Global Information Assurance Certification Paper

Protecting the Intranet,
Buffering Outbound Direct Networked Connections using Citrix

Kevin Noble
January 13, 2003
GSEC Version 1.4b Option 1

Abstract

Corporations are at risk connecting to customers via the Internet or any other hostile TCP/IP connection. Pooling users to a terminal server or a Citrix Metaframe server before a connection can be initiated adds a layer of security that can serve as a chokepoint for all traffic, including VPN connections. Leveraging both a Terminal type connection with a VPN connection gives a 'functional' connection without a 'direct' connection. This type of solution can be seen as a 'buffered network connection'.

Contents

Introduction

Using an indirect connection is derived from a strict corporate policy against outbound modem, VPN, and any other type of direct connection to customers and partners. The need to connect to customers is business critical in today's world, and using standard terminal connections is one way to get connected without having traffic traverse the all-important Intranet, and thus breaking the policy. In some cases customers don't have a host machine or are unwilling to host anything other then a VPN connection. The differential technologies forces the creation of a connection meeting a restrictive policy and a set of technical boundaries.

Imagine for a moment a policy that sets strict boundaries such as the ones listed below:

- No direct modem connection
- No direct VPN connection
- No distribution of the Customer VPN password for general access

Some of the technical boundaries for connectivity include:

- Customers might only allow modem connections
- Customers might only host a Terminal connection
- Customers might only host a VPN connection
- The VPN connection might only allow IPSEC
- The VPN connection might only allow PPTP
- Different customer networks may have the same private IP addresses

These extreme technical boundaries, coupled with the policy, certainly will not be the case for most companies but present an interesting challenge. While this paper tackles the technical boundaries, the first step is to understand what your organizational security policy and technical boundaries are. With the increased need to connect to customers, suppliers, and business partners, the VPN protocol has become the most popular means to allow users to act as part of a remote network. Connectivity growth is on the rise because it has become more important then ever to manage user connections and to minimize 'rogue' connections, where employees create a point of presence (POP) without any authority. Many companies are facing problems with improperly configured wireless networks and large companies do not often find the balance between the need to network to customers and a high degree of security. If the internal company network is to be guarded from attacks and unsolicited traffic, then a buffered network connection can add a layer of security that is affordable. A buffered connection is a bit hard for people to understand; most think in terms of either being connected or not thus read the next line carefully. A buffered network connection leverages an application layer solution, like Terminal Services, with standard layer-2 type connections, such as VPN connections, to stop actual customer traffic from traversing critical internal networks.

Deployment Considerations

Knowing what your customers and business partners require will be the first step. In most cases, one will only need a DMZ hosting a terminal server for outbound connections, and/or a VPN-appliance of some sort. The terminal server alone can give one the presence on the customer network with little effort by simply bridging terminal-to-terminal connections. While a number of terminal client server solutions exist, this paper uses Citrix Metaframe. Citrix allows the client to map drives where data can be passed from client to server. Other Terminal Services solutions can be used instead or in addition, for example Windows

Terminal Services or X-windows for Linux/Unix environments. The emphasis should be to have an application layer interface like Citrix server as a gateway to the DMZ that will in turn allow a direct connection.

Linux's iptables allows a single (guest) machine to connect to multiple customers who may have the same IP address and translate it without violating RFC-1918. While this would be rare in most cases, it needs to be considered when scaling a buffered network solution. Additionally, the Linux advanced routing will need the Point-To-Point Tunneling Protocol (PPTP) client software with Microsoft Point-to-Point Encryption Protocol (MPPE) support and possibly Layer 2 Tunnel Protocol (L2TP).

Consider for a moment how many accounts a single customer gives you to access the Customer VPN server, and then consider how many potential users in your company might need to create a VPN tunnel to that customer. Often only a single account is provided and many people need to connect. Most of the time this one to many model eliminates accountability and increases risk. Some of the connection models presented will have each user logon without exposing the password to all the users.

The Intranet buffering depends on the connection requirements or business requirements matched against the Security Policy, along with the customer technology restrictions. All three factors will help in determining if any of the connection models meet one's needs.

Simple Connection Model

Traditionally, network connections directly connect the guest network linking the host network by either encapsulating the traffic or sending it directly. The packet might be delivered encrypted and undergo a state-full inspection by a firewall. Figure 1 shows the typical connectivity, with the Terminal Server located in the DMZ. A typical user would initiate a session to the terminal server and then make connections to the Internet through the terminal server. This acts as a simple buffered outbound indirect connection, as none of the packets from Customer A or Customer B actually traverses the Corporate LAN because they are terminated at the Terminal Server in the DMZ. Keeping those packets out meets the security policy of 'no direct modem or VPN connection'. One may access the packets through a direct connection to the Terminal Server and then create VPN session to customers, not before. In essence, you control a machine that has limited Internet exposure and it in turn participates on the customer network. The Citrix Metaframe product located on a server in the DMZ becomes the buffer or proxy for user to customer connections.

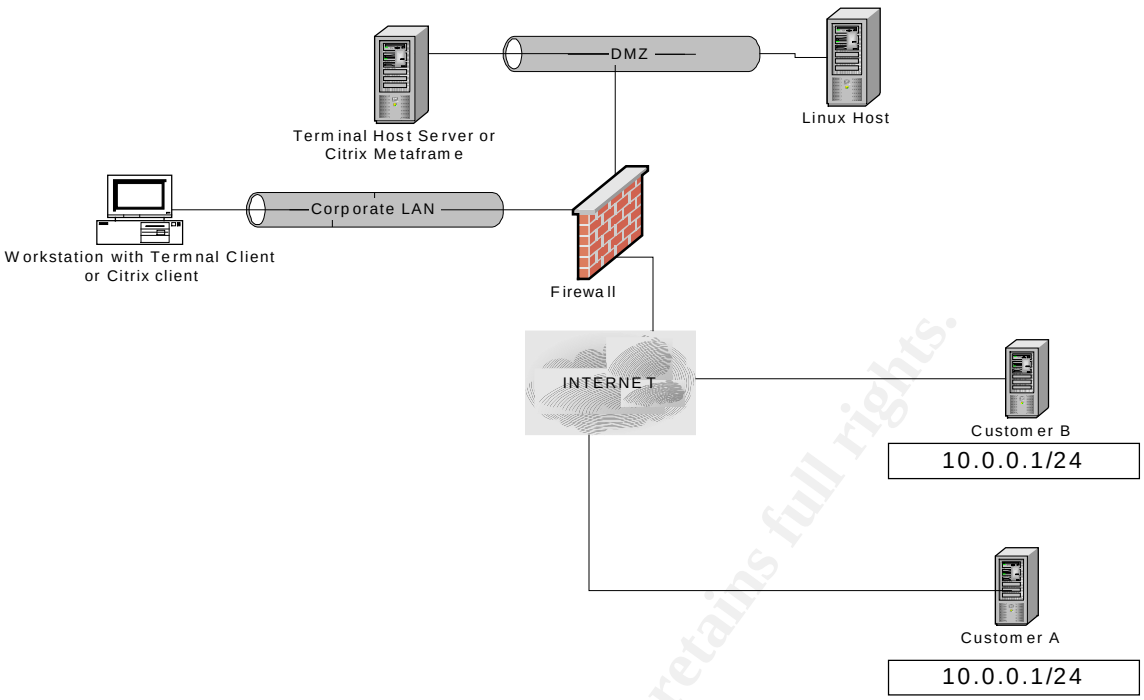

Figure 1

The firewall and the terminal host system should only allow sessions to be created from the authorized corporate LAN networks. An interesting side effect in this model is that the users of the Citrix session share multiple sessions on the terminal host, where users can examine the same applications or share the same data.

The terminal server host can also act as a VPN (L2TP) client connecting to customer sites. This model has some limitations, such as only supporting a limited number of simultaneous connections. Performance also can become an issue when more than a few active connections are created, expending considerable resources on the terminal host server.

It was decided early on to allow only Citrix ICA TCP port 1490 traffic to pass the firewall from Corporate to DMZ. This adds great comfort from a security standpoint. However, as the network becomes more secure, it also becomes more restrictive, requiring more administrative effort. Some consideration was given to using the same accounts that are used in the corporate domain by mirroring or replication from the corporate domain into the DMZ Citrix server, but the idea violates the DMZ solid barrier protecting the corporate Intranet and was quickly dismissed.

Some Variation on the Simple Connection Model

Putting a Citrix or Terminal server in a DMZ to initiate VPN tunnels to customer's networks allows a user to be present on the customer network. In a simple variation, a corporation could have a physical room with a number of workstations that are actually on the DMZ and not the corporate network. We will call this the 'VPN tunnel room' where users would have to simply enter the room and to know only the customer information and control a single machine, which will have a presence on the customer network. In terms of the corporate policy outlined in the introduction section, this solution meets all the criteria with a physical segregation of space and logical networks. It has an added advantage of using a common space where users can watch can keep each other in line. The disadvantage (or advantage depending on one's perspective) is the physical space requires the user to be in the room to perform and use the connections. In enterprises where users are spread out across the country or the world, it would require travel to the single site to initiate access or a 'VPN tunnel room' at every location. Using Citrix or Terminal Server extends the 'VPN tunnel room' to the desktop where users control a session that in turn connects to the customer. The tunnel will always end in the DMZ in some form or fashion to meet our Security policy.

Setting up multiple DMZ with the Citrix server at various points of presence throughout the corporate Intranet is another variation on the simple connection model as shown in Figure 2. This variation can be used to add redundancy or separate customer's VPN connections based on region, authority or need.

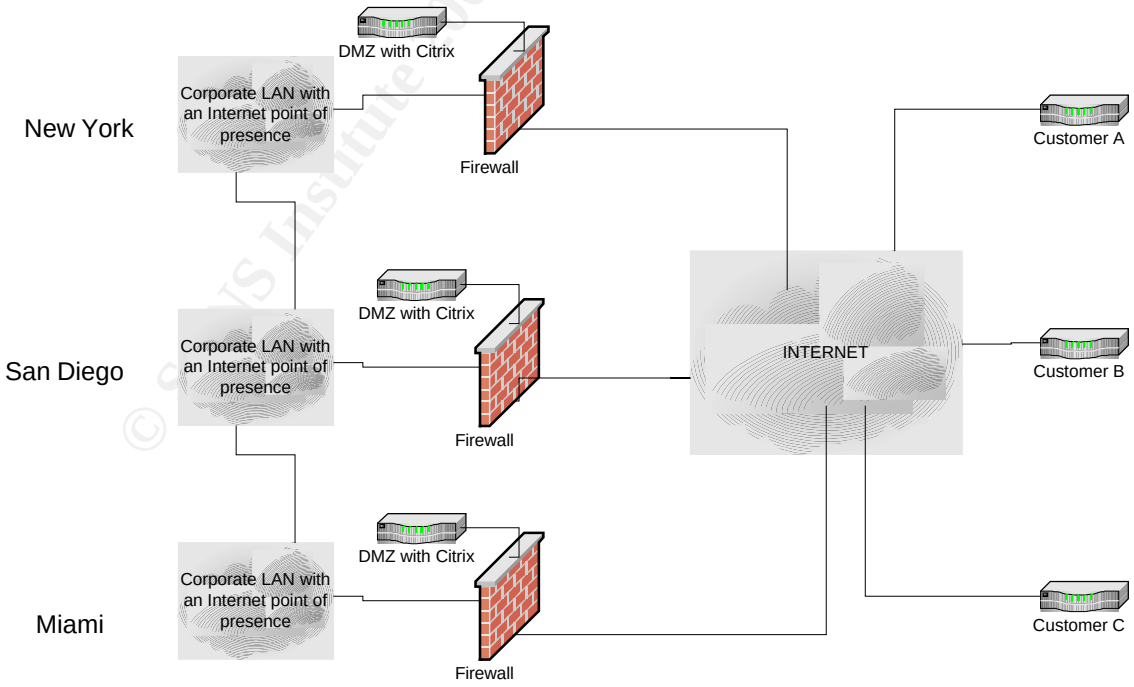

Figure 2

When only a limited number of connections are needed to interface to customers, a DMZ network with a properly configured Citrix Metaframe Terminal Server will be enough for most small to medium sized companies. For a larger number of customers or multiple concurrent VPN connection, adding additional systems loaded with Citrix Metaframe to the DMZ would solve the problem of shared connections.

Additionally, the demand for a separation of VPN sessions for users can be met using Virtual Machines. VMWare allows a host machine to have any number of guest operating systems. As explained by Human X in an article for linuxdig.com "VMWare is an emulator that will allow you to install multiple operating systems and run them all concurrently on top of any Microsoft / Linux operating system. For the purpose of this discussion, the initial operating system installed on your system will be considered the host and the installed VMWare operating systems will be considered slaves / virtual machines" (Human X, 1).

Each guest machine can in fact run Linux, Windows, or other operating systems with the capability to initiate VPN sessions as shown in Figure 3. VMWare initiates a bridged connection to a single networking card allowing multiple instances of an operating systems to appear independent of each other, having a different IP address and a semi-random MAC address.

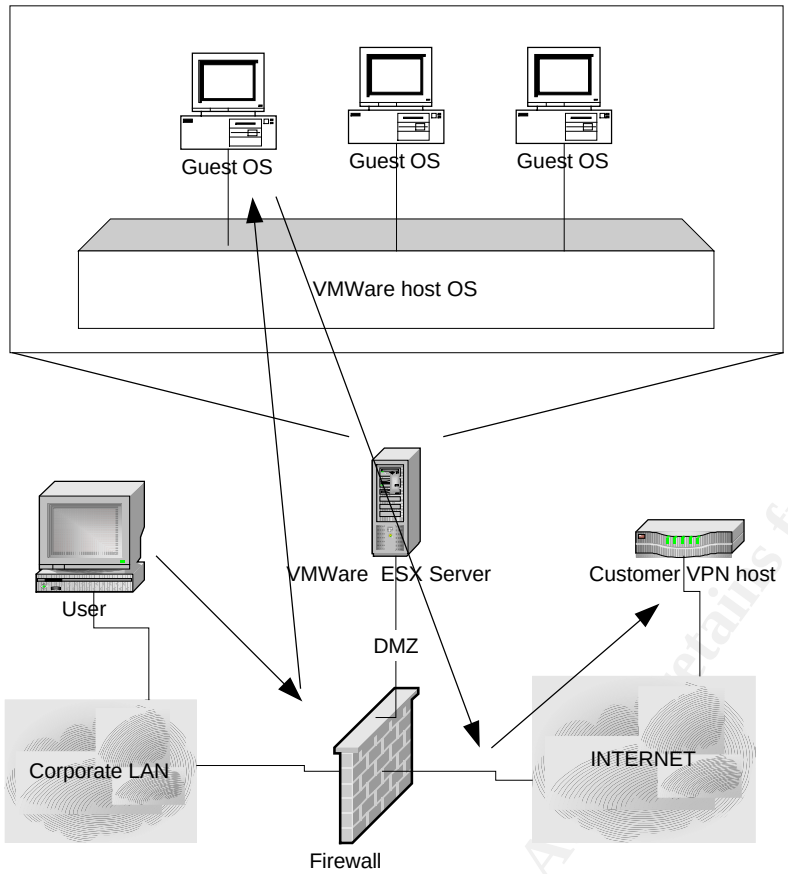

Figure 3

VMWare offers a family of products for Virtual Machine solutions; one of interest is the Vmware ESX Server capable of hosting many guest OS and provides remotely controlling of each guest OS that in turn could participate in customer networks privately as opposed to the Citrix or Terminal Server Solution. The VMWare ESX server will keep the Tunnel endpoint in the DMZ as expected, but it might be more difficult to monitor and manage the connections. VMWare also has single advantages of being able to use exact replica or imaged Guest operating system making it available on the fly. In cases of forensics, a Guest OS can quickly be suspended and moved to another physical machine for investigation. Of all the solutions presented in this paper I would have preferred to use VMWare ESX server, with the exception that customer account information is shared among Guest Operating Systems. This exception causes this solution to fail our security policy, but well worth looking into if sharing the customer logon information is acceptable.

For large companies with many customers, a collection of multiple Citrix Metaframe servers working in concert can be expensive with software licensing and hardware requirements. Even the VMWare ESX Server might be cost prohibitive in terms of software and hardware, forcing one to look for another

solution. Alternately, you might want to consider a routing gateway solution presented in the Advanced Connection Model.

Advanced Connection Model

The Advanced Connection Model meets a special condition to share VPN connection from a single connection. In a production environment, we successfully have a single Citrix Metaframe actively contolling mutiple VPN connection to customers. Meeting the unusual technical problem of different customers having the same private networks was explored by Brian Enright who worked out a method, which, in his own words, "Uses [sic] concurrent connections to multiple VPN hosts that have the same private network IP addresses; which is currently not supported by any terminal service [by itself]." (Enright, *RDC Presentation,* slide 2). Using Network Address Translation (NAT), which allows for connections to identical private Intranets, and by the Linux advanced routing solution, described below, creates this model. "This routing solution can allow for virtually any type of concurrent connectivity to different VPN hosts who might have the same private networks." (Enright, *RDC Presentation,* slide 2)

Linux allows for some very unique translation that deserves an explanation. Figure 4 (Enright, *RDC Presentation,* slide 3) shows the simplistic packet flow through Linux, essentially turning our Linux host into a VPN gateway.

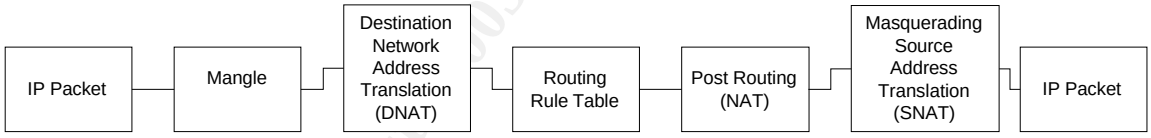

Figure 4 (Enright, RDC Presentation, *slide 3)*

The packet received from the terminal host is mangled; essentially 'marked' in the header field usually used for filtering, but in this case one is using it to mark the traffic for the intended customer. Next, the packet is given the "real" destination address, based on a lookup for the mangled (or "fake") IP in the NAT table; this is always done prior to routing and called Destination Network Address Translation (DNAT). From the mangled IP, the correct rule can be resolved from the routing rule table, and the correct route to the customer is known. Prior to sending the packet, however, the source must be masqueraded through Source Network Address Translation (SNAT). Masquerading is a specialized form of address translation usually used to dynamically allocate PPP dialup. Now the packet is ready to send.

This simple packet example is indicative of our larger solution. Nothing new here to the world of TCP/IP, but combining all these techniques to manipulate the

packet is rare. The steps to implement this translation and interpretation are carried out by Perl scripts called by the web server's requests over a secure connection from our Citrix server.

Except for the 'Advanced Routing Solution', all other models are modifications of 'off-the-shelf' product or technology. Details for implementing the Advance Routing Solution are moved to Appendix A and B, and call heavily on the work of Brian Enright's adaptation of the 'pptpclient'

Securing the Connection Models

Because the equipment in the DMZ where the buffered outbound equipment operates is exposed to the Internet, is a target for attackers. In the Advanced Routing Solutions, the client sessions hosted on the Citrix server access a secure web connection to the Linux host that should be controlled using TCP/IP wrappers; no other node should be allowed to access the Linux host because the Linux host only client is the Citrix server(s).

Password management is very important for all the connection models. All the models, with the exception of VMWare's ESX server, one should probably store the password and customer data in a RADIUS or LDAP server isolated from the functional equipment. The physical separation of an LDAP server, for example, could be setup to accept requests from the machines in the DMZ only. At a minimum, consider an encryption schema to protect the entire customer connection data. Another possibility for securing the information is to require the user creating the tunnel to offer the information over a secure connection at the time the connection request is made. This possibility is viable if you share the password among users. Give some thought to passwords and exposure in whatever model you choose.

Another important aspect of the design when considering security, is limiting the access to the customer VPN accounts. As customers restrict VPN access solely by the accessing organization and not the particular users within the given organization, dozens of users could come and go using only a single account. This freedom removes all real accountability. When users logon the Citrix front-end to use VPN to access customer sites, they don't necessarily need the VPN account and password. The Linux VPN gateway described in this paper allows a single customer VPN account to be shared among users without giving away the customer account information to those users that require access. The request is simply trusted by the Linux VPN gateway because all Citrix users have been logged on to Citrix itself. This makes a few things very important, such as the logging of events like the logon/logoff success and failures on the Citrix server tied to the Linux 'syslog' events for VPN tunnel creation and termination. Both the success and failure data sets will be needed to know who did what when. Timing is also important, if the clocks are not synchronized, it can be difficult to

impossible to match logs. Storing and managing log files are another paper entirely, quite a few people have written on the topic and more information can be found at http://www.sans.org/rr/logging/

The link between the Citrix server and the Linux VPN gateway is un-tunneled traffic and offers a chokepoint to log traversing traffic for customer connections. When the traffic log is correlated with the logon access to Citrix and the Linux connection data, the administrator has an excellent picture of events.

Limitations of the Models

With some limitations, the advanced routing solution allows multiple accesses to sites with the same IP address from a single network. The first limitation is when multiple connections are made and delivered to the Citrix server, users of the Citrix server can see multiple customers, and customer traffic does not traverse the other's network. However, users of the Citrix server could confuse customer networks. In some cases, it might not be desirable to have a single termination end point for all customer traffic, consider the advantages and disadvantages before using the advance routing solution.

With any of the models presented, performance decreasing with every connection brought online. Depending on a number of factors, including network and processor performance, one is probably limited to how many simultaneous connections each single model maintain and is especially noticeable in the Advanced Routing Solution. In theory, the marking of packets performed by iptables in our script on the advance routing solution might be limited to 256 concurrent connections based on the number of active routing tables but is probably far less based on physical constraints. Testing for this paper has only brought 4 simultaneous connections. Scaling this solution beyond the 256 connections would require additional Linux VPN gateways, running concurrently or a rewrite of the Linux kernel. In larger organization with many simultaneous connections needed, consider setting up multiple connection models at various point of presence within the organization.

Conclusion

In an environment where you would simply allow direct VPN tunnel access to customer networks, each user creates an encrypted tunnel that is not managed or monitored. If one cares about one's customer relationship and possibly supporting the customer network then protecting the customer network and relationship is important. Managing and monitoring that connection can be as important as keeping customer network traffic from traversing the corporate Intranet. In the connection models presented here, the initial VPN connection starts from a system maintained by the systems administrator in a controlled area (the DMZ), one could verify a clean system is participating in the customer network and keep minimize the potential of harmful network traffic in either direction.

It is said that the greatest protection is afforded through narrow passages with high walls. Using a terminal server to bring up tunnel connections to customer networks affords greater protection to the Intranet while offering a chokepoint for all outbound connections; thus giving a buffer to the connection by adding a layer of security. This solution was created to solve the conflict between policy and mission; having a strict policy on 'no direct outbound VPN connections' in spite of a need to connect to customers via VPN. Alleviating the risk of VPN tunnels from dozens of users to dozens of customers can be controlled, monitored and managed through the buffered outbound connection using the connection models presented in this paper.

BIBLIOGRAPHY

Application Serving White Paper. 2000. Citrix. http://download2.citrix.com/ctxlibrary/products/pdf/APPLICATION_SERVING_WP_0700.PDF. (02 November 2002)

Cameron, James. *All Traffic Through Tunnel (Draft).* 30 December 2002. PPTP Client How To Documentation. http://pptpclient.sourceforge.net/routing.phtml#all-to-tunnel. (30 December 2002)

Enright, Brian. Perl VPN Tunnel Script. 5 Sept 2002. Siemens Information and Communications Networks. Seen in Appendix A.

Enright, Brian. *RDC Presentation.* Oct 2002. Siemens Information and Communications Networks Inc. Unpublished, available upon request.

Human X. *Run Multiple O/S Concurrently with VMWare*. 25 November 2002. LinuxDig.com Technology Articles. http://www.linuxdig.com/news_page/1038240305.php (07 January 2003)

Index of /pub/mppe. 5 August 2002. Planet Mirror Directory Listing for MPPE. http://public.planetmirror.com/pub/mppe/. (11 November 2002)

Lear, E., de Groot, G. J., Karrenberg, D., Moskowitz, Y. B., and Rekhter, Y. *Address Allocation for Private Internets, RFC1918.* February 1996. Internet RFC/STD/FYI/BCP Archives. http://www.faqs.org/rfcs/rfc1918.html. (11 November 2002)

McCabe, Linus, and Cameron, James. *Routing How To from SourceForge PPTP Client.* 25 Nov 2002. http://pptpclient.sourceforge.net/routing.phtml. (28 November 2002)

Munro, Jay. *Virtual Machines & VMware, Part II*. 21 Dec 2001. Extreme Tech. http://www.extremetech.com/article2/0,3973,13793,00.asp (07 January 2003)

Pall, G., Palter, B., Rubens, A., Townsley, W., Valencia, A., and Zorn, G., *Layer Two Tunneling Protocol "L2TP",RFC2661*. August 1999. Internet RFC/STD/FYI/BCP Archives. http://www.faqs.org/rfcs/rfc2661.html. (28 September 2002)

Pall, G., Zorn, G. *Microsoft Point-To-Point Encryption (MPPE) Protocol, RFC3078.* March 2001. Internet RFC/STD/FYI/BCP Archives. http://www.faqs.org/rfcs/rfc3078.html. (29 September 2002)

PPTP Client. 20 March 2002. SourceForge hosting of HP Sponsored PPTP Client. http://pptpclient.sourceforge.net/. (28 September 2002)

Russell, Rusty. *6.1 Source NAT, 6.2 Destination NAT.* 14 January 2002. Linux 2.4 NAT HOWTO. http://www.netfilter.org/documentation/HOWTO/NAT-HOWTO-6.html. (2 October 2002)

Terminal Services: Providing the Benefits of Remote Application Execution 15 Dec 1999. Windows 2000 Terminal Services. http://www.microsoft.com/windows2000/server/evaluation/business/terminal.asp. (5 September 2002)

VMWare ESX Server Mainframe-Class Virtual Machines on High-Performance Intel Servers. 2002. http://www.vmware.com/pdf/esx_specs.pdf (07 January 2003)

VMWare Products. 2003. http://www.vmware.com/products/ (07 January 2003)

Appendix A - Implementing the Advanced Routing Solution

Reading the overview of the PPTP Homepage is recommended prior to the implementation of the full solution outlined in the paper. Install the PPP client on the Linux VPN Gateway, available from the download section of the PPTP website. The PPP-MPPE package for the i386 system works best and has all the features used in this paper. However, not all flavors of Linux support the MPPE protocol. A complete MPPE source, for inclusion in the kernel, can be found at Public Mirror's directory listing.

We use Linux 8.0 with Apache web support to call three basic Perl Scripts that carry out VPN tunnel start, stop, and status functions. Figure 1 (Enright, RDC Presentation, slide 4), below, outlines the web server's roll. For security, the web server status.cgi will initiate a socket connection on the local host to run the script 'pptp-new' as shown.

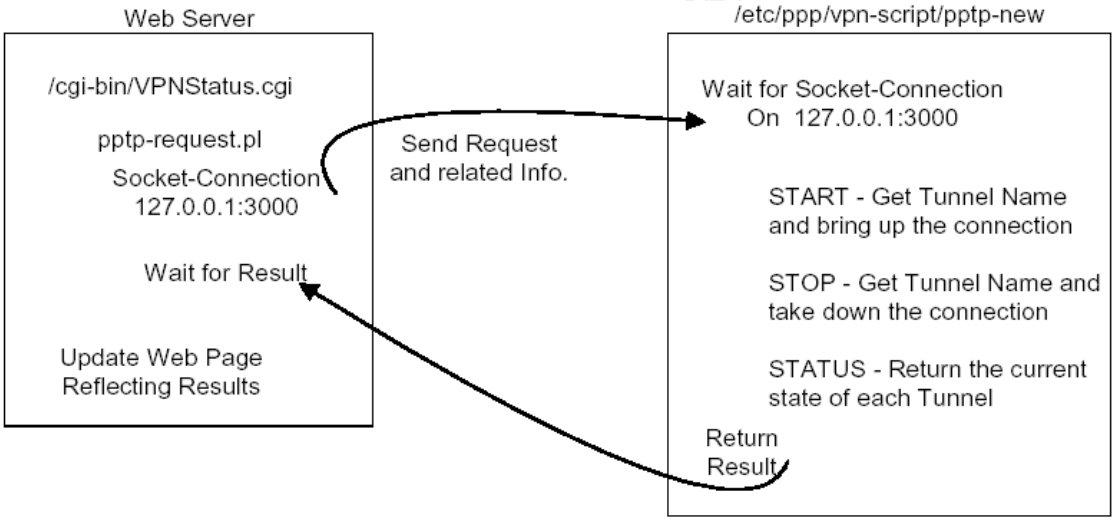

Figure 1 (Enright, RDC Presentation, slide 4)

The functions 'Start, Stop and Status' themselves are complex and some knowledge about iptables and Perl scripting is recommended. Each request calls functions similar to the examples, seen in Figures 3 (Enright, RDC Presentation, slide 4), 4 (Enright, RDC Presentation, slide 6), and 5(Enright, RDC Presentation, slide 7), to configure and use a PPTP tunnel. Another example similar to what our end functions will look like may be found at the PPTP Client Homepage. A manner to store the customer information, status, and routing information, which will be called by the scripts, is also needed. In the examples, the customer information is stored in tunnel.db and the routing information is stored in nat.db, while the user/password information for a tunnel is stored in the chap-secrets. (The schema for the necessary tables may be seen in Figure 2 (Enright, RDC Presentation, slide 4).)

/etc/ppp/vpn-database/tunnel.db

Tunnel_Name	IP/Phone#	Routing#	Mark#	UserName	Tunnel_Name	Passwd	ConnType

/etc/ppp/vpn-database/nat.db

Tunnel_Name	Fake_IP	Real_IP	Description

/etc/ppp/chap-secrets

User_Name	Tunnel_Name	Password

Figure 2 (Enright, RDC Presentation, slide 4)

Figure 3 (Enright, RDC Presentation, slide 4) displays the start function steps. It retrieves the tunnel name from the request, and looks up the associated information from the databases. It connects to Customer LAN using Dial-Up or VPN (IP: w.x.y.z DEV: pppX); and creates a new routing table. Next, the rule and route for the new table are added . For each entry in the nat.db database associated with the tunnel name, the packets are marked and associated with the Tunnel Name. After this has completed, the return packets are masqueraded and the routing cache is flushed. Lastly, a lock file is created at pptp-<tunnel_name>, where <tunnel_name> is the name given to the host tunnel.

START
#echo <*Table_Number*> <*Tunnel_name*> >> /etc/iproute2/rt_table
#ip rule add fwmark <*Mark_#*> table <*Tunnel_name*>
#ip route add default via w.x.y.z dev pppX table <*Tunnel_name*>
// 'mark and dnat' must be carried out for each entry in the 'nat' and 'mangle' database associated with the Tunnel Name while the masquerading needs to be only done once
#iptables -t mangle -A PREROUTING -d <*Fake_IP*> -j MARK --set-mark <*Mark_#*>
#iptables -t nat -A PREROUTING -d <*Fake_IP*> -j DNAT --to <REAL_IP>
#iptables -t nat -A POSTROUTING -o pppX -j MASQUERADE
#ip route flush cache
#create LOCK file /var/lock/subsys/pptp/pptp-<*Tunnel_Name*>

Figure 3 (Enright, RDC Presentation, slide 4)

The stop function, Figure 4 (Enright, RDC Presentation, slide 6), also begins by recovering the tunnel name from the request, and looks up associated information from the databases. It reads the lock file at pptp-<tunnel_name>, and removes the routing table entry from the rt_tables. The rule and route is deleted for this table. For each entry in the nat.db database associated with the tunnel name, all dynamically created mark and nat entries are cleaned, the

masquerading is also cleared (for reverse routing), the routing cache is flushed and the lock file for the tunnel name is deleted.

STOP
#ip rule del fwmark <*Mark_#*> table <*Tunnel_name*>
#ip route del default via w.x.y.z dev pppX table <*Tunnel_name*>
// 'mark and dnat' must be carried out for each entry in the 'nat' and 'mangle' database associated with the Tunnel Name while the masquerading needs to be only done once
#iptables -t mangle -D PREROUTING -d <*Fake_IP*> -j MARK --set-mark <*Mark_#*>
#iptables -t nat -D PREROUTING -d <*Fake_IP*> -j DNAT --to <*REAL_IP*>
#iptables -t nat -D POSTROUTING -o pppX -j MASQUERADE
#ip route flush cache
#delete LOCK file /var/lock/subsys/pptp/pptp-<*Tunnel_Name*>

Figure 4 (Enright, RDC Presentation, slide 6)

Seen in Figure 5 (Enright, RDC Presentation, slide 7), status is used to determine the current state of each tunnel. This test is relatively simple. The pptp directory is read. All tunnels in the tunnel.db table that have a lock file (pptp-<tunnel name) are marked as 'UP', while all others are marked 'DOWN'.

STATUS - Return the current state of each Tunnel
- Read the directory /var/lock/subsys/pptp/
- Mark all Tunnels with a file "pptp-<*Tunnel_Name*>" as being UP
- Otherwise mark it as being DOWN
- Return the list to the web Server

Figure 5 (Enright, RDC Presentation, slide 7)

For a working VPN tunnel script (in Perl) that carries out the start and stop functions, please see Appendix A (Enright, Perl VPN Tunnel Script).

Using the advance connection model does not solve all one's problems and responsibilities; it just shifts them to the DMZ. One will need to take steps in securing the DMZ equipment, like the Linux advanced router and the Citrix or terminal server, just as you would within your corporate network.

APPENDIX B – Perl VPN tunnel script carrying out the start and stop functions (based on open source script offered at http://pptpclient.sourceforge.net and is provided as a working example)

```
#!/usr/bin/perl -w
#functions:
# setup - configures tunnel servers and chap-secrets
# start - brings up a tunnel
# stop  - brings down a tunnel
#
#  09/05/02 -- Brian Enright
#  modified and stream lined to support Multiple VPN connections
#  by accessing connection data through the databases
#  nat.db and tunnel.db will be used
#  This is a modified version of the PPTP-COMMAND script
#   Provided by the PPPTP-COMMAND software included with the
#   PPTP client down a tunnel $Id:
#   pptp-command,v 1.12 2001/12/20 13:09:02 jwiedemeier Exp $
#######
use IO::Socket;

#
# database access module...
#
my $script_dir = "/etc/ppp/vpn-script";

require "$script_dir/getDataFromDB.pm";
require "$script_dir/tunnels_up.pm";

# Data
#
# the regexp for the list of characters that are unsafe
# to put inside a system() or ``
# it is built by saying everything but known safe characters
# anyone want to make bets on if this holds true for i18n'ed systems?
my $safe_set  = '-A-Za-z0-9\s\._\/:';
my $unsafe_re = "[^$safe_set]";
my $safe_re   = "[$safe_set]*";


#
# pppdir - the directory containing the ppp config files
#
my $pppdir = $ENV{"PPPDIR"};
die "Stop screwing with me and set PPPDIR to something reasonable\n" if
defined
```

```
$pppdir && $pppdir =~ /$unsafe_re/o;
$pppdir = "/etc/ppp" unless defined $pppdir;

#
# chap_secrets - the full path to the the CHAP
#       (Challenge/Handshake Authentication Protocol) secrets file
#
my $chap_secrets = "$pppdir/chap-secrets";
my $pap_secrets = "$pppdir/pap-secrets";

#
# subsys_dir - the place "rc" looks to see if a servics is started
#              before it runs the K* scripts
my $subsys_dir = "/var/lock/subsys/pptp";

#
# clean up the path since this is run as root.
$ENV{PATH} = "/bin:/usr/bin:/usr/sbin";
delete $ENV{BASH_ENV};
delete $ENV{IFS};
delete $ENV{ENV};

sub usage() {
       print "\nusage: $0 [start|stop|deamon|status] tunnel_name\n\n";
       exit 1;
}


#ConfiguredTunnels
#
# Returns a list of configured tunnels
#
#
# needs to read tunnel.db and return an array of the Tunnel Names
#
sub ConfiguredTunnels() {
  my @tunnels = ();
  my @result = get_tunnel_data("");
  my $rec;
  foreach $rec (@result) {
     push @tunnels, $rec->{"Tunnel"};
  }

  return @tunnels;
}
```

```
#QueryUser <prompt> <default>
#
# Ask the user <prompt> and return the answer, <default> if cr
#
sub QueryUser($$) {
	my ($prompt, $default) = @_;

	print "$prompt";
	print " [$default]" if defined $default;
	print ": ";
	my $answer = <STDIN>;
	chomp $answer;
	$answer = $default if $answer eq "" and defined $default;
	return $answer;
}



#bselect
#
# a rough equilivent of the bourne shell's select
sub bselect($@) {
	my $prompt = shift;
	my @choices = @_;
	for my $i (0..$#choices) {
		print $i+1 .".) $choices[$i]\n";
	}
	#print $i+1 .".) Abort\n";
	my $reply = QueryUser $prompt, undef;
	return $reply;
}



#SelectTunnel - interactive
#
# Prints $_[0] as a prompt and returns the choice.
#
sub SelectTunnel($$) {
	my ($msg, $option) = @_;
	my $tunnel = "";
	my @tunnels;
	if ($option eq "UP") {
	  @tunnels = get_tunnels_up($subsys_dir);
```

```
        }elsif ($option eq "ALL") {
          @tunnels = ConfiguredTunnels;
        }

        while($tunnel eq "") {
                #$tunnel = bselect $_[0], @tunnels;
                $tunnel = bselect $msg, @tunnels;
        }
        return $tunnels[$tunnel - 1] if $tunnel =~ /^\d+$/;
        return $tunnel if grep {/$tunnel/} @tunnels;
        return "";
}

#start
#
# This does the old pptp-start work
sub start() {
        my ($tunnel, $f, @filter, @ifs, $if, @foo);
        my @tunnels = ConfiguredTunnels;
        die "no configured tunnels!\n" if @tunnels == 0;

        if(defined $ARGV[1]) {
                $tunnel = $ARGV[1];
        } elsif(-t STDIN && -t STDOUT) {
                $tunnel = SelectTunnel "Start a tunnel to which server?",
"ALL";
        } else {
                usage;
        }


        #
        #  if $subsys_dir/pptp-$tunnel exists
        #    die "LINK for $tunnel is already up...\n"
        #  else continue
        #
        if( -f "$subsys_dir/pptp-$tunnel") {
           die "LINK for $tunnel is already up...\n";
        }

        die "Nasty characters in $tunnel\n" if $tunnel !~ /^($safe_re)$/o;
        $tunnel = $1;

        #
        #  get Connection information form databases
        #
```

```
        my @tmp_tunnel = get_tunnel_data($tunnel);
        my $Tunnel_data = $tmp_tunnel[0];
        my @nat_data = get_nat_data($tunnel);

        if ($tunnel ne $Tunnel_data->{'Tunnel'}) {
          die "ERROR:: getting data for $tunnel\n";
        }


        #build a regexp of the currently existing interfaces
        my @ifconfig = `/sbin/ifconfig`;
        foreach $f (@ifconfig) {
                next unless $f =~ /^[a-z]/;
                @foo=split ' ', $f;
                push @filter, $foo[0];
        }
        my $if_re = join '|', @filter;

        #bring up the tunnel
        my $server = $Tunnel_data->{'ServerIP'};
        my $uname =  $Tunnel_data->{'UserName'};
        my $rname = $Tunnel_data->{'RemoteName'};
        my $conntype = $Tunnel_data->{'ConnType'};
        $conntype =~ s/^\s+//;
        $conntype =~ s/\s+$//;

        my $retry_count = 1;
RETRY:
        my $child = fork;
        if ($child == 0) {

          if ($conntype eq "VPN") {
                exec "/usr/sbin/pptp $server name $uname remotename $rname
disco
nnect \"$script_dir/pptp-new stop $tunnel\" file /etc/ppp/options.pptp";
          }elsif($conntype eq "DialUP") {
                exec "/usr/sbin/pppd /dev/ttyS4 connect \"/usr/sbin/chat -T
$ser
ver -f $script_dir/dial.scr\" name $uname remotename $rname disconnect
\"$script
_dir/pptp-new stop $tunnel\" file /etc/ppp/options.dialup";
          }

          die "exec of pppd failed.";
        }
```

```
        my $timeout=30;
        while(1) {
                if (($timeout == 0) && ($retry_count > 0)) {
                  $retry_count--;
                  # print "DEBUG:: connection faild attempting to
retry\n\n";
                  goto RETRY;
                }elsif ($timeout == 0) {
                  die "ERROR!  Connection timed out.\n";
                }
                $timeout--;
                @ifs = ();
                sleep 1;
                @ifconfig=`/sbin/ifconfig`;
                foreach $f (@ifconfig) {
                        next unless $f =~ /^[a-z]/;
                        @foo=split ' ', $f;
                        push @ifs, $foo[0];
                }
                ($if, undef) = grep {!/$if_re/} @ifs;
                last if defined $if;
        }
        die "something screwy in your interface names: $if\n" if $if !~
/^($safe
_re)$/o;
        $if = $1;
        (grep {/inet/} `/sbin/ifconfig $if`)[0] =~ /:(\d+\.\d+\.\d+\.\d+)/;
        $ip = $1;
        (grep {/P-t-P/} `/sbin/ifconfig $if`)[0] =~
/P-t-P:(\d+\.\d+\.\d+\.\d+)/
;
        my $pptpip = $1;


#
# add $tunnel to routing table using RT_Num
#
   open(RTTABLE, ">>/etc/iproute2/rt_tables") or die "ERROR opening
/etc/iproute
2/rt_tables\n\n";
   flock(RTTABLE, 2);
   seek(RTTABLE,0,2);
   print RTTABLE "$Tunnel_data->{'RT_num'} $tunnel\n";
   close RTTABLE;

#
```

```
# add rule for new routing table using Mark_num
#
  system("/sbin/ip rule add fwmark $Tunnel_data->{'Mark_num'} table
$tunnel");

#
# add MANGLE translations
# add NAT translations
#
  my $nat_entry;
  foreach $nat_entry (@nat_data) {
    system("/sbin/iptables -t mangle -A PREROUTING -d
$nat_entry->{'FakeIP'} -j
 MARK --set-mark $Tunnel_data->{'Mark_num'}");
    system("/sbin/iptables -t nat -A PREROUTING -d $nat_entry->{'FakeIP'}
-j DN
AT --to $nat_entry->{'RealIP'}");
  }


#
# NOW add default route to routing table $tunnel
#
   system("/sbin/ip route add default via $pptpip dev $if table $tunnel");

#
# NOW add MASQUERADING for interface pppX
#
   system("/sbin/iptables -t nat -A POSTROUTING -o $if -j MASQUERADE");


#
# Make sure ftp modules are loaded...
#
   my $conntrack_ftp = (grep {/ip_conntrack_ftp/} `/sbin/lsmod`)[0];
   if (!defined $conntrack_ftp) {
    system("/sbin/insmod ip_conntrack_ftp");
    print "\n\nip_conntrack_ftp is NOW loaded\n\n";
   }

   my $nat_ftp = (grep {/ip_nat_ftp/} `/sbin/lsmod`)[0];
   if (!defined $nat_ftp) {
     system("/sbin/insmod ip_nat_ftp");
     print "\n\nip_nat_ftp is NOW loaded\n\n";
   }
```

```
        my @pppd_list = (grep {/pppd/} `/bin/ps ax`);
        my $num=0;
        for ($num=0; $num<=$#pppd_list; $num++) {
          if (index($pppd_list[$num], "<defunct>") > 0 ) {
              splice(@pppd_list, $num, 1);
              $num = $num-1;
          }
        }
        my $pppd_pid = $pppd_list[(@pppd_list-1)];
        $pppd_pid =~ s/^\s+//;
        $pppd_pid = (split(' ',$pppd_pid))[0];

        open(LOCK, ">>$subsys_dir/pptp-$tunnel") or die "couldn't open lock
file
: $!";
        print LOCK "$pppd_pid\n";
        print LOCK "$if\n";
        print LOCK "$ip\n";
        print LOCK "$pptpip\n";
        close LOCK or die "couldn't close lock file: $!";

        print "Tunnel $tunnel is active on $if.  IP: $ip.  P-t-P:$pptpip\n";


#
# MAKESURE WE FLUSH THE ROUTING CACHE!!!!
#
   system("/sbin/ip route flush cache");

   exit 0;
}



#stop
#
# this does the old pptp-stop work
sub stop() {

        my ($tunnel, $f, @filter, @ifs, $if, @foo);
        my @tunnels = ConfiguredTunnels;
        die "no configured tunnels!\n" if @tunnels == 0;

        if(defined $ARGV[1]) {
              $tunnel = $ARGV[1];
```

```
        } elsif(-t STDIN && -t STDOUT) {
                $tunnel = SelectTunnel "Start a tunnel to which server?",
"UP";
        } else {
                usage;
        }


        die "Nasty characters in $tunnel\n" if $tunnel !~ /^($safe_re)$/o;
        $tunnel = $1;

        #
        #  if $subsys_dir/pptp-$tunnel NOT exists
        #    die "LINK for $tunnel is NOT up...\n"
        #  else continue
        #
        if(! -f "$subsys_dir/pptp-$tunnel") {
           die "LINK for $tunnel is NOT up...\n";
        }


        #
        #  get Connection information form databases
        #
        my @tmp_tunnel = get_tunnel_data($tunnel);
        my $Tunnel_data = $tmp_tunnel[0];
        my @nat_data = get_nat_data($tunnel);

        if ($tunnel ne $Tunnel_data->{'Tunnel'}) {
          die "ERROR:: getting data for $tunnel\n";
        }


        #
        # Get info from the lock file (order is important, see sub start)
        #
        open(LOCK, "<$subsys_dir/pptp-$tunnel") or die "ERROR unable to open
$su
bsys_dir/pptp-$tunnel";
        my $pptppid = <LOCK>;
        my $if_dev = <LOCK>;
        my $ip_addr = <LOCK>;
        my $pptp_addr = <LOCK>;
        chomp $pptppid;
        chomp $if_dev;
        chomp $ip_addr;
```

```
        chomp $pptp_addr;
        close LOCK;


        # sending pptp process SIGHUP... (ie STOPING pptp)
        print "Sending HUP signal to PPTP processes $pptppid...\n";
        system("kill -s HUP $pptppid");


# ----May NOT be needed???
# delete default route established during sub start
#
    system("/sbin/ip route del default via $pptp_addr dev $if_dev table
$tunnel"
);

#
# del rule for new routing table using Mark_num
#
    system("/sbin/ip rule del fwmark $Tunnel_data->{'Mark_num'} table
$tunnel");
#
# del $tunnel to routing table using RT_Num
#
   open(RTTABLE, "/etc/iproute2/rt_tables") or die "ERROR opening
/etc/iproute2/
rt_tables\n\n";
   @rttable_data = <RTTABLE>;
   close RTTABLE;

   my $i = 0;
   my $index = -1;
   foreach $line (@rttable_data) {
      if ((index($line,"#") == 0) or (length($line) < 3)) {
        $i = $i + 1;
        next;
      }
      my ($rtNum, $rtName) = split(' ',$line);
      if (($rtNum eq $Tunnel_data->{'RT_num'}) and ($rtName eq
$Tunnel_data->{'
Tunnel'})) {
          $index = $i;
      }

      $i = $i + 1;
   }
```

```
  if ($index >= 0) {
    splice(@rttable_data, $index, 1);
  }

  open(RTTABLE, ">/etc/iproute2/rt_tables") or die "ERROR openeing
/etc/iproute
2/rt_tables\n\n";
  flock(RTTABLE, 2);
  seek(RTTABLE,0,0);
  print RTTABLE @rttable_data;
  close RTTABLE;


#
# del MANGLE translations
# del NAT translations
#
  my $nat_entry;
  foreach $nat_entry (@nat_data) {
    system("/sbin/iptables -t mangle -D PREROUTING -d
$nat_entry->{'FakeIP'} -j
 MARK --set-mark $Tunnel_data->{'Mark_num'}");
    system("/sbin/iptables -t nat -D PREROUTING -d $nat_entry->{'FakeIP'}
-j DN
AT --to $nat_entry->{'RealIP'}");
  }

#
# del MASQUERADEING
#
  system("/sbin/iptables -t nat -D POSTROUTING -o $if_dev -j MASQUERADE");

      unlink "$subsys_dir/pptp-$tunnel";
      sleep 2;

#
# MAKESURE WE FLUSH THE ROUTING CACHE!!!!
#
  system("/sbin/ip route flush cache");

  exit 0;
}


sub get_status_all {
```

```
  my @status_list = ();
  my @tunnel_list = ConfiguredTunnels;
  my $index = 0;
  foreach my $tunn (@tunnel_list) {
    chomp $tunn;
    my $tunnel_data = (get_tunnel_data($tunn))[0];

    if (is_tunnel_up($subsys_dir,$tunn) == 1) {
        $status_list[$index] = "$tunn,UP,".$tunnel_data->{'ConnType'};
    }else {
        $status_list[$index] = "$tunn,DOWN,".$tunnel_data->{'ConnType'};
    }
    $index++;
  }
  return @status_list;
}


sub status{
   my @tmp = get_status_all;
   print "The Status for each Tunnel is as follows::\n";
   foreach my $tmp (@tmp) {
        chomp $tmp;
        print "$tmp\n";
   }
   print "\n\n\nDisplay as a single List!\n";
   my $list = join '|', @tmp;
   print $list;
   exit 0;
}




sub deamon{

  my $server = IO::Socket::INET->new (
        LocalAddr => '127.0.0.1',
        LocalPort => 3000,
        Type => SOCK_STREAM,
        Reuse => 1,
        Listen => 5
  ) or die "\n\npptp-new:: ERROR Binding to port 3000\n\n";
  while() {
    while(my $client = $server->accept()) {
      my $action = <$client>;
```

```
    chop $action;

    #print "\n\nDEBUG1:  action=>($action)\n\n";

    if ($action eq "start") {
       my $tunnel = <$client>;
       chomp $tunnel;
       #print "DEBUG2:  tunnel=>($tunnel)\n\n";
       my @cmd = ("$script_dir/pptp-new", "start", $tunnel,
">/dev/null");
       system(@cmd);
       #
       # sleep 20;
       #
       my $result = is_tunnel_up($subsys_dir, $tunnel);
       if ($result == 1) {
         $result = "SUCCESS!!";
       }else {
         $result = "FAILED!!";
       }
       print $client "Request -- $result\n";

    }elsif($action eq "stop") {
       my $tunnel = <$client>;
       chomp $tunnel;
       #print "DEBUG3:  tunnel=>($tunnel)\n\n";
       my @cmd = ("$script_dir/pptp-new", "stop", $tunnel, ">/dev/null");
       system(@cmd);
       #
       # sleep 10;
       #
       my $result = is_tunnel_up($subsys_dir, $tunnel);
       if ($result == 0) {
         $result = "SUCCESS!!";
       }else {
         $result = "FAILED!!";
       }
       print $client "Request -- $result\n";

    }elsif($action eq "status") {
       my @status = get_status_all;
       my $list = join '|', @status;
       print $client $list;
    }else {
       print "\n\nERROR: action ($action) unknown\n\n";
    }
```

```
      close($client);
    }
  }
  close($server);
  exit 0;
}

if(defined $ARGV[0]) {

        if($ARGV[0] eq "start") {
            start;
        } elsif($ARGV[0] eq "stop") {
            stop;
        } elsif($ARGV[0] eq "status") {
            status;
        } elsif($ARGV[0] eq "deamon") {
            deamon;
        } else{
            usage;
        }
}


if(! -t STDIN || ! -t STDOUT) {
        usage;
}

my $mode = bselect "What task would you like to do?", "start", "stop",
"status",
 "deamon", "quit";
if($mode eq "1") {
        start;
} elsif($mode eq "2") {
        stop;
} elsif($mode eq "3") {
        status;
} elsif($mode eq "4") {
        deamon;
} elsif($mode eq "5" or $mode eq "q") {
        exit 0;
}
```