements. Unfortunately *Iptables* can be difficult to learn and master because of its numerous options and features. While working with *Iptables*, it is best to start simple and apply additional rules and features only as required.

Thus, in conclusion, the original objective has been satisfied. Furthermore, because Linux-based routers are cheap to build, and, with the proper set of instructions, relatively simple to implement, they may be better suited to certain uses than commercial-based routers. The difference in cost may also be a determining factor for those with specific routing requirements and concerns.

References

[1] Red Hat. Red Hat Linux 9: Red Hat Linux x86 Installation Guide. Revision 2003-01-16T18:24-0400. Red Hat Press. Guide. 2003. http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/pdf/rhl-ig-x86-en-9.pdf

[2] Hollingshead, James (2005). INSTALL for Frox. Installation instruction manual. http://www.hollo.org/cgi-bin/darcs?frox*.

[3] Ziegler, Robert. Linux Firewalls Second Edition. ISBN 0-7357-1099-6. New Riders. Book. Nov 2001.

This page intentionally left blank.

Annex A Additional points of consideration

A.1 Various uses of a Linux router

The Linux router could be expanded to support as many physical network segments as there are physical networks connected to it.

Using Linux, it is possible to create a network bridge, a routing bridge (also known as a brouter), a switch, a routing switch, or even a VLAN. However, these capabilities must be activated within the Linux kernel and this generally requires a kernel reconfiguration, recompilation, and reinstallation. However, there are some commercially available versions of Linux that are pre-built to provide these capabilities for immediate use.

The following possibility is deploying a Linux router as a NIDS system. NIDS-based software is generally installed on central routing/internetworking points of the network. These points can be devices such as the corporate routers/firewalls connected to the Internet, or routers that connect subnets, switches, and bridges. NIDS software will work efficiently when it can intercept all of the network traffic on one or more subnet segments, analyze the flow of traffic, and understand the types of data being transmitted. In doing so, it can look at and determine whether any potential breaches or unauthorized access to network resources have occurred. Combined with other open source software such as *Snort* and *Ntop*, it is possible to build a very robust and low-cost NIDS solution. These systems are becoming important in today's large-scale networks and are commonly found in commercial and military networks alike. NIDS systems are uniquely positioned to leverage on their key central location on networks. Some systems come as readily available pre-built network appliances while others build their own, not unlike what has been done here in this document.

Another potential use is in building a high-security firewall. This system could be used to provide a connection to the Internet, or interconnect networks that must be rapidly deployed and made secure, possibly in a military theatre of operation. Using NAT and the Linux *Iptables* firewalling mechanism, it is possible to build a robust, powerful, low-cost, multi-homed router capable of either packet filtering or state-based filtering. For classified networks, SELinux is a possible open source alternative.

A final possible use that is worth taking note of is building an interconnected network with a Linux router at the center. Each attached network could be a subnet, and through the router, they could interconnect and internetwork. If necessary, certain firewall rules could be applied to minimize traffic. Furthermore, using DHCP it is possible to dynamically allocate IP address to all of the various network clients. It is also possible to provide statically based DHCP addresses for those who are more security conscious.

These are only some of the possible uses and niches for a Linux-based router. Many more uses are possible.

A.2 Tools for software monitoring, enhancing security, tracking statistics and performance metrics

The first tool to examine is the open source tool *Ntop*. This tool was originally designed for network monitoring and intrusion detection. Commonly referred to as a lightweight NIDS, it is commonly used for examining both network traffic and dataflows. The tool works by collecting network packets on the various network interfaces of a given system. *Ntop* can then parse, sort, analyze, and categorize the collected data into very meaningful tables that are accessible via its internal web server. Through this web server, it is possible to visualize all of the traffic and dataflows that have gone through any of the system's network interfaces. Furthermore, it can distinguish between multicast or unicast traffic. It can also present statistical information for the various protocols collected and graph it in percentages or as a function of time. Different graphs, tables, charts, and flow diagrams enable the network administrator to better examine and understand the network. It is also possible to break down the protocols according to use and with which systems it communicated with. All these features enable a network administrator to better track those that abuse network bandwidth, performing unauthorized scans or attacks, or accessing network resources that normally should not be accessible.

Squid has the ability to provide various types of performance metrics that can be determined from its web cache. The *Squid* system provides a CGI interface that can be accessed by a web browser to provide various information. Through the interface, it is possible to learn more about how the cache has handled the load as well as provide basic statistics about the network traffic handled by the cache.

Calamaris is another open source tool that can be used to generate powerful information from the *Squid* log files. It parses these log files and generates HTML-based reports as a function of time. Furthermore, it can provide an overview *Squid*'s performance metrics. It can also display the bandwidth information of other services such as web, FTP, and Gopher. The tool, however, is not designed to provide a protocol-by-usage, protocol-by-system, or protocol-by-protocol breakdown. If this is required, then *Ntop* is suggested.

EtherApe, an open source tool, is a near real-time GUI network exploration tool. It can graphically represent traffic intercepted on the network. Its GUI console is easy to understand and use. It can display the various network streams, including the systems involved in the communications. One powerful feature of the tool is that it can replay network streams if they were captured using other tools such as *Tcpdump*. This could be a very useful feature for forensic analysis of captured network traffic. The traffic can be replayed repeatedly. *EtherApe* is well suited to finding and tracking down systems that abuse network bandwidth. Abuse could be caused by an incorrect system configuration, broadcast storm, or an attack. It can also help track down systems that are attempting to access or perform non-authorized network actions.

Ethereal is probably the most powerful protocol analyzer and network packet currently in existence. It is open source and far superior to any commercially available equivalent. It offers support for most packet capture file formats, and unlike many of the commercial tools, is capable of working with extremely large data files. On powerful systems, it can also examine incoming packets in pseudo-real-time. Using *Tcpdump*-based filters, it selectively captures desired network data. It runs both as a GUI and command-line tool and offers extensive filtering capabilities. Furthermore, it can provide various statistics on intercepted traffic. It currently provides support

for 658 protocols that it can analyze and dissect. It is used by network professionals around the world for troubleshooting, analysis, software and protocol development, and education. It has all of the standard features expected in a commercial protocol analyzer. It runs on all popular computing platforms, including UNIX, Linux, and Windows.

The Linux kernel, via its TCP/IP stack, can keep basic network flows. This information can be displayed with the *Iptables* command. Each network interface also has a byte counter that keeps track of every byte crossing a given system's interface. This information can be queried using the *ifconfig* command. Unfortunately, however, the *Iptables* command will reset after a given time period or after a given threshold of packets has been surpassed.

Snort is another powerful and robust open source heavyweight NIDS system. However, it is a very complex piece of software to correctly configure and install. Furthermore, it requires advanced knowledge from the network layout in order to be correctly implemented so that it can be used in an effective manner. It can, and often will complain about network traffic that looks like an attack, a scan or probe, or virus. The documentation is heavy to read through. However, this tool is mentioned here because it is capable of analyzing and graphing network dataflows on a system-by-system basis or by protocol. It is as good at this as *Ntop*; however, its configuration is far more complex.

The aforementioned tools are relatively simple to use, yet are powerful in spite of it.

Annex B System configurations and outputs

B.1 Final Squid configuration file

The following is the entire *Squid* configuration file for the Linux router. The configuration file can be found under */etc/squid/squid.conf*, has been configured for reverse proxy caching and is as follows:

```
cache_mem 64 MB
cache_dir ufs /var/spool/squid 1000 16 256
log_fqdn on
ftp_user bob@drdc-rddc.gc.ca
ftp_list_width 64
ftp_passive off
ftp_sanitycheck off
auth_param basic children 5
auth_param basic realm Squid proxy-caching web server
auth_param basic credentialsttl 2 hours
refresh_pattern ^ftp:           1440    20%     10080
refresh_pattern ^gopher:        1440    0%      1440
refresh_pattern .               0       20%     4320
acl all src 0.0.0.0/0.0.0.0
acl manager proto cache_object
acl localhost src 127.0.0.1/255.255.255.255
acl to_localhost dst 127.0.0.0/8
acl SSL_ports port 443 563
acl Safe_ports port 80          # http
acl Safe_ports port 21          # ftp
acl Safe_ports port 443 563     # https, snews
acl Safe_ports port 70          # gopher
acl Safe_ports port 210         # wais
acl Safe_ports port 1025-65535  # unregistered ports
acl Safe_ports port 280         # http-mgmt
acl Safe_ports port 488         # gss-http
```

```
acl Safe_ports port 591          # filemaker
acl Safe_ports port 777          # multiling http
acl QUERY urlpath_regex cgi-bin \?
acl CONNECT method CONNECT
no_cache deny QUERY
acl our_network src 0.0.0.0/255.0.0.0
http_access allow all
http_access allow our_network
http_access allow localhost
http_access allow all
http_reply_access allow all
icp_access allow all
cache_effective_user squid
cache_effective_group squid
visible_hostname router
http_port 3128
httpd_accel_port 80
httpd_accel_single_host on
httpd_accel_with_proxy on
httpd_accel_uses_host_header on
https_port 3128
httpd_accel_host virtual
```

The following four lines from the configuration file are responsible for reverse proxy caching:

```
httpd_accel_port 80
httpd_accel_single_host on
httpd_accel_with_proxy on
httpd_accel_uses_host_header on
```

B.2 Squid start-up output

To start-up *Squid*, according to the configuration file specified in Annex B.2, the following command is used:

```
$       squid -N -d 1 -f /etc/squid/squid.conf
```

The output from the program is as follows, using said configuration file:

2006/01/25 00:04:53| Starting Squid Cache version 2.5.STABLE1 for i386-redhat-linux-gnu...

2006/01/25 00:04:53| Process ID 5265

2006/01/25 00:04:53| With 1024 file descriptors available

2006/01/25 00:04:53| Performing DNS Tests...

2006/01/25 00:04:53| Successful DNS name lookup tests...

2006/01/25 00:04:53| DNS Socket created at 0.0.0.0, port 32769, FD 5

2006/01/25 00:04:53| Adding nameserver 142.145.32.2 from /etc/resolv.conf

2006/01/25 00:04:53| Unlinkd pipe opened on FD 10

2006/01/25 00:04:53| Swap maxSize 1024000 KB, estimated 78769 objects

2006/01/25 00:04:53| Target number of buckets: 3938

2006/01/25 00:04:53| Using 8192 Store buckets

2006/01/25 00:04:53| Max Mem size: 65536 KB

2006/01/25 00:04:53| Max Swap size: 1024000 KB

2006/01/25 00:04:53| Rebuilding storage in /var/spool/squid (DIRTY)

2006/01/25 00:04:53| Using Least Load store dir selection

2006/01/25 00:04:53| Current Directory is /root

2006/01/25 00:04:53| Loaded Icons.

2006/01/25 00:04:53| Accepting HTTP connections at 0.0.0.0, port 3128, FD 12.

2006/01/25 00:04:53| commBind: Cannot bind socket FD 13 to *:3128: (98) Address already in use

2006/01/25 00:04:53| Accepting ICP messages at 0.0.0.0, port 3130, FD 13.

2006/01/25 00:04:53| WCCP Disabled.

2006/01/25 00:04:53| Ready to serve requests.

2006/01/25 00:04:53| Done reading /var/spool/squid swaplog (328 entries)

2006/01/25 00:04:53| Finished rebuilding storage from disk.

2006/01/25 00:04:53| 327 Entries scanned

2006/01/25 00:04:53| 0 Invalid entries.

2006/01/25 00:04:53| 0 With invalid flags.

2006/01/25 00:04:53| 327 Objects loaded.

2006/01/25 00:04:53| 0 Objects expired.

2006/01/25 00:04:53| 0 Objects cancelled.

2006/01/25 00:04:53| 1 Duplicate URLs purged.

2006/01/25 00:04:53| 0 Swapfile clashes avoided.

2006/01/25 00:04:53| Took 0.3 seconds (1110.9 objects/sec).

2006/01/25 00:04:53| Beginning Validation Procedure

2006/01/25 00:04:53| Completed Validation Procedure

2006/01/25 00:04:53| Validated 326 Entries

2006/01/25 00:04:53| store_swap_size = 6084k

2006/01/25 00:04:54| storeLateRelease: released 0 objects

B.3 Frox configuration file

The *Frox* configuration file */usr/local/etc/frox.conf* for usage on the Linux router is as follows:

Listen 0.0.0.0

Port 2121

ResolvLoadHack wontresolve.doesntexist.abc

User squid

Group squid

WorkingDir /var/spool/frox

LogFile /var/spool/frox/frox-log

PidFile /var/run/frox.pid

BounceDefend yes

MaxForks 10

MaxForksPerHost 4

ACL Allow * - *

Bibliography

Böhme, Uwe. Linux BRIDGE-STP-HOWTO. Revision 0.4. The Linux Documentation Project. Howto. Jan 2001. http://www.tldp.org/HOWTO/text/BRIDGE-STP-HOWTO.

Breuer, Peter. Linux Bridge+Firewall Mini-HOWTO. Revision 1.2.0. The Linux Documentation Project. Howto. Dec 1997. http://www.tldp.org/HOWTO/text/Bridge+Firewall.

Chmielewski, Tomasz. Bandwidth Limiting HOWTO. Revision 0.9. The Linux Documentation Project. Howto. Nov 2001. http://www.tldp.org/HOWTO/text/Bandwidth-Limiting-HOWTO.

Cole, Christopher. Bridging mini-HOWTO. Revision 1.22. The Linux Documentation Project. Howto. May 2002. http://www.tldp.org/HOWTO/text/Bridge.

Dawson, Terry, Alessandro Rubini. Linux Networking-HOWTO. Revision 1.5. The Linux Documentation Project. Howto. Aug 1999. http://www.tldp.org/HOWTO/text/NET3-4-HOWTO.

Edwards, Bob. ProxyARP Subnetting HOWTO. Revision 2.0. The Linux Documentation Project. Howto. Aug 2000. http://www.tldp.org/HOWTO/text/Proxy-ARP-Subnet.

Gonzato, Guido. Configuration HOWTO. Revision 1.99.7. The Linux Documentation Project. Howto. Nov 2001. http://www.tldp.org/HOWTO/text/Config-HOWTO.

Gortmaker, Paul. Linux Ethernet-Howto. Revision 2.9. The Linux Documentation Project. Howto. Aug 2003. http://www.tldp.org/HOWTO/text/Ethernet-HOWTO.

Grennan, Mark. Firewall and Proxy Server HOWTO. Revision 0.8. The Linux Documentation Project. Howto. Feb 2000. http://www.tldp.org/HOWTO/text/Firewall-HOWTO.

Hubert, Bert, et al. Linux Advanced Routing & Traffic Control HOWTO. Revision 1.1. The Linux Documentation Project. Howto. Jul 2002. http://www.tldp.org/HOWTO/text/Adv-Routing-HOWTO.

Kiracofe, Daniel. Transparent Proxy with Linux and Squid mini-HOWTO. Revision 1.15. The Linux Documentation Project. Howto. Aug 2002. http://www.tldp.org/HOWTO/text/TransparentProxy.

McCarthy, Bill. Red Hat Linux Firewalls. ISBN 0-7645-2463-1. Red Hat Press. Book. 2003.

Pearson, Oskar. Squid - A User's Guide. Revision 1.1. Qualica Technologies Ltd. Guide. 2003. http://squid-docs.sourceforge.net/latest/book-full.html.

Pillay, Harish. Setting up IP Aliasing on A Linux Machine Mini-HOWTO. Revision 1.2. The Linux Documentation Project. Howto. Jan 2001. http://www.tldp.org/HOWTO/text/IP-Alias.

Radtke, Nils. Ethernet Bridge + netfilter Howto. Revision 0.8. The Linux Documentation Project. Howto. Jul 2005. http://www.tldp.org/HOWTO/text/Ethernet-Bridge-netfilter-HOWTO.

Ranch, David A. Linux IP Masquerade HOWTO. The Linux Documentation Project. Howto. Nov 2005. http://www.tldp.org/HOWTO/text/IP-Masquerade-HOWTO.

Ridruejo, Daniel Lopez. The Linux Networking Overview HOWTO. Revision 0.32. The Linux Documentation Project. Howto. Jul 2000. http://www.tldp.org/HOWTO/text/Networking-Overview-HOWTO.

Russell, Rusty. Linux 2.4 NAT HOWTO. Revision 1.18. The Linux Documentation Project. Howto. Jan 2002. http://www.netfilter.org/documentation/HOWTO//NAT-HOWTO.txt.

Russell, Rusty. Linux IPCHAINS-HOWTO. Revision 1.0.8. The Linux Documentation Project. Howto. Jul 2000. http://www.tldp.org/HOWTO/text/IPCHAINS-HOWTO.

Wirzenius, Lars, et al. The Linux System Administrator's Guide. Revision 0.9. The Linux Documentation Project. Book. 2004. http://www.tldp.org/LDP/sag/html/sag.html.

List of symbols/abbreviations/acronyms/initialisms

ACL	Access Control List
ARP	Address Resolution Protocol
Brouter	Bridging Router
BSD	Berkeley System Distribution
C2	Command & Control
CD-ROM	Compact Disc Read Only Memory
CNET	Classified Network
CVS	Concurrent Versioning System
DHCP	Dynamic Host Configuration Protocol
DNAT	Destination NAT
DND	Department of National Defence
DNS	Domain Name Service
DOS	Disk Operating System
DRDC	Defence Research Development Canada
DWAN	Defence Wide Area Network
EST5EDT	Eastern Time North America Eastern Daylight Savings Time
FTP	File Transfer Protocol
GB	Gigabyte
Gbps	Gigabits per second
GPL	GNU Public License
GRUB	Grand Unified Boot-loader
HTML	Hypertext Mark-up Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Device Electronics
IEEE	Institute of Electrical and Electronics Engineers, Inc.
IETF	Internet Engineering Task Force
IP	Internet Protocol
IT	Information Technology
LAN	Local Area Network
LILO	Linux Loader
JCDS21	Joint Command Decision Support 21

MAC	Media Access Control
MB	Megabyte
MD5	Message Digest 5
MHz	Megahertz
MS	Microsoft
NAT	Network Address Translation
NIDS	Network Intrusion Detection System
NTLM	[Windows] NT LAN Manager
OSI	International Organization for Standardization (ISO)
OSPF	Open Shortest Path First
PC	Personal Computer
PCI	Peripheral Component Interconnect
PS/2	Personal System/2
RAM	Random Access Memory
R&D	Research & Development
RFC	Request for Comment
RH	Red Hat
RH9	Red Hat 9
RIP	Routing Information Protocol
SCSI	Small Computer System Interface
SELinux	Security Enhanced Linux
SNAT	Source NAT
SNMP	Simple Network Management Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TDP	Technical Demonstration Project
TLDP	The Linux Documentation Project
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VPN	Virtual Private Network

Glossary

ACL (Access Control List)

An Access Control List is a list of specific privileges users and systems can have. These privileges are generally based on system access, file access, network access, or access to various services. ACL's exist in various forms. A firewall is a form of an ACL where the firewall describes which systems have access to which remote system/service/port.

ARP (Address Resolution Protocol)

The Address Resolution Protocol is the TCP/IP protocol responsible for the dynamic mapping between IP addresses and network interface hardware addresses. Generally, a host system wishing to communicate with a destination system will send an ARP request that requests the hardware address (MAC address) of the destination system and returns this information to the host system so that it can continue with its communication. This is a very low-level protocol on the TCP/IP stack and is generally used on the local LAN. For hardware addresses outside of the local LAN, usually the router/gateway's hardware address is used as substitute for the real destination's hardware address. ARP is an Ethernet-specific protocol (802.x networks) and is defined in RFC 896.

Bridge

A bridge is a networking device that connects network segments via the Data Link Layer (in the OSI model); a network switch in most respects acts like a bridge.

Bridging Router

See Brouter.

Broadcast

A broadcast is information sent across the network to all the systems or individuals of the network. Broadcasts are generally used to advertise new systems or services, or query for them.

Broadcast Address

The broadcast address is an address that identifies all of the systems on a given network. It is not the same as the Network Address. More specifically, it is a **multicast** address and it allows IP information to be sent to all of the systems on a given network/subnet. Its standard is found in RFC 919.

Brouter (Bridging Router)

A brouter is a bridge with routing capabilities and it can be used to separate networks according to broadcast traffic. While bridges ordinarily work on the Data Link Layer of the OSI Model, brouters can work on the Data Link and Network Layers of the OSI Model.

BSD (Berkeley System Distribution)

Berkeley Software Distribution is a very popular flavour of UNIX that was first developed at the University of California at Berkeley. BSD is very well known for its robust TCP/IP stack, excellent stability, scalability, and robustness. There are currently three popular open source versions of the BSD operating system, FreeBSD, NetBSD, and OpenBSD.

CVS (Concurrent Versioning System)

The Concurrent Versioning system is an open source tool designed for version control of source code and is a highly exploited development collaboration system.

Data Link Layer

This is the second layer of the OSI Model that defines the protocol-independent method for data packet creation and transmission for the lower OSI layers. The data is encoded and decoded into bits and performs transmission error checking. It is at this layer that network switches work.

DHCP (Dynamic Host Configuration Protocol)

The Dynamic Host Configuration Protocol is a protocol used for dynamically assigning a requesting client system all of the TCP/IP and networking configuration parameters it requires for functioning correctly on the network, such as IP address, gateway, netmask, DNS information, etc. It is defined in RFC 2131.

DNAT (Destination NAT)

DNAT is used when it is appropriate for converting IP addresses from a router's external network interface to a single IP address for use on the internally connected networks/interfaces such that it appears as if the remote system(s) connected to the external interface were on the same network segment as the internal network(s). However, DNAT blocks the router's internal systems from accessing the external network yet it allows the external network to access the internal networks and their resources. DNAT is the opposite of SNAT.

DNS (Domain Name Service)

The Domain Name System is a protocol used to communicate between clients requesting lookup information for either a machine name or an IP address. The DNS Server, through a lookup in a flat-file database (or via queries to other Internet DNS servers) converts machine names to their respective IP addresses and returns this information to the requesting client. It could also be that the client wishes to get the machine name associated to a specific IP address. It is defined in multiple RFC's.

DOS (Disk Operating System)

A very common operating system found in the 1980's and 90's. It was the operating system of choice on PC's until 32-bit versions of Windows came to market. While it is outdated and has not been improved upon since the mid to late 1990's, it is nevertheless an operating system of choice for those performing troubleshooting and diagnostic related work on a PC.

Ethernet

A local area networking technology that is the most widely used and is simple and effective, and today can reach speeds up to more than 1 Gbps (gigabits/second). This is defined by IEEE (Institute of Electrical and Electronics Engineers) standard 802.3. It is a networking technology and not a networking protocol. Any suite of protocols can run on Ethernet.

Firewall

A firewall may be either a hardware and/or software solution that through a series of rules or security policies enforces the general network behaviour for network traffic passing through either side of the networks connected to it. Firewalls are often found in conjunction with NAT-based systems or are themselves NAT-enabled. Firewalls are generally used when one wants to connect an internal network to one or more untrusted external networks such as the Internet.

Frox

Frox is an open source FTP caching system, not unlike Squid; however, it has fewer capabilities than Squid. Frox is a suggested replacement for Squid's FTP caching feature when Squid is unable to complete FTP connections due to interfering firewalls.

FTP (File Transfer Protocol)

The File transfer Protocol is one of the oldest TCP/IP protocols that are used to transfer files between remote systems. FTP access can be granted via anonymous or authenticated login. FTP has been around since before the World Wide Web was created. FTP is defined by RFC 959.

Gateway

A gateway is generally an entry point into another network. The address denoted by the gateway often means the system is a router or is routing-capable. Gateways can often provide services such as NAT, firewalling, or translation (i.e. PC-to-mainframe gateway; ASCII to EBCDIC). Gateways generally work on layers 4 through 7 of the OSI Model.

Gopher

Gopher is an Internet-based document browsing and searching system that allows for the browsing, querying, and retrieval of text from the Internet. It is defined in RFC 1436.

GPL (GNU Public License)

The GPL is a public license that was created by the FSF (Free Software Foundation). This license provides the end-user with the right to freedom in choice and diversity of software. Furthermore, this freedom is provided by having access to the software's source code. GPL-licensed software does not have to be free however, and developers can charge end-users for it. Those that develop with GPL-licensed software are, by the terms of the agreement, required to distribute any changes to the original source code and publish it using the GPL license.

GRUB (Grand Unified Boot-loader)

A more advanced boot loader than LILO; it is the official boot loader of the GNU/Linux operating system.

Host IP address

The Host IP address is the actual IP address denoted to a physical network host that can be any type of network-enabled device. Valid addresses range in the form of A.B.C.x where A.B.C denote a Class A, B, C, or subnetted network and x can be any number between 1 and 254. The address 0 is reserved for the Network Address and 255 are reserved for the Broadcast Address.

How-to

A How-to document is a descriptive text for how to accomplish a given task/series of tasks. It is very commonly found in the UNIX and Linux world (at TLDP). These documents can range in length from short to very long, sometimes long enough to be a short book. These documents are meant to impart knowledge and wisdom on a given subject to other non-experts. As How-to suggests, these documents are often like a recipe; follow the instructions and understand the basic concepts and out comes a working result.

HTTP (Hypertext Transfer Protocol)

The Hypertext Transfer Protocol is a set of established rules that most web browsers support to varying degrees that allows for the exchange of files and information (i.e. text, video, sound, multimedia, graphics, etc.) across the Internet and other TCP/IP-based networks. It is also the standard protocol used for transmitting web pages. HTTP 1.1 is defined by RFC 2616.

Ifconfig

A UNIX-based command used to directly modify the system's network interfaces and network-based properties.

IP (Internet Protocol)

See TCP/IP.

IP Address

An IP address is a computer-based address for communicating via the TCP/IP protocol stack. This address consists of four numbers separated by dots (.). Each number can range from 1 – 255. These addresses enable a computer-based system to connect and interoperate with other systems on a computer network, including but not limited to the Internet.

Ipchains

Very similar to Iptables it is used to provide Linux systems with firewall capabilities, and includes the ability to provide NAT and other such services. It is a packet-filtering firewall and cannot perform any state-based inspections on traffic. It is been replaced in more modern kernel distributions of Linux by Iptables. It was found with kernels 2.2 and earlier.

Iptables

It is a robust, high configurable, high-performance IP packet filtering and stateful inspection firewall toolkit built-in to the Linux kernel and has replaced Ipchains. Iptables is found with Linux kernel 2.4 and higher and has become the de facto firewalling standard for Linux platforms. Iptables is both a packet filtering and state filtering firewall tool.

IP Masquerading

See Masquerading.

LAN (Local Area Network)

A LAN is a conglomeration of computers and other network-enabled devices which that use a standard media (i.e. Ethernet) as a means of intercommunicating and interoperating with one another. Furthermore, a LAN does not have to be made up of systems of the same type, use, or operating system. If a system is connected to a local network but is unable to communicate with other systems because it is not configured to do so, then it normally is not considered as a part of the network even if it is attached to it.

LILO (Linux Loader)

A Linux boot loader.

Linux

Linux is a free and open source software operating system based on UNIX. It was originally conceived and developed (the kernel) by Linus Torvalds, who holds and maintains the copyright to Linux. With the assistance of thousands of developers around the world, it has turned into a robust, stable, and secure operating system, not unlike its other UNIX-based counterparts. Other than the kernel, it is mainly composed of developer-contributed software, much from the FSF (Free Software Foundation).

MAC (Machine Access Control) Address

The Media Access Control Address is a 48-bit (for Ethernet) hardware address that is used to designate the physical hardware address of a given network interface. Most vendors have their own specific address ranges (so that they do not overlap with other vendors') and these different ranges can be used to denote the different types, class, and make of various network interfaces.

Masquerading/IP Masquerading

Similar to NAT, it differs in that IP Masquerading/Masquerading provides dynamically assigned addresses, where as NAT simply relies on statically defined addresses.

MS Exchange

It is a commonly used corporate mail server that has replaced many other corporate e-mail systems. It can be used in collaborative efforts and fits into Microsoft's line of commercial server-based products.

Multicast

Information sent across the network used by a single system or host to a series of devices or clients; it is similar to a broadcast but is not as far reaching.

Multi-homed

A PC configured as a router/firewall/gateway with two or more network interface cards. Generally, most multi-homed systems have only two-interfaces. Three or more interfaces are rarer because they require a more advanced knowledge of TCP/IP, routing, a thorough understanding of the underlying operating system.

NAT (Network Address Translation)

Also known as Network Address Translation, it is a techniques used by routers and firewalls to rewrite the source or destination address as network packets pass through its network interfaces. This technique is used to help conceal the identity of the system sending the packet outside of the local network, often done in conjunction with a link to the Internet, or some other external network. It is defined in RFC 2663. Also, see Masquerading.

Netmask

A 32-bit address similar to the IP address that denotes the most local part of the network; it is a way of denoting which network a given machine belongs to. A machine that does not have the same netmask as other systems on a LAN is not considered as a part of that network.

Network Address

A network address is the IP address that addresses the entire local network or subnet. It is not the same as a "host IP address" nor is the same to a broadcast address. Broadcast addresses refer to the address upon which information can be sent to all systems on a given network or subnet, while the network address logically groups these systems into a collective grouping.

Network Layer

This is the third layer in the OSI Model. It is responsible for translating and converting logical and physical addresses from one to the other. It is at this layer that routing takes place.

NIDS (Network Intrusion Detection System)

Network Intrusion Detection System is generally a software-based service running on a system (but can also be found on network appliances) that attempts to find, alert, and stop a user or system from performing suspicious network activity such as attacks, port scans, distributing virus and Trojan horses, and other unwanted forms of network traffic.

NTLM ([Windows] NT LAN Manager)

NTLM is a Microsoft communication security protocol that is used as an authentication mechanism designed specifically for the Windows NT 4.0 operating system. It uses a three-message challenge-response mechanism that is password-less; that is to say that a client

system via the challenge-response can prove its identity to the server. It is also supported under Windows 2000. It can be used in HTTP and mail-based communications.

OSI Model

The International Organization for Standardization (ISO) Model for networking establishes a seven-layer model that can serve as a framework for the interoperability of various telecommunication protocols for computer-to-computer internetworking. Most modern networking protocol stacks (i.e. TCP/IP) are based on the OSI Model, even if these protocol stacks do not necessarily accommodate for the same number of layers. In the OSI Model, every layer is responsible for one or more specific tasks to bring data from the network to the user and vice versa. The layers are Physical, Data Link, Network, Transport, Session, Presentation, and Application.

OSPF (Open Shortest Path First)

Open Shortest Path First is a dynamic routing protocol that is used mainly used by large-scale routing devices. It is a link-state algorithm, often known as a shortest path algorithm. It is found very commonly in today's larger routers and has replaced RIPv1 and RIPv2 as the preferred routing protocol. It requires less router-to-router communication than RIP and is more efficient in its management of its routing tables. It incorporates many subroutines into its algorithm that take into account shortest path, routing tables, transmission speed, bandwidth, and many others. It is defined in RCC 2328.

Proxy ARP

Proxy ARP is a method in which a given network system responds to various ARP requests for another given system or network. This is usually done by a router/gateway. By responding to the ARP requests, the router/gateway then assumes responsibility for routing the packets to that other system/network for which the requests were originally destined. This is usually done as a substitute for subnetting, routing, or NAT. Proxy ARP usually requires the manual definition of ARP tables on the router/gateway. Furthermore, this is usually done for systems that are connected to one of the router/gateway's hidden interfaces from where the ARP requests originate but cannot travel to the destination without aid. It is defined in RFC 1983.

Proxy Server

This is a type of server that caches Internet-based requests such as HTTP, FTP, and DNS requests. Proxy servers generally find themselves on routers, gateways, firewalls, or other devices connected to or near an Internet connection.

RFC (Request for Comment)

Managed by the Internet Engineering Task Force (IETF), it is a process used to propose new standards and make requests for the Internet, which inevitably affects the various implementations of TCP/IP. RFC's can be found on the Internet at http://www.faqs.org/rfcs/.

RIP (Routing Information Protocol)

The Routing Information Protocol is a static routing protocol. It is a very simple routing protocol that takes into account its own routing tables and those of other routers around it, including the number of hops between itself and a given destination. It consumes far more bandwidth than OSPF because it is frequently re-querying other routers for updated information. RIP is a distance-vector protocol and is not well-suited for routing on large networks or for the modern Internet. RIPv1 is defined in RFC 1058 and RIPv2 is defined in RFC 2453.

Route

(1) A system administration tool used on most modern computing platforms used to modify static routes and tables. (2) A route can also be a path that a network packet will travel to from a given source address to a specific destination address.

Router

An internetworking device designed to take network packets from one network and route it on its way to the destination router. A router may perform various actions on the network packets and can pass it forward, drop it, block it, or query another router for the next best path. Modern routers tend to come in the form of a pre-built appliance especially designed for the task; however, PC's can also be used for this task. Routers generally work on the third layer (the Network Layer) of the OSI model.

SNAT (Source NAT)

SNAT is used when it is appropriate for converting IP addresses from within a router's internally connected networks in order to access various remote resources on the router's external interface as a single unique IP address. SNAT is used when either the internal IP addresses are not using appropriately Internet numbers for accessing the Internet or when remote resources will not accept foreign IP addresses. SNAT blocks systems external to the router's internal systems from accessing the internal network(s). SNAT is the opposite of DNAT.

SNMP (Simple Network Management Protocol)

The Simple Network Management Protocol is a TCP/IP protocol used for monitoring, controlling, and collecting information such as statistics, security, performance, and configurations from SNMP-enabled networks and host systems. It is a based on a client/server architecture where the host (the client) has an agent running which provides information to SNMP-based queries. It is defined in RFC 1089.

SSL (Secure Sockets Layer)

The Secure Sockets Layer is a TCP/IP protocol developed by Netscape Inc. and RSA Data Security Inc. for use in transmitting information securely over a public network. It uses key-based encryption to ensure that the network is secure and prevent unauthorized systems from listening in. It is a low-level protocol, lower than the standard web-based protocols and is commonly found in e-commerce sites where personal information is exchanged. It is defined in multiple RFC's.

Squid

Squid is a high performance, open source web caching system that caches web-based requests. It can act as a standard proxy, a web accelerating proxy (reverse proxy), and a transparent proxy.

Subnet

A subnet is a logical subdivision of a larger LAN. This is to say that a given network is broken down into smaller, more manageable chunks. Each subnet is considered to be independent of the other subnets and information between them must be routed to and from in order for information to move across the various subnets.

Subnet Mask

This is another 32-bit number that is similar to the netmask; it could be said that the standard netmask is in fact a special case of a subnet mask for Class A, B, and C networks without the use of subnetting.

Subnetting

This is the act of splitting a network into one or more subnets by creating logical divisions where the various subnets are able to communicate with one another via a router.

Switch

Often referred to as an intelligent hub, the switch has replaced the hub, making hubs obsolete for large networks. Switches are far more efficient than hubs, and use network-bridging logic to interconnect various systems and networks. There exist various forms of switches such as switch-routers and VLAN-capable switches. The switch works on the MAC address of network packets (the Data Link Layer of the OSI Model). Layer 3 switches perform routing (they work on the Network Layer of the OSI Model). Switches are used to build networks in star-fashion (as done with hubs) and build heterogeneous networks by interconnecting them.

TCP (Transmission Control Protocol)

See TCP/IP. Also can be a TCP packet. A TCP packet is a datagram that is encapsulated inside of an IP packet, and then inside of an Ethernet packet (if Ethernet is the network media being used). TCP differs from UDP in that is a connection-based protocol and makes reasonable efforts to ascertain that the data was correctly received by the destination and if not will retransmit. TCP is used with other TCP/IP protocols that require some form of transmission control.

TCP/IP (Transmission Control Protocol/Internet Protocol)

The Transmission Control Protocol/Internet Protocol is a suit of protocols developed by DARPA to facilitate and enable communications between different hosts and different networks. Internet Protocol (IP) is a connectionless protocol and makes no guarantees about the reliability of the transmission of data. UDP is also connectionless and is a part of IP. Network packets are routable because IP is routable. IP contains the necessary information within its header to successfully route data. The Transmission Control Protocol (TCP) is a connection-based protocol that provides multiplexing and is more reliable than IP because

TCP makes reasonable efforts to ensure that packets are received by the destination without corruption. It is the modern standard for networking and is found in most modern networks, as well as the Internet.

TLDP (The Linux Documentation Project)

The Linux Documentation Project is a volunteer-based project whose objective is to maintain the large amounts of written documentation about Linux and related documentation. All of the documentation is made available online and is published under the GPL. Documents range in complexity from the novice to the very advanced. One of the problems facing the project is that Linux has evolved very rapidly, and many of the more recent documents are not capable of taking into account all of the possible uses and configurations of the various Linux distributions out there. TLDP should be considered as a starting point for many things Linux related; however, there is no guarantee that all of the information sought after will be found at TLDP. TLDP is well known for its many Linux guides, tutorials, FAQ's, and How-to's.

UDP (User Datagram Protocol)

A UDP packet is a datagram that is encapsulated inside of an IP packet, and then inside of an Ethernet packet (if Ethernet is the network media being used). UDP differs from TCP in that is a connectionless protocol and makes no efforts to ascertain whether the data was correctly received by the destination. It is very commonly used in TCP/IP protocols that do not require forms of transmission control.

Unicast

Unlike a multicast, a unicast sends data streams to only one destination rather than a grouping or series of destinations. Unicast could be considered as an isolated point-to-point broadcast to between two given systems.

UNIX

A multi-user, multi-tasking, multi-threaded operating system based on a kernel that provides a consistent interface to the user for both interactive and background job processing. UNIX is multi-platform and hardware independent, and supports advanced API's. It is generally considered by the computing industry as the hallmark of scalable, robust, secure, and reliable computing. It was originally developed at AT&T Labs by Dennis Ritchie and Ken Thompson.

VLAN (Virtual Local Area Network)

Also known as a Virtual LAN (Local Area Network), they are used to reduce broadcast traffic and help to secure networks. The goal is to reduce the collision domain of the broadcast domain, and several VLAN's can generally coexist on one switch. VLAN's are often used to help separate and secure various network segments. A router is needed to establish communications between different VLAN's. VLAN's work on the Data Link Layer of the OSI Model.

This page intentionally left blank.

Distribution list

Document No.: DRDC Valcartier TN 2006-630

LIST PART 1: Internal Distribution by Centre:

3 Document Library
1 Richard Carbone (author)
1 Yves van Chestein
1 Guy Turcotte
1 LCdr Elizabeth Woodliffe
1 Dany Dessureault

8 TOTAL LIST PART 1

LIST PART 2: External Distribution by DRDKIM

DRDC Toronto

1 Pierre Michaud
1 Wenbi Wang
1 Kevin Trinh

DRDC Corporate HQ

1 Donna Wood
1 Directorate R & D – Knowledge and Information Management (PDF FILE)

4 TOTAL LIST PART 2

12 TOTAL COPIES REQUIRED

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

DRDC Valcartier

2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.)

Unclassified

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C, R or U) in parentheses after the title.)

(U) Deploying open source routers: A low-cost return on investment

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

Carbone, R.

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)	6b. NO. OF REFS (Total cited in document.)
November 2006	44	3

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Note

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

JCDS21

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
15AT	

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC Valcartier TN 2006-630	

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

Network routing can be a difficult concept to grasp as can the configuration and deployment of routers. As with any specialized equipment, time and understanding are required. Specialized equipment also tends to be expensive. Under the auspice of the Urban Operations project, due to budgetary constraints, a custom Linux-based router was built for interconnecting multiple networks together. While routing with Linux is not any more difficult than with any other operating system, the problem was establishing how to implement and deploy it with three or more interconnected networks. All of the existing documentation was written for two networks and dial-up connections. Through trial and error, one method among many other possibilities was discovered. This method is described in this document and focuses on the inherent capabilities of the Linux operating system. In sharing this work with others it becomes possible for them to expand on it and make better use of it. Linux is a powerful and flexible operating system that is ready for C2 application and this document demonstrates different ways in which it can be deployment for military purposes.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

open source, Linux, router, firewall, gateway, routing, routing table, RIP, OSPF, Iptables, Ip-chains

Defence R&D Canada

Canada's Leader in Defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

WWW.drdc-rddc.gc.ca