Global Information Assurance Certification Paper

SANS GSEC Practical Assignment
Matt Wilbur
San Diego Mentor-lead Security Essentials
Version 1.4b

Implementing meshed IPSEC VPNs using Netscreen Firewalls

Abstract

Virtual private networks (VPNs) give an organization considerable flexibility in choosing a network topology, and provide a level of privacy and integrity equal or better to Frame Relay or other private line means of connectivity.

This paper describes a medium-sized company's migration from a bottlenecked hub-and-spoke Frame Relay Wide Area Network (WAN) topology to dedicated Internet access for each site, with meshed site-to-site VPNs used to create a "virtual" company network. Security is always about trade-offs, so the issues of added complexity, added support costs, and reduced network costs are also discussed.

Introduction

Company X is a company with roughly 200 employees in six offices around the US. The "desktop" computing environment consists of Windows NT 4.0 and 2000 on the desktop. NT 4 servers provide domain services, file/print services, and Exchange for email/groupware. On the backend a diverse range of UNIX hosts are used for modeling and simulation, including SGI, Sun, Linux and FreeBSD systems. Inter-office connectivity is required for "infrastructure" basics like email, corporate intranet access, and file sharing. Additionally, the WAN allows the company's staff to utilize remote company computing resources, running interactive X11-based jobs remotely displaying locally, number crunching jobs on remote systems, and even performing demonstrations for customers visiting our remote offices. The WAN is the lifeblood of this company's computing infrastructure. It should also be mentioned that for a number of reasons (research environment, DoD contracts) company X is very security conscious and considers protection of company data to be of paramount importance.

"WAS" WAN configuration

Prior to this project, company X's corporate headquarters, located in Phoenix, had the company's only Internet connection. Each office had a 256K private frame relay link connecting them directly to the corporate HQ in a hub and spoke configuration (see figure 1.) One obvious limitation of this configuration is if someone in Arlington wanted to send data to someone in Orlando, their packets had to travel across the US once to reach Phoenix, then back across the country to reach Orlando. 256kb/sec was already insufferably slow for running interactive applications, and with two cross-country trips over slow lines the WAN was often unusable. Additionally, the company's monthly data services bill, comprised of Frame Relay and Internet charges, was around

$9000/month, at a time when a T1 from a good ISP could be had for approximately $1000/month. Speed and cost are important issues, but even more important was that if the Internet line was down, not only was the Phoenix site cut off from the Internet, but the rest of the company was as well.

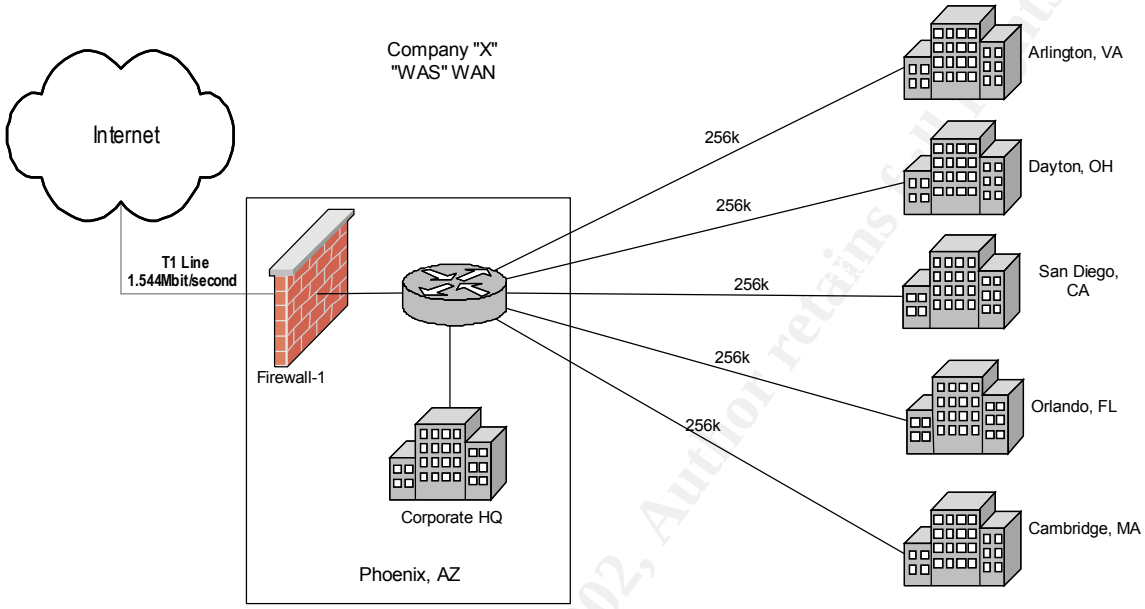

Figure 1

This configuration was not without its strong points. With one firewall, IT had one Internet access point which required security controls. We only required one NIDS to monitor external and DMZ networks, and had one Cisco ACL to manage.

"New" WAN Configuration

Our evaluation of current generation VPN products, including Cisco, Checkpoint, Sonicwall, and Netscreen offerings, led us to believe that we could migrate from this configuration to a topology where each office would have dedicated Internet access, with meshed VPNs connecting each office to each other office. After numerous product evaluations, we decided on Netscreen-10 firewall appliances to build our new WAN. We liked their appliance nature, their utilization of ASICs to do fast crypto processing, their lack of an underlying, potentially vulnerable OS we would have to maintain, and the flexibility of their CLI and Web UIs. We were able to convert 99% of our Firewall-1 objects, including addresses, services, and users, into Netscreen CLI-ingestible entries using a simple perl script. Also, some of our remote offices had very junior or no IT staff, so we planned to keep a spare Netscreen available, manage the configurations under a configuration management system like RCS, thereby ensuring we could bring any site in the company back up within 24 hours without requiring significant on-site expertise. To complete the background information, the planned physical topology of the new WAN is depicted in Figure 2.

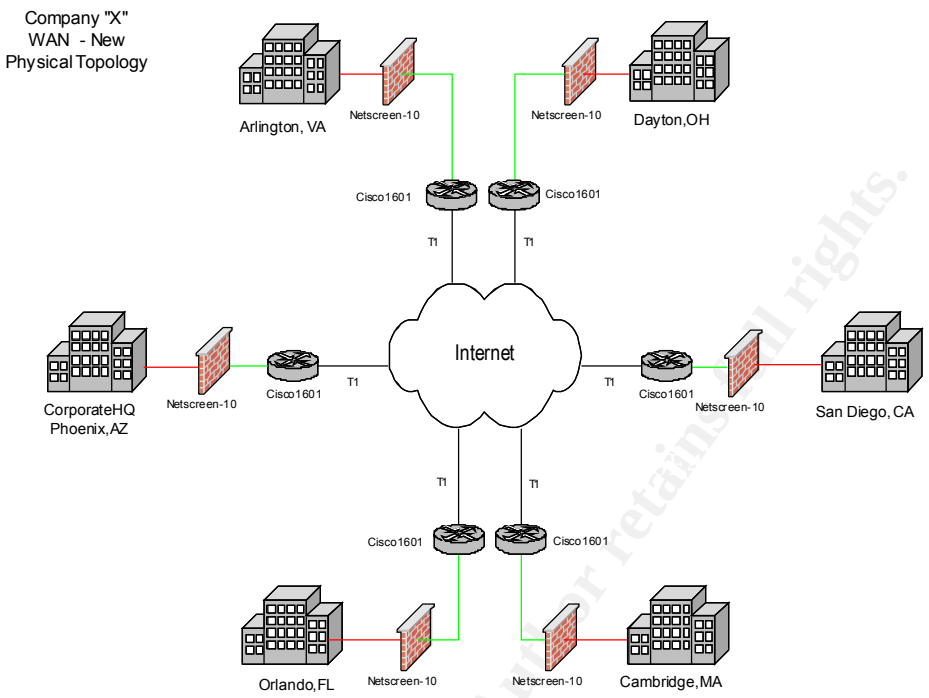

Figure 2

The planned logical topology, one of meshed VPNs, is illustrated in Figure 3.

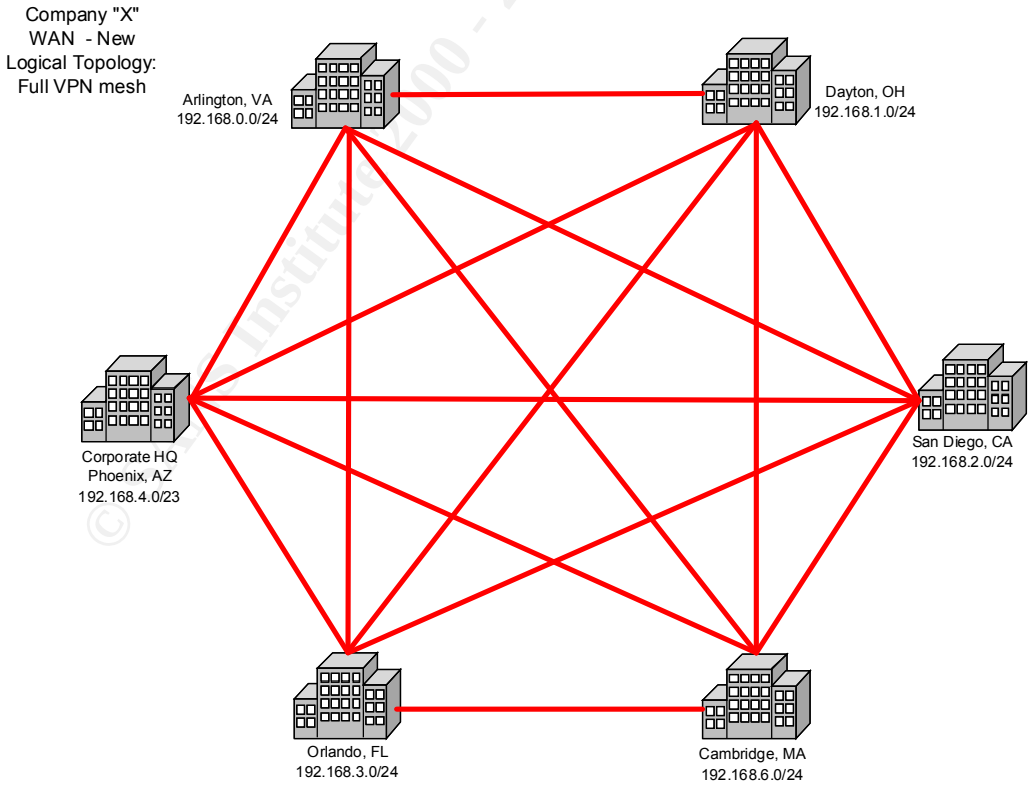

Figure 3

Implementation

For the purposes of this paper I will be focus on the configuration of three sites, Phoenix, Dayton, and Arlington. The configuration of the remaining sites is left to the reader, as they didn't differ other than IPs and IKE keys. Additionally, I will be abbreviating the process we used as the means we used to do this cut-over site by site without impacting workflow could be the subject of an additional paper, and would probably obfuscate the primary theme.

Next we gathered the information we'd need to build this system. We built a table of all the important IPs we would need. For our three sites, it looks something like this (router information excluded intentionally):

Office	Internal network	Netscreen "Untrust" IP
Phoenix	192.168.0.0/24	10.0.20.1
Dayton	192.168.1.0/24	10.0.25.1
Arlington	192.168.4.0/23	10.0.30.1

I'm expressing the networks with CIDR notation. If that is foreign to you, a /24 means 24 "1"s in the netmask, or 255.255.255.0. See the CIDR FAQ[1] for more information. Also, the "Untrusted" interface using Netscreen terminology is the public or "outside" interface. The next planning step required some scrutiny of IPSEC. This isn't intended to be a complete description of IPSEC, so I'm leaving out some things like the distinction between tunnel and transport mode (all of our configurations are using tunnel mode) and the wrinkles of using NAT+IPSEC, etc.

IPSEC offers two encapsulation methods, Authentication Header (AH), and AH with Encapsulated Security Payload (ESP). As you may have guessed from the names, AH ensures that each end of the IPSEC tunnel is who they say they are (foiling man-in-the-middle, traffic replay and other attacks), and that data is received exactly as it was sent, but it offers no encryption of the payload. ESP is the means to encrypt the payload with a variety of available ciphers and key exchange methods, ensuring the traffic is unreadable by bad guys when it hits the wire. AH + ESP offers the most security, so for us this choice was a no-brainer, it met the criteria of authentication, integrity, non-repudiation (AH), and confidentiality (ESP). Next we had to choose a hash method, a cipher, and a keying method.

The Netscreen offers two hash algorithms for message integrity, MD-5 and SHA-1. Since our tests showed that these Netscreens had plenty of horsepower, we chose the strongest algorithm, SHA-1 for our hashes. By strong in this context I meant the algorithm that uses the most bits to produce a hash. The MD5 hash uses 128 bits to describe its hash[2], while SHA-1 uses 160 bits[3]). Both have been scrutinized for years, so the chances of someone "cracking" either algorithm are negligible. Next was our

[1] InetDaemon Enterprises, p1.
[2] Rivest, p1.
[3] National Institute of Standards and Technology FIPS 180-2, p3.

cipher. At the time of this implementation, Netscreen only offered DES and Triple DES (3DES) block ciphers for ESP mode. Once again, having horsepower to spare, we went with 3DES as our cipher. DES is officially considered "weak" given the computing power available these days. Newer Netscreen operating systems offer the newer NIST Advanced Encryption Standard (AES)[4] ciphers in the form of AES-128, AES-192 and AES-256. So we have a cipher, but as you probably know, DES and 3DES are both symmetric key ciphers. How will we ensure both sides use the same secret key?

With most IPSEC implementations you have two primary means to key your tunnel, manual keying, and automatic key exchange using a protocol called Internet Key Exchange (IKE), formerly called ISAKMP/Oakley. Manual keying is simple, you manually configure each IPSEC participant with the same secret key, and you're on your way. For testing manual keying can be quite helpful, but as you may guess, it doesn't scale well. IKE provides a means, using public key cryptography (Diffie-Hellman key exchange) to exchange secret keys in a secure fashion. It also will periodically re-key the tunnel based on a time or traffic volume interval. Netscreen's IKE provides two ways for each side to identify itself, one is using a pre-shared key that each side must know (sound familiar?), the other is to use x.509 certificates. Since we had no Public Key Infrastructure in place at the time of this deployment, we opted for using IKE with pre-shared keys.

I think I've covered enough background; I'll now describe the actual setup. I will lean toward using the command-line interface commands, only referring to the GUI when it is helpful. The first step, abbreviated for the sake of the paper as it's out of the paper's scope, was to cut each office over from its Frame Relay link to it's own Internet T1. Before we could do this, we had to build the router configurations. We configured and installed in site a Cisco 1601 with a very basic configuration, following the example given in the NSA's Router Security Configuration guide[5] and implementing strict ingress and egress ACLs per BCP46[6] blocking potentially spoofed traffic.

Next, in each office we installed a Netscreen-10 which we had pre-configured to use the existing "internal" site IP addresses, running in Network Address Translation (NAT) mode. NAT translated the internal addresses to the external IP addresses we were provided by the new ISP. Using the IP's I showed you before, here's how we set up the Phoenix Netscreen from an SSH session, so far:

```
set interface trust ip 192.168.0.254 255.255.255.0
set interface untrust ip 10.0.20.1 255.255.255.0
set interface untrust gateway 10.0.20.254
```

The Dayton and Arlington Netscreens were configured in similar fashion. Internet access was tested and verified.

[4] National Institute of Standards and Technology AES Fact sheet, p1.
[5] National Security Agency, p82.
[6] Killalea, 4.3-4.4.

Next we went about configuring a number of "objects" the Netscreens would need to build tunnels between these firewalls. We needed to describe to each Netscreen the IP networks that would eventually be reached behind the other VPN endpoints. Again, on the Phoenix firewall:

```
set address untrust "Dayton-192.168.1/24" 192.168.1.0 255.255.255.0
set address untrust "Arling-192.168.4/23" 192.168.4.0 255.255.254.0
```

Basically address book entries, the "set address" fields are as follows: "untrust" tells the netscreen this network is reachable via it's "outside" or "public" interface, "Dayton-192.168.1/24" is a unique descriptive name, 192.168.1.0 is the network number, and 255.255.255.0 is the netmask. On the Dayton firewall:

```
set address untrust "Phoenix-192.168.0/24" 192.168.0.0 255.255.255.0
set address untrust "Arling-192.168.4/23" 192.168.4.0 255.255.254.0
```

On the Arlington firewall:

```
set address untrust "Dayton-192.168.1/24" 192.168.1.0 255.255.255.0
set address untrust "Phoenix-192.168.0/24" 192.168.0.0 255.255.255.0
```

You won't see these address entries until later, when we actually implement policies that refer to them. Next we needed to build the VPN properties. On the Netscreen this requires configuring a "VPN Gateway", then an "Autokey IKE entry". The VPN gateway is the entry that requires a pre share key, so we have a brief key-generating interlude. Rather than use easy to remember (hence easy to crack) pre-shared keys, we opted to generate them using the /dev/arandom device on an OpenBSD system. The following command line is something like what we used to build pre-shared keys on an OpenBSD host:

```
cat /dev/arandom |head -100 | strings| sed s/{\s\\t042}.//g \
| tr -d '\012'|cut -c 1-32
```

This produces a random-enough stream of 32 ASCII characters. To explain, /dev/arandom produces an endless stream of random bytes, head -100 gives us the first 100 lines, strings converts the byte values to ascii, the sed expression removes spaces tabs and double quotes from the stream (the Netscreen can't handle double quote characters in its pre-shared keys). The tr command chops newlines from the character stream, and cut –c 1-32 gives us the first 32 characters.

Here's the command we used to build the Dayton VPN Gateway on the Phoenix firewall ("\" used to denote continuation of the line – it is not recognized by the Netscreen CLI. I'll continue to use this notation throughout.):

```
set ike gateway "phoenix-gw" ip 10.0.20.1 Main preshare\
".McqW<(r6QdP#?f@DwiyjQ~S>=U{rsmE" proposal "pre-g2-3des-sha"
```

The “set ike gateway” command parameters, in order, are user-chosen gateway name, IP of the untrusted interface on the remote gateway, in this case the Phoenix Netscreen. Main indicates Main mode IKE (affords better identity protection than the alternative, aggressive mode), preshare indicates we’re using preshared keys, the long hex number is the preshared key we generated earlier, and “proposal pre-g2-3des-sha” indicates our Phase 1 proposal will be using preshared key (pre), Diffie-Hellman group 2 (g2), triple-des (3des) and the SHA-1 hash mechanism.

This is a good time to talk about the last remaining IPSEC wrinkle we haven’t yet discussed, that of proposals. Without delving too deeply into the nuts and bolts, IKE negotiation consists of two phases, called Phase 1 and Phase 2. In Phase 1, IKE creates an authenticated, secure channel between the two IPSEC peers using Diffie-Hellman key exchange. Once Phase 1 is completed, Phase 2 uses the established secure channel to negotiate what’s known as Security Associations, or SAs. SAs are basically agreements between two peers describing how they will communicate securely. They’re a key component of IPSEC, whether you’re using manual or automatic keying. With manual keying you describe each SA by hand, but with IKE they’re created in Phase 2 negotiation.

So we’ve configured the Phoenix-gw on the Dayton firewall, let’s finish making all the IKE gateways:

On the Dayton firewall:

```
set ike gateway "arln-gw" ip 10.0.30.1 Main preshare \
"G[GbU|@YrR^tO<rhw2]OChU;Qe\j1_?`" proposal \
"pre-g2-3des-sha"
```

On the Phoenix firewall:

```
set ike gateway "dayton-gw" ip 10.0.25.1 Main preshare\
".McqW<(r6QdP#?f@DwiyjQ~S>=U{rsmE" proposal \
"pre-g2-3des-sha" (Note the key matches the phoenix-gw key on
the Dayton firewall)

set ike gateway "arln-gw" ip 10.0.30.1 Main preshare\
"HZ2:8h<O0.SEhM(i-\d^,~XFWf*%{iG^" proposal \
"pre-g2-3des-sha"
```

On the Arlington firewall:

```
set ike gateway "dayton-gw" ip 10.0.25.1 Main preshare\
"G[GbU|@YrR^tO<rhw2]OChU;Qe\j1_?`" proposal \
"pre-g2-3des-sha"

set ike gateway "phoenix-gw" ip 10.0.20.1 Main preshare\
"HZ2:8h<O0.SEhM(i-\d^,~XFWf*%{iG^" proposal \
"pre-g2-3des-sha"
```

To provide further illustration of the possible IKE gateway settings, here's a screenshot from the Netscreen GUI's IKE Gateway settings page:

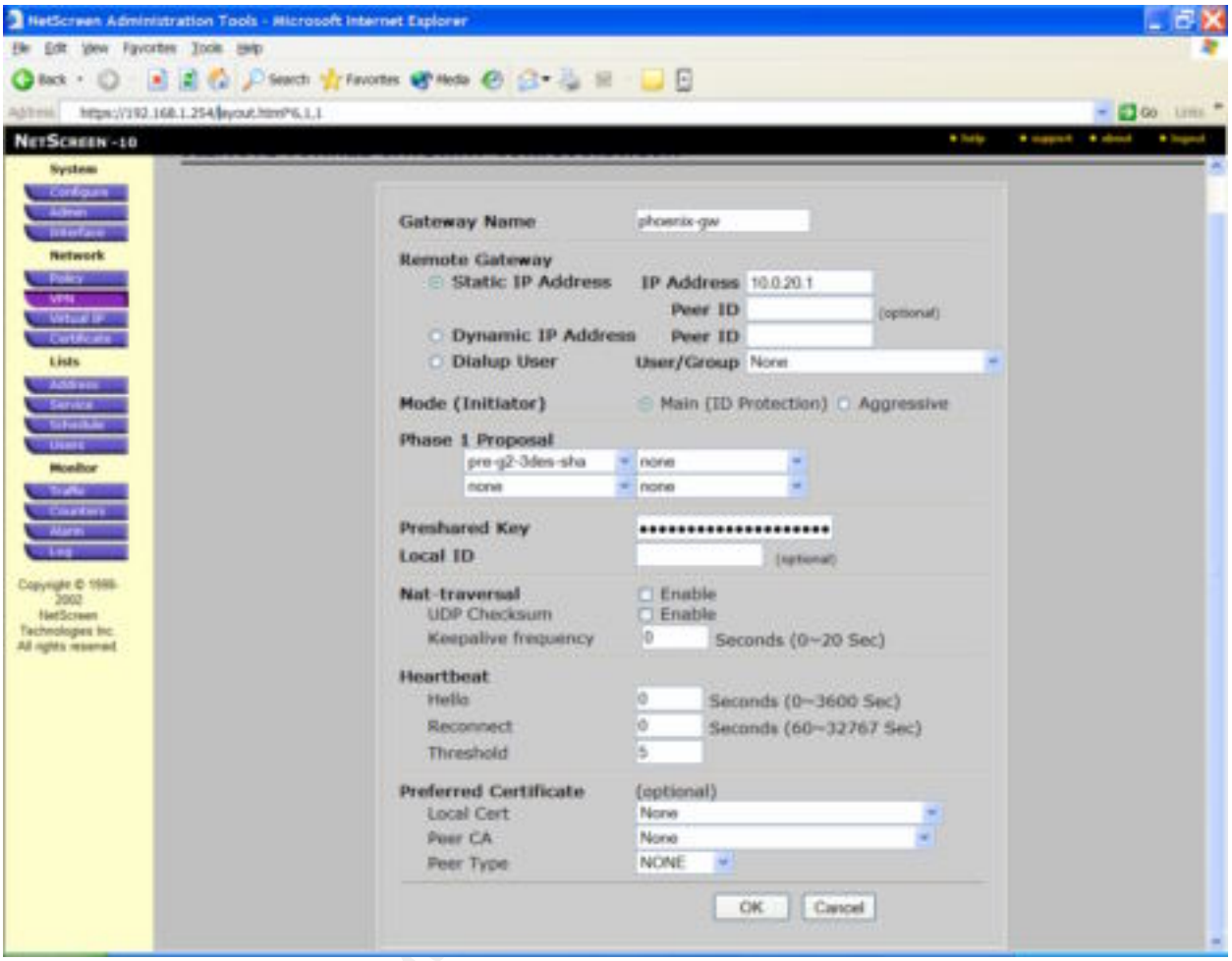

Figure 4

With our gateways configured, the next step is to build the "Autokey IKE Entries", as Netscreen calls them. It's a way to associate a set of Phase 2 proposals with an IKE gateway, as well as define any other properties needed during Phase 2. On the Dayton Netscreen:

```
set vpn "dtn-phnx" gateway "phoenix-gw" no-replay tunnel \
idletime 0 proposal "g2-esp-3des-sha"
```

The set vpn syntax is straightforward. To summarize, "we're going to build a security association with certain properties, using the previously defined gateway, and our phase 2 proposal will be g2-esp-3des-sha". Looking at the example, "dtn-phnx" is a name we've chosen, gateway says to use the IKE gateway named in the next parameter, no-replay indicates that we won't be using replay protection, tunnel indicates tunnel mode, idletime 0 means we don't ever shut down the tunnel from inactivity. Proposal "g2-esp-3des-sha" indicates that the Phase 2 proposal used will be Diffie-Hellman group 2 (g2),

Encapsulated Security Payload (ESP – as opposed to just AH, remember?), Triple-DES (3des), and again, we'll use the SHA-1 hashing mechanism. Additional Autokey IKE entries follow:

On the Dayton Netscreen we add the second entry:

```
set vpn "dtn-arln" gateway "arln-gw" no-replay tunnel \
idletime 0 proposal "g2-esp-3des-sha"
```

On the Phoenix Netscreen:

```
set vpn "phnx-dtn" gateway "dayton-gw" no-replay tunnel \
idletime 0 proposal "g2-esp-3des-sha"

set vpn "phnx-arln" gateway "arln-gw" no-replay tunnel \
idletime 0 proposal "g2-esp-3des-sha"
```

On the Arlington Netscreen:

```
set vpn "arln-phnx" gateway "phoenix-gw" no-replay tunnel \
idletime 0 proposal "g2-esp-3des-sha"

set vpn "arln-dtn" gateway "dayton-gw" no-replay tunnel \
idletime 0 proposal "g2-esp-3des-sha"
```

As before, here's an Autokey IKE entry page from the Netscreen's GUI:

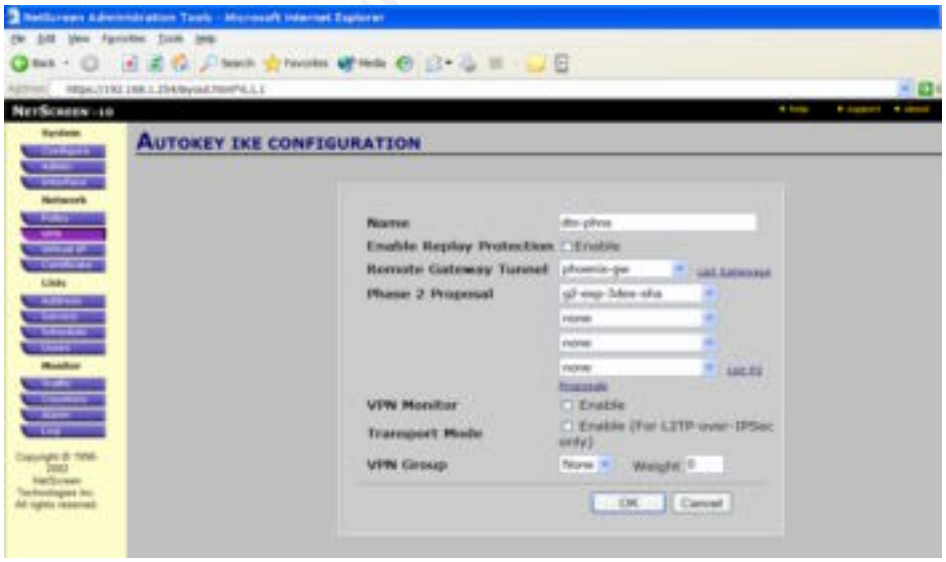

Figure 5

So we've created everything we need to describe our VPNs, all that's left is to implement them in a policy. Netcreen's policy rules are like most other firewalls, they describe an action to taken when they encounter traffic that matches the policy. On the

Netscreen-10 they divide policies into four categories, Inbound, Outbound, From-DMZ and To-DMZ. We're going to ignore the DMZ policies, they're not necessary for our implementation. Policies are parsed from the top down, so the first rule that matches is the action taken. If your first Outbound rule is to allow HTTP from any source IP to any destination IP, and your second rule is to block HTTP from inside users to say, www.badguys.com, then the second rule will be ineffective. The first rule already allowed the packets through! This point is absolutely critical to understand, both for correctly implementing firewall policies, and for building VPN policies on a Netscreen that actually work.

Basically, on each firewall, we need to create four VPN policies. Two pair for each office – Netscreens require an "in" and an "out" policy with the VPN information. Also, the VPN policies MUST be at the top of the policy listing or they won't be used correctly. On the Dayton firewall we execute the following commands:

```
set policy top outgoing "Inside Any" "Phoenix-192.168.0/24" \
"ANY" Tunnel vpn "dtn-phnx" log count
set policy top incoming "Phoenix-192.168.0/24" "Inside Any" \
"ANY" Tunnel vpn "dtn-phnx" log count
```

Set policy syntax is as follows: "top" places the rule at the top of the list, "outgoing" is the policy category of our rule, "Inside Any" is the source address, in this case the network on the "trusted" side of the Dayton firewall, or 192.168.1.0/24. "Phoenix-192.168.0/24" is the destination address, "ANY" is the service (in this case all TCP and UDP services, and ICMP). Tunnel is the action to be taken on traffic that matches the policy, the other two choices are Permit and Deny. Since we chose tunnel, the Netscreen needs to know which VPN to send the traffic over, and we tell it to use "dtn-phnx", or the Dayton->phoenix tunnel. Log and count are options to log each packet that matches this rule, and to include them in traffic counts. Notice that I had to make both an outgoing and an incoming policy. So we need to finish up on the Dayton firewall:

```
set policy top outgoing "Inside Any" "Arling-192.168.4/23"\
"ANY" Tunnel vpn "dtn-arln" log count
set policy top incoming "Arling-192.168.4/23" "Inside Any"\
"ANY" Tunnel vpn "dtn-arln" log count
```

On the Phoenix firewall:

```
set policy top outgoing "Inside Any" "Dayton-192.168.1/24" \
"ANY" Tunnel vpn "phnx-dtn" log count
set policy top outgoing "Inside Any" "Arling-192.168.4/23" \
"ANY" Tunnel vpn "phnx-arln" log count
```

And on the Arlington firewall:

```
set policy top outgoing "Inside Any" "Dayton-192.168.1/24" \
"ANY" Tunnel vpn "arln-dtn" log count
```

```
set policy top outgoing "Inside Any" "Arling-192.168.4/23" \
"ANY" Tunnel vpn "arln-phnx" log count
```

To verify that these made the policy list correctly, we'll check using the GUI.
Since our firewalls are in operation and I've changed all the names, addresses, and some methods to protect the innocent, I'll show you a fuzzed out and modified version of an operational Netscreen-10's policy listing with correctly configured VPN policies:

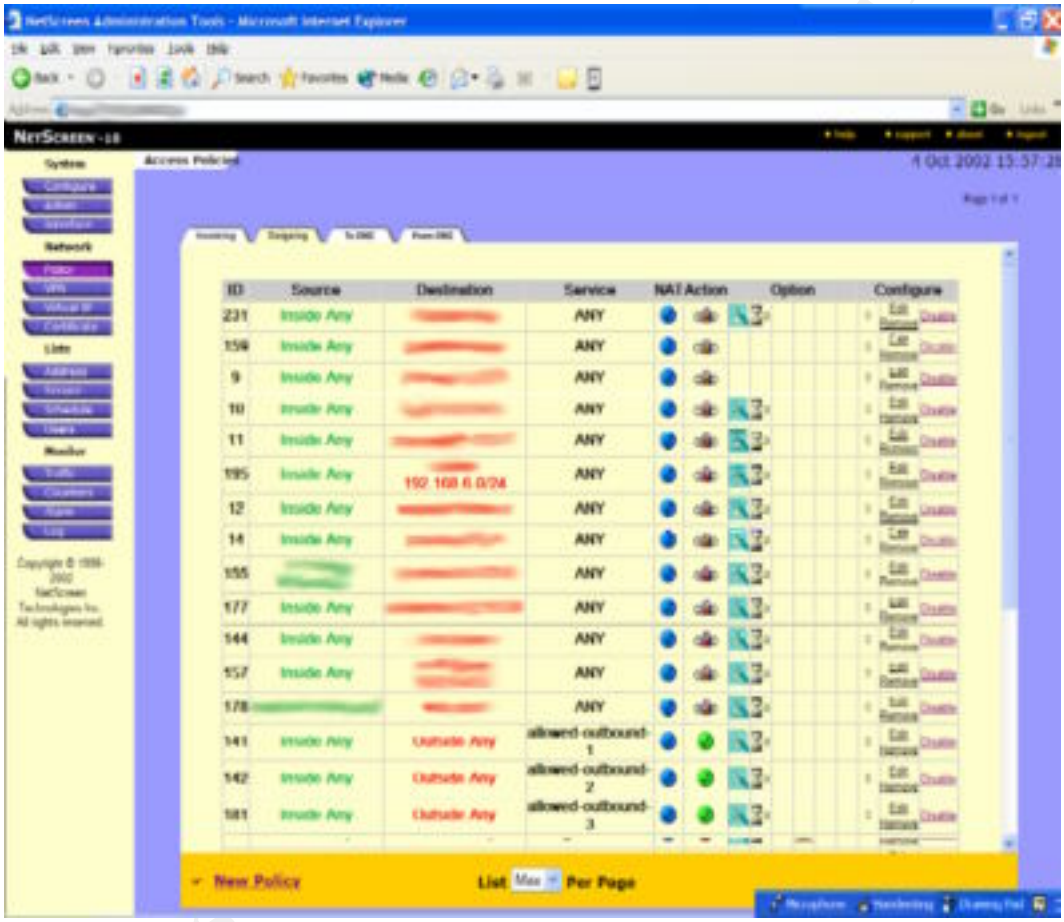

Figure 6

The Action for the first few rules is depicted as a lock with two arrows, indicating a rule with an action of "Tunnel". This picture was chopped up significantly to eliminate any specific information on live firewalls. After eyeballing the configuration on each of the three routers, we issued a "save" command in each system's command line, and shipped the firewalls to their respective offices.

Next we tested. The following command issued on the Dayton router verified connectivity to the Phoenix trusted network, or 192.168.0.0/254:

```
dayton-gw-> exec ping 192.168.0.254 from trust
Type escape sequence to abort
```

```
Sending 5, 100-byte ICMP Echos to 192.168.0.254, timeout is 2
seconds from trust
ip 192.168.1.254
!!!!!
Success Rate is 100 percent (5/5), round-trip time
min/avg/max=105/117/131 ms
```

When using Netscreen's ping tool, if you don't specify the source interface using the from directive, it'll use the untrusted interface's IP address, making you think the tunnel's down when you're actually using an IP that isn't going to transit the VPN! We ran similar commands to verify actually connectivity. Next we used FTP to transfer large files between offices to ensure we didn't have any fragmentation issues, which often pop up with IPSEC usage.

The last stage of the implementation was to discuss lessons learned. Most of the technical lessons learned were mentioned in this paper. VPN policies must live at the top of the policy listing. Ping doesn't work exactly like you may think it will. We didn't run into an enormous amount of pitfalls because we spent 95% of the time on this project in learning the technology, planning the implementation, and testing our ideas on pre-production networks. The actual implementation of 6 offices, with hot one-by-one cutovers from frame-relay took no more than one hour per office/firewall. The VPN layer was as invisible to the user population as the previous Frame-Relay WAN had been.

Summary

As alluded to previously, the goals of the transition to VPN were improve WAN speeds, reduce data communication costs, eliminate single points of failure, and to maintain or improve the confidentiality, integrity, authentication, and non-repudiation of our inter-office traffic. This effort was a success on all counts. Inter-office network speeds went from 256kb/sec to 1.5Mb/sec. Costs dove from $9000/month to $5500/month, a savings of $42,000/year. The failure of one firewall, router, CSU/DSU, T1 will only impact that office, not the entire company. And while private frame relay claims to offer a modicum of privacy by partitioning customer traffic within Private Virtual Circuits (PVCs), it offers no integrity, authentication or non-repudiation. By implementing our own controls we now know that we've covered the bases of information assurance.

We did add to the complexity of the security infrastructure. IT staff had to be trained on Netscreen and VPN basics – how to add services and policies, how to configure and test VPNs, and how to bring them back to life when they hiccup. A major advantage to our meshed configuration is that if one tunnel goes down for any reason, staff can "bounce" to any other office in the mesh and access the firewalls at both ends of the "down" tunnels. Policy management was another new challenge, as we now had 6 firewalls to maintain. We've handled that added complexity by running any policy changes through the IT Director for approval before implementation, because over time the policies for each office diverged as each system admin added local policies to get

their office's work done. A funny side effect of this implementation is that our VPNs are actually far more reliable than our Frame Relay service had been, mostly because we're using Internet service from a more reliable provider with a fast backbone, rather than an oversold, overly congested Frame network. We averaged three outages a month of over one hour before – now we average one such outage every two months, and it's nearly always an ISP issue, not our VPNs.

Sources:

InetDaemon Enterprises, "CLASSLESS INTER-DOMAIN ROUTING NOTATION (CIDR NOTATION)", http://www.inetdaemon.com/tutorials/internet/routing/bgp/cidr_notation.html (5 October 2002)

Rivest, R. "Request for Comments 1321." April 1992. http://www.ietf.org/rfc/rfc1321.txt (4 October 2002)

National Institute of Standards and Technology. "Federal Information Processing Standards Publication 180-2." 1 August 2002. http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf (4 October 2002)

Computer Security Division, National Institute of Standards and Technology. "Advanced Encryption Standard Fact Sheet." 28 January 2002. http://csrc.nist.gov/encryption/aes/aesfact.html (4 October 2002)

National Security Agency, The. "Router Security Configuration Guide." 25 March 2002. http://nsa2.www.conxion.com/cisco/guides/cis-2.pdf (4 Oct 2002).

Killalea, T. "Best Current Practice 46." Nov 2000. http://www.faqs.org/rfcs/bcp/bcp46.html (4 October 2002)