Global Information Assurance Certification Paper

OpenVPN and the SSL VPN Revolution

Charlie Hosner 8.8.2004

GSEC v.1.4b

TABLE OF CONTENTS

Abstract

True SSL VPNs are beginning to appear in the market. One of the best, and definitely the least expensive, is the open source SSL VPN, OpenVPN.

IPSec VPNs are either too expensive or too difficult to use securely. IPSec is dense and contains too many options to be configured and administered securely by non-expert personnel. It also operates in kernel space providing the opportunity for catastrophic failure. OpenVPN rejects the complexity of IPSec by using the battle tested SSL/TLS protocol and cryptographic libraries to provide equal or better function in a simpler package. OpenVPN also operates in user-space increasing security and stability.

Many of the products that claim to be SSL VPNs are actually just SSL gateways operating under the guise of a true VPN. Many of these products open the unsuspecting user to serious security issues. OpenVPN is the first real SSL VPN to provide the same function and security as its IPSec predecessors.

Introduction

> "IPSec VPNs protect IP packets exchanged between remote networks or hosts and an IPSec gateway located at the edge of your private network. SSL VPN products protect application streams from remote users to an SSL gateway. In other words, IPSec connects hosts to entire private networks, while SSL VPNs connect users to services and applications inside those networks."[Phi03]

The above statement is totally wrong. The myth that ***Secure Socket Layer*** (SSL) ***Virtual Private Network*** devices (VPNs) are used to connect applications together is not true. The commercial SSL VPN market has falsely labored under this misdirected paradigm, but it is a mishandling of terms and represents an untrue statement. This document covers the emerging trend of SSL based VPNs. It is important to be absolutely clear that when this document refers to a VPN, it is not referring to an application level access to a remote network's application. A VPN is a site-to-site tunnel. Let me say that one more time, a VPN is a site-to-site tunnel. There is a terrible misunderstanding in the industry right now that pigeon-holes SSL VPNs into the same category with SSL enabled web servers and proxy servers. People hear SSL and immediately think of a protocol that encrypts traffic for an application, or for several applications, one at a time via proxying, application translation, or port forwarding. This is NOT a VPN. It is an application level gateway, a firewall, or an SSL gateway, but it is not a VPN. A VPN, or Virtual Private Network, refers to simulating a private

network over the public Internet by encrypting communications between the two private end-points. This provides the same connectivity and privacy you would find on a typical local private network. A VPN device is used to create an encrypted, non-application oriented tunnel between two machines that allows these machines or the networks they service to exchange a wide range of traffic regardless of application or protocol. This exchange is not done on an application by application basis. It is done on the entire link between the two machines or networks and arbitrary traffic may be passed over it. See the section on other SSL VPNs at the bottom of this document for more information on this issue.

In the past, the method for creating such a site-to-site tunnel was to use the ***Internet Protocol Security*** (IPSec) standard. IPSec was not chosen due to its great strength as a protocol. It was chosen because it was the only game in town. IPSec has received much criticism for its unnecessary complexity and tight coupling with the OS kernel [SF99], but due to its monopoly on function, it has enjoyed widespread implementation.

Enter OpenVPN. OpenVPN is a user-space SSL-based VPN that illustrates the ease of use and simplicity of SSL VPNs while providing protection and function equivalent, and in some cases superior, to IPSec. OpenVPN does away with the complexities of IPSec from an installation, configuration, and management perspective. Security's worst enemy is complexity and OpenVPN defeats this enemy. Unlike IPSec, OpenVPN holds true to the secure ***OS Ring Architecture*** philosophy of non-interference with kernel space or keeping applications out of Ring 0, which we will discuss more shortly. Adherence to this philosophy gives OpenVPN the ability to operate more safely today and provide greater protection against unknown attacks of tomorrow.

> Note: The IETF has taken over development and management duties for SSL and have renamed it ***Transport Layer Security*** (TLS). For the rest of this document you may see it referred to as SSL, TLS, or SSL/TLS. Unless otherwise noted, all of these refer to the latest version of TLS.

SSL/TLS is the most widely deployed security protocol in the world [Res01]. As such, it has undergone extensive scrutiny and has yet to be degraded by any known weakness. This does not mean it is guaranteed secure for the future, but it does mean that many of the brightest minds in cryptography and mathematics have been unable to find any holes in its cryptographic armor. In the past, SSL/TLS was a general protocol that would be tightly coupled with specific applications, thus the extreme confusion about what an SSL VPN really is. It would be used to secure session communication between two hosts using a single application or protocol at a time. The most well known use of SSL is in the HTTPS protocol to enable secure web-based ecommerce. SSL is the default

security solution for application to application needs, but it has never been implemented to handle arbitrary multiple protocols at the same time, until now.

Before jumping into OpenVPN, we need to cover a couple general issues on Cryptography, VPNs and IPSec.

Quick Intro to Cryptography

In order to talk about VPNs we must know a little bit about cryptography. VPNs rely heavily on cryptography to maintain a tunnel between end points and to securely build this tunnel. Cryptography is very complex and easy to do wrong, so its a good thing there are products like OpenVPN that have it already implemented for us.

There are four ***cryptographic primitives*** that relate to our discussion on VPNs: symmetric ciphers, asymmetric ciphers, message digests, and digital signatures. There are also four goals we have with information security: ***Confidentiality***, ***Integrity***, ***Authentication***, and ***Non-repudiation***. The trick is to assemble our four primitives to achieve our four goals.

Symmetric Ciphers – Confidentiality

In order to keep our data secure from prying eyes, we must encrypt it. ***Symmetric encryption*** uses a very fast block level algorithm to encrypt and decrypt data and is the primary primitive used to protect data confidentiality. Both sides of the tunnel will use the same encrypt/decrypt key which presents us with the primary weakness of symmetric ciphers, ***key distribution***. Common symmetric ciphers are DES, 3DES, Blowfish, AES (Rijndael), RC5, RC6, Serpent, and IDEA.

Message Digests – Integrity

With VPNs we are sending our sensitive data over the public Internet. This uncontrolled network subjects our data to all sorts of malicious and accidental tampering and modification. We want to make sure what we send is the same as what the other side receives, and vice versa. To maintain this integrity, we use ***message digests***. A message digest is an irreversible mathematical function that takes a message of any size and encodes it as a fixed length block of cipher text. This fixed length cipher is called the digest. It is essentially a cryptographic "summary" of the message. Every message has only one digest and ideally, no two messages should ever create the same digest. If even one letter of our message is changed, the entire message digest will be different.

Before we send our message, we run it through a message digest function and get our fixed length block of cipher text. We then send this cipher text along with

the message. When the other side of our communication receives our message, they will run the same message digest function on the text of our data and compare the result to our attached message digest. If they are the same, the receiver knows the message has not been changed since we sent it.

If we add a key to our message before running the message digest we get even better protection. We will discuss this later under the HMAC section below. Commonly used message digest algorithms are MD5 and SHA-1.

Asymmetric Ciphers – Everything Else

We have two goals left to cover, authenticity and non-repudiation. We want to guarantee that the entity we are talking to is the entity we think we are talking to. To authenticate this fact we use ***asymmetric encryption***, or ***public key cryptography***. This involves the creation of a key pair. These two keys are mathematically related is a very useful way. Data encrypted with one key can only be decrypted with the other key in the pair, and vice versa. One key is labeled the public key and it is distributed to the world. The other key is the private key and it is kept secret. We can use this system to authenticate the entity by checking that it has something that no other entity should have, its private key. In order to check this, we have the entity send us a message encrypted with this private key. Since the entities public key is available to the world, we can use it to decrypt the message. If this works, we know the entity is who they claim to be. This gets a bit more complex below.

We also want to make sure that everyone is held accountable. In order to hold entities accountable we need to make it impossible for someone to send traffic and later claim that they did not, non-repudiation. Again, since the only person who knows an entity's private key is the entity itself, we can use this to gain non-repudiation. Just as in the above case, if an entity encrypts its message with its private key, we can decrypt the message using the public key and assure that the sender is the only entity that could have sent the message, meaning they can not later claim that someone else forged it.

In actual practice, we use ***digital signatures***. When an entity needs to send a message, they will run a message digest for it. They will then digitally sign the message digest by encrypting it with their private key. The whole package is bundled up and run through symmetric encryption for confidentiality. This gets sent to the other end of our communication tunnel where the symmetric encryption is decrypted. The receiver then decrypts the message digest using the sender's public key. If it works, we have authentication and non-repudiation. The receiver then runs a message digest and compares it to the one it received. If they match, the receiver knows the data has not been altered, thus we have integrity. The most commonly used asymmetric algorithm is RSA.

We will talk about each of the above primitives in much greater detail as we go.

VPN in a Nutshell

VPN stands for Virtual Private Network. VPN is the term used to refer to any device that is capable of creating a semi-permanent encrypted tunnel over the public network between two private machines or networks to pass non-protocol specific, or arbitrary, traffic. This tunnel can carry all forms of traffic between these two machines meaning it is encrypting on a link basis, not on a per application basis. VPNs are useful in situations where an entity is paying for dedicated leased lines due to security concerns or the need to provide layer two communications over a WAN link via ***transparent bridging***, ***WINS servers***, or other ***broadcast repeaters***. The VPN allows the end points to connect to the Internet and have this same functionality without the need for expensive leased lines. The other common use for VPNs is to provide dial-up access or ***network extension*** for remote employees. Instead of making expensive calls and maintaining access servers with modem banks, a remote user can dial up and connect to the Internet locally, then use the VPN to access the main site securely over the Internet. This allows for reduction in phone bills and elimination of expensive and hard to secure modem banks and access servers.

One of the key elements of VPNs is encryption. To protect sensitive or non-routable data as it passes over the public Internet, we need to create a virtual private tunnel. This tunnel is built by encrypting the packets or frames and then encapsulating these in regular IP traffic between the two hosts or networks. The protection and encapsulation of these packets is vital to the function of a VPN and one of the most complex pieces to get right.

What the heck is IPSec?

In November of 1998 the ***Internet Engineering Task Force*** (IETF) came out with a series of ***Request for Comments*** (RFC's) defining the protocols necessary to create VPNs. Specifically, RFC 2401-2412 represent the backbone of the technologies that have come to be known collectively as IPSec. IPSec is a standard set of protocols and rules for their use that allow the creation of VPNs. The theory was if vendors implement IPSec to create their VPN products, they would interoperate with other vendor's products. This has had varying success as IPSec allows for significant latitude in design choices and often leads to IPSec compliant products from different vendors that do not interoperate. Some of the highlights of this series of RFC's are: RFC 2401 (IPSec), RFC 2402 (Authentication Header), RFC 2406 (Encapsulating Security Payload), RFC 2408 (ISAKAMP), and RFC 2409 (IKE). For a comprehensive collection of IPSec related RFC's see Pete Loshin's book *Big Book of IPSec RFC's*.

IPSec creates a secure tunnel by first using a handshake protocol called ***Internet Key Exchange*** (IKE). IKE authenticates the end points of the tunnel to each other, and then follows a secure procedure to exchange the necessary information to create a more permanent tunnel using symmetric encryption. Once this tunnel is in place, any arbitrary traffic sent between these two end points will be passed through the protected tunnel. This tunnel can be used by any application or protocol and is semi-permanent, meaning it will stay up indefinitely provided both end points continue to desire its existence.

IPSec was created by a committee and some believe this process added more functionality, bloat, and complexity than is needed or reasonable. The committee approach has received criticism as a viable way to develop security standards. The preferred method is to use contests like the one used to choose the new Advanced Encryption Standard or AES. As Bruce Schneier and Niels Ferguson put it, "IPSec is too complex to be secure" [SF99]. Be that as it may, IPSec is used to create a majority of the VPN products found today. Checkpoint VPN-1, Cisco PIX, and the open source FreeS/WAN are all examples of commonly used VPN solutions that implement IPSec. So in the past, if you wanted a VPN, you suffered with the complexity of IPSec.

> Note: The FreeS/WAN project is now dead. Its original charter was to secure the Internet using ubiquitous Opportunistic Encryption [Free04]. Failing to make progress in that direction, they closed their doors. The excellent code base they left behind has continued to develop in the form of OpenS/WAN and StrongS/WAN.

In addition to configuration complexity, IPSec has strayed from the secure OS Ring Architecture design principle of non-interference with kernel space. This principle breaks out the OS into rings of privilege. Ring0 is reserved for the kernel and other essential processes. Ring1 for other system processes that need low level access to hardware. As you move outward in rings, the privilege of the process is decreased. Ring3 is where most user processes are found. The architecture rules state that processes in higher numbered rings can not interfere with processes in lower numbered rings. This provides greatly enhanced stability and security in our applications and allows for multi-user, multithreaded systems.

> "The part of the OS that needs to access the hardware and provides the basic metaphors of processes, memory and devices, run in ring0, some system tasks run in ring 1 etc... The normal user processes run in the ring with the lowest privileges. This means a process running in a certain ring cannot harm the processes in a ring with more privilege. Multics was the OS that brought this idea to us, and

> formed the base for all later operating systems up to now. This architecture offers …. a lot more stability and security than the earlier architectures, and is able to provide multitasking and multi-user facilities." [Dum97]

To reduce the impact of application failure on the stability and security of the system, non-essential processes should not interfere with the kernel. In order to gain the level of control needed to secure traffic over the interface link, IPSec needs to be tightly integrated into the OS kernel, in Ring0. This violates our design principle and puts the entire operating system at risk. This violation also makes installation difficult and puts up road blocks to developing client and server applications for other platforms.

Anyone who has installed FreeS/WAN on Linux understands the degree of coupling necessary under IPSec. Having to install touchy, kernel specific code hacks can definitely be discouraging, especially for security conscious administrators who upgrade their kernels on a regular basis. Additionally, even though IPSec is touted to be interoperable between vendors; the reality is if you have a vendor's VPN product on one side of the tunnel, you often need to use the same vendor's client or server on the other end. This reduces the flexibility of many products as they don't make clients for Windows or have a hard time installing with the existing Windows IPSec VPN client. This issue of variation in implementation results in many headaches that eliminate the benefit of using an open standard in the first place.

So how do these things work?

VPNs work by creating a virtual tunnel over the public Internet. In order to create this tunnel, symmetric encryption is used. Both sides of the tunnel share common encryption and decryption keys and use them to encrypt all traffic in both directions. Symmetric encryption is very fast and there are many solid algorithms available to implement this (Blowfish, AES, 3DES). There are two problems with symmetric encryption. First, how do we get these common keys to both sides of the tunnel? This is called ***key exchange*** or ***key agreement***. Second, how do we know we are exchanging keys with the correct entity? This is called authentication.

There are many ways to exchange keys, some elegant and some barbaric. One way to exchange keys is to call the administrator on the other end of the tunnel and read them the key over the phone. Another way is to send them the key in an email using ***Pretty Good Privacy*** (PGP) to encrypt the exchange. Both of these methods will work, but they are not very effective. This is referred to as a pre-shared secret and it does not scale well or provide us with perfect forward secrecy, which we will talk more about in a minute.

A foundation of solid cryptography is that you change your encryption keys on a "regular" basis. The definition of "regular" is pretty broad. I have seen philosophies that say the lifespan of a key should be less than the time it takes to break that key. The literal interpretation of this strikes me as kind of silly. Imagine an attacker had a system that could break a DES key in 1 hour (not that far from reality). If you change your DES key every hour, all this means is your attacker needs to archive your traffic and get to work breaking it. They will begin seeing unencrypted traffic one hour after that traffic is sent, so all you've really done is add a one hour delay to the compromise of your data. I feel the true spirit of this philosophy is to change your keys as often as you can without putting an unreasonable resource load on your system or administrators. This frequent change also provides what is called ***Perfect Forward Secrecy*** meaning if your key is broken for one series of transactions, it does not compromise any future series.

If you want to change your keys once an hour, or even once a day, you can see how the phone call or PGP method is not really practical. Especially if you have 80 VPN users with whom you need to exchange keys.

To overcome this cumbersome key exchange issue, VPNs often use ***certificates***. Certificates use Public Key Cryptography, meaning a host generates a public and private key pair that are mathematically related to one another. Any data encrypted with the public key can only be decrypted with the private key, and vice versa. Each end system has its own public/private key pair. The public key is given out to the world to encrypt traffic bound for the system, and the private key is kept secret to decrypt this traffic. The private key can also be used to prove that data was actually sent by a specific entity, which is called non-repudiation. If I encrypt something with my private key you can confirm it is really me by decrypting it with my public key. The problem with this is I will need a copy of every host's public key that I want to connect with. If I have 100 hosts I'm keeping VPN connections with, this again becomes a scalability problem.

The solution is to use a ***certificate authority*** (CA). A certificate authority looks over an entity's credentials and certifies that they are who they say they are. Once an entity is certified, the certificate authority will sign the entity's public key with the CA's private key. Now, in order to prove that your entity is really the entity you want to talk to, you just need to prove that they have been approved by your CA. We essentially are saying "We trust the CA and anyone the CA trusts we will trust too". To prove that our CA trusts this entity all we need is the CA's public key. When you get a certificate from the entity, it should have a signature created by the CA's private key. You use the CA's public key to decrypt this signature to make sure the certificate is valid. Now you can have 100 hosts who have all been preapproved by your CA. You can authenticate these hosts by checking the CA signature on their certificates with the CA's public key, and only

need to keep one key on your system, the CA's public key. This solves our scalability problem.

SSL/TLS to the Rescue.

The new kid in town is the user-space SSL/TLS based VPN. SSL has been in existence since the early 90's. SSL was initially developed by Netscape and was eventually joined by another similar code branch created by Microsoft. In the late 90's the IETF created TLS is an attempt to consolidate the different SSL branches into a common, open standard. TLS is essentially SSLv3 with some minor fixes and enhancements.

User-space SSL VPNs use the highly mature and widespread SSL/TLS protocol to handle the tunnel creation and cryptographic elements necessary to create a VPN. We are going to focus mostly on an open source SSL VPN, OpenVPN. There are other commercial products available to create SSL VPNs, but most if not all of them miss the mark on creating a usable site-to-site VPN. For a detailed explanation of this see the section below on other SSL VPNs.

OpenVPN is a user-space VPN that uses the well tested and mature SSL/TLS infrastructure to create the same site-to-site connection functionality found in IPSec VPNs. OpenVPN is referred to as a user-space VPN because it does not require sophisticated intertwining with the OS's kernel to function. It operates in Ring3 of our secure OS Ring Architecture, which is right where we want it. Usually, in order to do link encryption, an application must be intertwined with the kernel to provide low level access to the interface where the link is found. User-space VPNs use a "virtual interface" they control and access without this kernel dependence. This gives user-space VPNs a more secure starting point than standard IPSec devices, as well as provided more flexibility in porting to other operating systems and ease of installation and maintenance. The flexibility of this architecture even allows it to exist on the same box with IPSec VPNs. You can install OpenVPN on Windows machines without any conflicts between it and the Windows IPSec client which, as anyone who has tried to install a third party IPSec client on Windows knows is a pretty big plus. In fact, you can run an IPSec VPN from your Windows machine, and still have an SSL/TLS based VPN running at the same time.

SSL/TLS is a standard protocol for encrypting Internet traffic. It is very mature and has been widely implemented and tested for vulnerabilities. As long as no one figures out how to factor large prime numbers in a hurry, SSL/TLS appears to be in good shape to provide security for quite some time to come. SSL/TLS is much easier to implement than IPSec and provides a platform that is solid, simple, and well-tested.

It is important to note that SSL/TLS based VPNs are able to encrypt link traffic for site-to-site connectivity just like IPSec VPNs. The RSA handshake (or DH) is used exactly as IKE in IPSec, and the SSL crypto library is used to secure the symmetric tunnel after that, again using similar encryption techniques to those protecting IPSec tunnels. This tunnel can pass arbitrary traffic, just like an IPSec VPN. No restrictions, no tricks.

> Note: One downside to SSL/TLS is in packet drop performance. IPSec will inspect and drop a packet at a lower level in the protocol stack than SSL/TLS which will take it higher and process it more before rejecting it. This could be an issue with DoS attacks and some very high capacity usage scenarios. In most cases this is not a problem.

OpenVPN Installation

OpenVPN is built with portability in mind and currently runs on most OS's including windows 2000/XP, Linux, Solaris, BSD, and Mac OS X. Since it runs in user-space instead of as a kernel module, installation is a breeze. There are highly detailed installation documents available on the OpenVPN website. I will not repeat those instructions here, but will give a quick over view and touch on areas I feel are important to highlight. For comprehensive, step-by-step instructions please see the Source Forge OpenVPN project link in the work cited section [Yon04].

On Windows, OpenVPN installs just like any other program. It comes bundled up as an executable and all you need to do it double click on the installer. It's that simple. You will still need to see the next section for configuration issues, but for the most part installation is complete with this single step. To automate the starting and stopping of OpenVPN at reboot, you will need to run OpenVPN as a Windows service. This is also very easy to do following the simple instructions on the OpenVPN site. Total installation of the Windows client takes about 10 minutes including configuration. For anyone who has tried to configure the built-in Windows IPSec client that should be impressive. For people who have tried to install and configure third party IPSec clients, that number should be shocking!

On Linux, installation is just about as simple. Most distributions have OpenVPN as part of their package system. Gentoo has an OpenVPN ***ebuild*** and Redhat has ***RPM's*** available. OpenVPN uses the ***TAP*** and ***TUN virtual drivers***. If you are using Linux kernel 2.4.x or greater, and you should be, these drivers are already bundled with your kernel. If you are using Linux kernels earlier than 2.4.x, shame on you, but you can still download and install the TAP/TUN drivers quite easily. You can also install OpenVPN from source which isn't much more

difficult than installing from one of the various package systems. Again, the complete documentation is available on the OpenVPN site [Yon04].

OpenVPN has a pretty long list of installation options, but the only one I found really essential is the --enable-pthreads option. This option is very important as it allows for multithreading to create a different control channel over which new key exchanges are done. The default rekeying period is one hour, so pthreads allows you to eliminate hourly latency by rekeying over a separate channel and switching over to the new keying material seamlessly.

Those concerned about bandwidth may also want to look into the LZO compression library which compresses data before it is encrypted. As I'm sure we all know it is important to view with suspicion any product that says it compresses encrypted data. Compression works by identifying common patterns in our data and replacing them with smaller place holders. One of the characteristics of a quality encryption algorithm is a flat histogram, meaning the encrypted text does not have any common patterns and thus can not be compressed.

OpenVPN Configuration

OpenVPN is configured like most UNIX services using a config file. One of the blessings of OpenVPN is the fact that the config file format is almost exactly the same for all platforms. There are a couple minor differences, but for the most part, the config files are portable. This is a really important feature considering the rather dramatic differences found between Windows and the various *NIX variants.

OpenVPN tunnels traffic over UDP port 5000. As of the 2.0 release, multiple connections will use the same UDP port on the server, as opposed to 1.6 and earlier which required one UDP port per connection. If you are wondering why UDP is used instead of TCP, there are problems when you tunnel TCP over TCP. TCP keeps track of packet sequence and packet loss and requests that missing packets be resent, which is a good thing when you only have one layer of TCP. It also has adaptive timers that dictate how long it will wait before it requests resends. This interval changes and basically increases exponentially as failures to receive packets continue. If you have TCP riding on top of TCP, you now have two flow control layers that are each providing timers and resend requests. If things line up poorly, for instances the "lower TCP layer" has a longer interval than your "higher layer" you can get a build up of requests from above that cause an internal meltdown in your flow control system. You end up slowing your TCP connection down to a crawl as redundant layers of flow control work against each other in an attempt to get packets resent.

OpenVPN works in two modes. Either it uses the TUN driver to pass IP traffic or it uses the TAP driver to pass Ethernet traffic. I found it easiest to use the TUN driver and set up a WINS server on the other end to handle layer two broadcasts, so that is the configuration I am going to focus on. Configuring the TUN driver is very easy and only requires a couple of commands and a one line entry into modules.conf that is probably already there. Again, excellent instructions are found on the Source Forge site.

Once the TUN interface is set, it's smooth sailing. OpenVPN uses a config file that is very easy to work with. Example config files and suggested changes are available on the website. There are a couple changes you will want to make to get the most out of the security and performance OpenVPN provides.

User nobody

An essential item to a security conscious administrator is the user the OpenVPN daemon runs under. The config file has this set of options:

```
# Downgrade UID and GID to
# "nobody" after initialization
# for extra security.
;user nobody
;group nobody
```

It is imperative that you uncomment these bottom two lines. This will cause OpenVPN to initialize, then downgrade to operate as user and group "nobody". For those of you unfamiliar with UNIX, "nobody" is an unprivileged user with just enough permission to operate OpenVPN, but not enough to access much else. If an attacker somehow finds a vulnerability that breaks OpenVPN, the best they will be able to achieve is the permissions of the user OpenVPN is running under. If this user is "nobody", the attacker will have extremely limited ability to do further damage to the machine. At very least, this will slow intruders down and give you more time to detect the intrusion.

> Note: The user "nobody" option does not make sense on Windows machines unless you first create a user and group "nobody". In UNIX, these accounts already exist and are often used by other services in a similar way.

chroot the server

If you're using a UNIX variant, right after the user "nobody" lines, you should add the following option:

`chroot /usr/local/openvpn` {directory you want to "jail" OpenVPN to)

This option will lock the OpenVPN process into the OpenVPN directory (or whatever you specify) and not allow it to access the rest of the system. This provides another layer of defense against any future vulnerability that allows compromise of the OpenVPN daemon. Use chroot in combination with user "nobody" and you provide a level of proactive protection against many unknown attacks that could appear in the future.

TLS-auth

And if the above options aren't enough to make you feel secure, OpenVPN includes the tls-auth option. With the tls-auth option enabled, OpenVPN will use a second level of authentication by creating an HMAC key for use in the TLS handshake process. This feature does incorporate some administrative overhead as all connecting machines must have this extra pre-shared secret, but it provides a high level of protection against attacks like the buffer overflows we found last year in OpenSSL. With tls-auth enabled, an attacker scanning the Internet for SSL enabled devices will not even be able to initiate a TLS handshake without the proper HMAC signature.

Adjust the MTU

The expected tunnel size and the actual tunnel size on either end of the connection may be different. This will give you an error message in your log files telling you that the actual remote options do not match the expected remote options. When you read this message closely you will see that the link-mtu or the tun-mtu do not match on both sides. This is caused by the headers increasing packet size to larger than expected and can cause fragmentation and performance degradation. This one can be tricky to find as OpenVPN will still run correctly and not let you know there is a problem, your only clue will be decreased performance which you may only discover under extreme load. You must look in your logs to find this. The suggested fix is to use the tun-mtu options and the mss-fix options in the configuration file. I suggest

```
tun-mtu 1500
mss-fix 1400
```

These values should eliminate the mismatch errors and fragmentation. You may need to fiddle with the values a bit to get it working perfectly on your implementation. Season to taste.

Route

Just a quick comment on the route command. Since you are connecting to a remote network you will probably need to use the route command to get traffic over the VPN tunnel. You will most likely have something like 192.168.1.x on

one side of the tunnel and 192.168.2.x on the other side. This means you need to tell your routing table to use the VPN's TUN/TAP device to access this private network. Windows and UNIX use route just a little bit differently so when you make your config files you will need to make sure you use the correct format. This is one of the few platform specific changes you will need to make to your config files. As a quick example,

```
route add –net 10.1.0.0/24 gw 10.2.0.1
```
format for UNIX

```
route 10.1.0.0 255.255.255.0 10.2.0.1
```
format for Windows

OpenVPN Features

When comparing VPN products, there are several items that most people look for. OpenVPN excels at many of these points. We will talk about OpenVPN's security model and features in the next section so for now, let's just assume we are talking about products that provide similar security.

Throughput/Performance

VPNs require encryption/decryption of traffic and that takes CPU cycles. One of the important measures of a VPN is its throughput or the amount of data is can pass before it is unable to keep up with the decrypt/encrypt activities. With hardware VPNs this is an easy number to find, but with software products like OpenVPN, your throughput will depend a lot on your hardware. For this document, OpenVPN was tested with a Pentium III 1Ghz machine with 512K RAM running Gentoo Linux. The other end of the tunnel was a Pentium IV 2.7 GHz machine running Windows XP. The link between these two machines max's out at 3 Mbps and OpenVPN was able to keep up with this load without any degradation in throughput. The processor loads on both sides were miniscule and while one should not expect OpenVPN to scale linearly, it should handle enough throughput to service most small to medium-sized implementations, and with load balancing or more serious hardware, it could handle many larger implementations as well. Additionally, there is the very real possibility that OpenVPN can benefit from the myriad of hardware SSL accelerator cards out there as it is using the standard SSL/TLS functions. (Check the OpenVPN user mailing list for more information). OpenVPN does not have a hard limit to the number of tunnels it can sustain.

NAT Traversal

One of the serious drawbacks to IPSec VPNs is their inability to function behind a device that does NAT. The ***Authentication Header*** (AH) mode in IPSec hashes the source address as part of its authentication process. If NAT changes this source address, as it always does, the VPN on the other end of the tunnel will get

a different hash when it checks the packet integrity and drop the packet thinking it has been tampered with. The solution for this problem in IPSec is to run in tunnel mode using only ***Encapsulating Security Payload*** (ESP). This keeps the source address from being hashed in the packet integrity check. OpenVPN, or more accurately SSL/TLS, does not run authentication on the packet source address so it can successfully traverse a NAT device.

> Note: It is my belief that Authentication Header (AH) was developed to verify and protect the source of a message back in the late 90's when this was more of an issue. I feel this relic has an uncertain place in modern networking. Most addresses are NAT'ed, which accomplishes the same source address masking, and verification of the origin address is of limited value.

X509 Authentication

This refers to using certificates to handle initial authentication as we described above. The alternative to using certificates to handle authentication is to use pre-shared secrets which are described as having problems with key distribution, non-repudiation, and lack of Perfect Forward Secrecy. OpenVPN allows you to use pre-shared secrets, or X509 certificates for authentication, as should all other quality VPN products. The more secure and robust solution is to use certificates.

Ease of Configuration

One of the biggest problems with IPSec VPNs is their complexity. This complexity provides many opportunities for administrators to undermine the security of their devices. Do you want tunnel mode or transport mode? Should you use Authentication Header or Encapsulating Security Payload, or both? The more combinations available that can yield insecure configurations, the more likely one will be chosen. OpenVPN provides strong simple default configurations that fit most implementation needs. OpenVPN can be configured and installed by someone with basic security knowledge while still maintaining a high level of security.

Load Balancing

Load Balancing allows high capacity links to handle large amounts of traffic by sharing the load between several identical servers. This activity is transparent to the end applications and users. OpenVPN does not have built in features to handle load balancing but this can be done quite easily using IPtables. A simple rule can send traffic to a range of addresses in a round robin fashion essentially creating a load balanced environment. The OpenVPN mailing list archives have

specific instructions on creating this type of set up. The 2.0 release of OpenVPN addresses this issue further.

Failover

Another important feature for larger enterprises is failover. When your VPN box dies, can your connections be serviced by another device? Again, OpenVPN does not have built in features to handle this but with a little extra cabling and a floating static route you can create a secondary path to another OpenVPN box and create a poor man's failover system without too much hassle.

Central Management

When we start to talk about really large implementations, centralized management becomes a feature many administrators require. A console that enables monitoring and configuration of multiple VPN devices from a central location is a feature found in some of the large commercial products like the high-end products from Cisco, Checkpoint, or NetScreen. OpenVPN does not offer any such capability, but I'm sure James Yonan would just love it if some programmer out there wanted to contribute to an open source project by designing such a feature.

OpenVPN Security

OpenVPN is built on a solid security foundation. Its core crypto system, SSL/TLS, is the most wide spread system in the industry. It has survived heavy scrutiny without showing any known weaknesses. Properly implemented, SSL/TLS gives the best security currently available. OpenVPN's creator, James Yonan, has done an excellent job of implementing SSL/TLS. But he didn't stop there. OpenVPN also has added features that increase its ability to cope with unknown vulnerabilities that may crop up in either OpenVPN or the SSL/TLS core.

Key Generation

Just a quick note on key usage. Once the SSL/TLS handshake has authenticated both ends, it will generate four different keys; an HMAC send key, an HMAC receive key, an encrypt/decrypt send key, and an encrypt/decrypt receive key. You should never use the same key for more than one security primitive (asymmetric, symmetric, hash, or digital signature) and OpenVPN holds true to this philosophy.

Note: When we are using pre-shared keys in OpenVPN we only have two keys. HMAC send and receive are the same and the encrypt/decrypt keys are the same on each side! This is not ideal. I know this is splitting hairs, but it does violate our one key per primitive rule. To fix this, you can use the --secret option and set the direction parameter like this

`secret extra_key d=1` {where extra_key is the file containing an additional key to be used}

Key Derivation/Exchange

If you remember from the above sections, in order to protect our data, we need to encrypt it with symmetric encryption. To do this we need to have the same symmetric key on each end of the connection. This requires a key derivation/exchange protocol. IPSec uses a system called Internet Key Exchange (IKE) to exchange keys. This system consists of six messages back and forth that eventually lead to the authentication of each system and generation or exchange of symmetric keys. OpenVPN uses the standard RSA/DHE handshake with client authentication to accomplish the same goal. Below is a quick breakdown of the handshake.

Client		**Server**
→→→	Message 1	→→→
→→→	Client Hello	→→→
←←←	Message 2	←←←
←←←	Server Hello	←←←
←←←	Certificate	←←←
←←←	Certificate request	←←←
←←←	Server Hello Done	←←←
→→→	Message 3	→→→
→→→	Certificate	→→→
→→→	Client Key Exchange	→→→
→→→	Certificate Verify	→→→
→→→	Change Cipher Spec	→→→
→→→	HMAC Finish (encrypted)	→→→
←←←	Message 4	←←←
←←←	Change Cipher Spec	←←←
←←←	HMAC Finish (encrypted)	←←←

Let's go through each of these messages and talk about what purpose they serve.

Message 1
The client sends a hello greeting starting the handshake. Included in its greeting is a list of ciphers it supports as well as one of the parameters to the RSA or Diffie-Hellmann key generation.

Message 2
The server chooses a cipher from the client's list and sends it back along with the server's certificate which includes the server's public key digitally signed by our certificate authority's private key. This is an important step. When we get this certificate, we will use our copy of our certificate authority's public key to verify that the signature on the certificate was really created with our certificate authority's private key. If this works, we are assured we are dealing with an authentic end point. Not doing this step leaves us wide open to man in the middle attacks. We also get the server's half of the RSA or Diffie-Hellmann key generation parameter and a request from the server for our certificate.

Message 3
At message three, the client sends over its certificate, also digitally signed with the certificate authority's private key. The server will use its copy of our certificate authority's public key to verify this certificate's authenticity. The client also generates and sends what is called a pre-master secret; this is called the Client Key Exchange. This is the pivotal step in the handshake and is the whole purpose of the other steps. If this step goes wrong, your security is blown. The pre-master secret is the last parameter in the key derivation/exchange function (RSA or Diffie-Hellmann) and is encrypted with the server's public key. Once the server gets this parameter, both sides will have the necessary information to compute equivalent symmetric keys. This pre-master secret can only be decrypted by an entity with the server's private key and if everything is going correctly, the only entity with that key is the server we are trying to connect to.

At the end of this message we see the Change Cipher Spec which means we are now shifting into the encryption scheme we agreed on in Message 2 and all future communications will be encrypted. The HMAC finish step here is very important. In this step, the client sends a hash value of the entire handshake. This guarantees that both sides are on the same page. A common attack is for a hostile entity to intercept the cipher lists and remove all the strong ciphers, causing the connection to establish a weak cipher that can be broken. The HMAC finish step will detect such an omission and close down the connection.

Key Generation
At this point, both sides generate the symmetric keys necessary to begin our protected communication. Our pre-master secret is used to generate our master secret. The master secret is the same on both sides of the tunnel and is

expanded using multiple hash concatenations to produce a long key block that is chopped into the appropriate sized keys. All keys are generated from sections of this key block.

Message 4
The server is doing its Change Cipher Spec here to switch to the encryption scheme agreed upon in Message 2. It is also doing an HMAC finish to make sure both sides agree that everything was sent and received correctly. We now have an encrypted tunnel using symmetric encryption between our two end points.

Symmetric Ciphers

Once we are to this point, the protection we get from IPSec and SSL/TLS is pretty much the same. It is reliant on the encryption ciphers we select and most implementations of IPSec allow us to select from a variety of algorithms. OpenVPN certainly gives us a good list to choose from but provides us with a very good default using bf-cbc and sha1.

When selecting a symmetric cipher, we want to avoid any scheme that includes DES. DES only uses a 56-bit key which is no longer considered secure. We also want to avoid algorithms with 3DES as their symmetric cipher. 3DES, with its effective key length of 112 bits, is still considered secure, but it requires significant processor overhead as it just runs the DES algorithm three times. There are many algorithms out there that are stronger and faster and you should not have to rely on 3DES any more.

We also want to use an encryption mode that continues to change our cipher text in random ways. If an attacker can get enough cipher text encrypted with a common key, they may be able to eventually weaken the key to the point of compromise. This is very hard to do and often requires an enormous amount of cipher text, but if you have a long duration tunnel between two networks, you can generate a lot of cipher text over time. When not using pre shared secrets, OpenVPN changes its keys every hour by default but to add shorter term protection, we want to use ***Cipher Block Chaining*** mode (CBC). This mode takes input from each proceeding block of encrypted text and uses it to modify (XOR) the next block of test. This way no amount of cipher text can leak information about our key. The first block of text is modified by a random string of characters called an ***initialization vector*** (IV). Generation of a random or hard to predict IV is vital to the effective function of CBC.

OpenVPN defaults to using bf-cbc for its symmetric cipher. This refers to Blowfish in its Cipher Block Chaining mode using a 128-bit key. Blowfish is a very strong algorithm with no known weaknesses. Its 128-bit key provides us with a large enough key space to make brute force key attacks impossible in ***polynomial time***. Blowfish is not only very secure, it is also one of the faster

algorithms available. On Linux you can run an OpenSSL speed check to see the relative speed of algorithms on your hardware.

```
Spiritwrack root #openssl speed
```

The 'numbers' are in 1000s of bytes per second processed.

Type	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
md2	781.82k	1658.54k	2302.72k	2551.81k	2646.65k
mdc2	2303.33k	2601.19k	2691.07k	2713.26k	2722.47k
md4	8088.59k	28497.81k	81425.75k	152377.69k	204186.28k
md5	6658.93k	23101.61k	64989.44k	119329.11k	154162.52k
hmac(md5)	7759.01k	26624.06k	72236.29k	125175.13k	156237.06k
sha1	6341.00k	19797.27k	46929.07k	71337.98k	84077.23k
rmd160	5810.05k	16892.76k	36562.60k	51938.65k	59072.51k
rc4	90142.04k	103726.75k	109965.65k	111563.43k	112017.41k
des cbc	20494.70k	21585.24k	21716.22k	21814.97k	21749.76k
des ede3	7474.74k	7685.31k	7816.62k	7852.37k	7858.86k
idea cbc	15655.19k	16604.78k	16738.56k	16802.47k	16826.37k
rc2 cbc	8038.72k	8356.65k	8447.32k	8468.48k	8467.80k
rc5-32/12 cbc	57561.14k	64905.50k	66914.74k	67767.64k	68263.94k
blowfish cbc	32069.08k	34651.74k	34664.70k	34956.07k	35124.57k
cast cbc	22870.35k	24536.13k	25176.06k	25432.19k	25384.28k
aes-128 cbc	20298.75k	20858.41k	21088.09k	21236.53k	21181.78k
aes-192 cbc	17338.95k	17913.43k	18051.93k	18111.83k	18128.90k
aes-256 cbc	15485.55k	15934.95k	16042.58k	16142.21k	16100.01k

Note: some unrelated output omitted. System is Dell PIII 1Ghz 512K RAM running Gentoo Linux.

The above numbers will help us understand which algorithms are good options. Let's just look at the 64-byte numbers so we have a common reference. We aren't going to use DES, but its good to have it included in the speed comparisons; DES can encrypt/decrypt roughly 22M per second. DES-ede3 is the symbol used to represent 3DES and it is not surprising to see it coming in around 7.5M per second. This is roughly one third of DES, which makes sense as 3DES is really just DES run three times. Now look at Blowfish, nearly 35M per second, which is over 50% faster than DES and almost 500% faster than 3DES. The Advanced Encryption Standard (AES) in 128-bit mode is also an excellent choice for block ciphers running about 21M per second. If you are paranoid and have the extra processing power, AES provides a 192-bit and 256-bit key space with throughputs of 17M/s and 15M/s respectively.

Note: RC4 is a stream cipher and is not practical for use with the block level data we would most likely want to transfer. RC4 would be good for streaming media like audio or video, but is not a good solution for a tunnel designed to pass a variety of undetermined media types.

Note: RC5-32/12 is listed in this output but is not an available cipher for TLS in OpenVPN. It has better performance numbers than Blowfish, but that is partially due to the low number of rounds of permutation it uses. It is also starting to show some cryptanalysis weakness particularly with this lower number of rounds. The 12 in the cipher name refers to the number of rounds of permutation done and analysis has shown that 18 is the number needed to ensure good encryption with RC5 [LDH03]. Several smaller key variants of it have also been cracked using mathematical weakness in the algorithm, including RC5-32/12/6 to RC5-32/12/8 [RSA04]. While the TLS implementation is RC5-32/12/128, it is not available in OpenVPN. It seems prudent to use Blowfish or AES.

Note: I often wondered why Rijndael was chosen over Twofish (striped down Blowfish) to receive the NIST's title of Advanced Encryption Standard (AES) considering the speed advantages of Twofish. For those of you familiar with the AES competition, you know that it also required an algorithm to perform well on a variety of hardware including small, low energy smart cards. Apparently Rijndael was better overall and performs quite well in the very restricted environment of low-energy devices [LDH03]. Since the competition, there has been a lot of noise about the XSL algebraic attack on AES. This attack theoretically weakens AES to a point where it may be prematurely broken. At this point, the attack is just academic as it is still to large a key space to be tested, but one may be wise to assume that the XSL attack or one like it will cause a premature retirement of AES over the next couple decades [Cou04].

HMAC/Hashing

Once we have our keys exchanged and are using a symmetric algorithm to secure our tunnel, we can start sending data. There are two things we want to accomplish with data transfer. First, we want to make sure that what we sent is the same as what is received on the other end. Attackers may not be able to

read our messages as they cross the public network, but the can still go in and randomly change data. The attacker won't know what they have changed, but it could have negative effects when it reaches the other side. Protection against data corruption or tampering is referred to as data integrity. Second, we want a way to ensure that when someone sends something, they can not go back later and say they did not send it. This is called non-repudiation.

To ensure data integrity, we use what is called a hash. We run the text of our message through a one way function that creates a fixed length string (128 bits for MD5 and 160 bits for SHA1) of characters and letters that represents our message. We send our message along with this cryptographic summary attached at the end. When our message is received at the other end, the receiver runs our message through the same one way function and compares the results. If the strings match, then the receiver knows the message has not been changed in route. This is how we guarantee data integrity.

So what is stopping an attacker from simply removing our hash string, changing the message, and making a new hash string? We use what is called HMAC. Before we run our message through our one-way function, we will attach a secret key to the front of it. This key will get hashed along with our message. When the message is received at the other end of the tunnel, the receiver will open the message and make sure it has our key attached to the front of it. This HMAC key, by the way, is one of the keys we exchanged above during our Key Derivation/Exchange step. If an attacker changes our messages and attaches a new hash, they will be unable to reproduce our key and thus the receiver will know the message did not come from us.

A pleasant side effect of using an HMAC is we have a way to achieve non-repudiation. Since we require the presences of a key that only our sender knows, we can now prove they did indeed send this message. They can not go back and deny the messages origin. This is an important feature for ecommerce, but is just a nice touch with VPNs.

OpenVPN selects by default the only hashing algorithm that we should use, SHA-1. MD5 is in wide spread use, but has begun to crack. The strength of a hash algorithm is equal to one half the actual size of the key space due to what is called the birthday paradox. So, MD5's 128-bit key really only provides 2^{64} protection, which is approaching a level that can be brute forced. Also pseudo collisions have been found in MD5 that may weaken it even more [RSA96]. Expect MD5 to fall apart over the next decade, so conservative approaches will use the SHA-1 hashing algorithm. SHA-1 uses a 160-bit key for a 2^{80} effective key space making it about 65000 times harder to brute force than MD5. SHA-1 has not been weakened by mathematical attacks. Our speed chart above shows 23M per second for MD5 and 20M per second for SHA-1. We can probably live with 20M per second in most implementations.

Additional OpenVPN Add-ons

We mentioned some of these above but they deserve another mention under this section. OpenVPN doesn't stop with the security provided by SSL/TLS. It goes several steps further in providing better security today as well as a level of proactive security for unknown exploits in the future.

OpenVPN can be set up to run as user "nobody" on UNIX/Linux as well as chrooted to its home directory. These together create a powerful sandbox effect that dramatically slows or completely neutralizes the effect an attacker will have when compromising the OpenVPN daemon. Even before the attacker can access the daemon, OpenVPN has the tls-auth option. This feature allows you to have an additional pre-shared secret. This key is checked before the TLS handshake is even initiated. This provides tremendous protection against unknown buffer overflows or other problems in SSL/TLS itself and provides one more level to our defense-in-depth. OpenVPN also provides certificate revocation list ability with the --crl-verify option to stop compromised certificates from accessing the server.

OpenVPN Future

OpenVPN development is alive and very active. The project is run by James Yonan and is continuing to improve all the time. The current stable version is 1.60 but 2.0 beta is now available. The 2.0 version includes some significant enhancements to move OpenVPN further ahead of the alternatives. Following are a couple of the more important improvements.

Single UDP port, config file and TUN interface

In versions 1.6 and lower, you need to use a separate UDP port, configuration file, and TUN/TAP interface for each connection you want to make to the VPN. This can be a bit of a maintenance issue in larger implementations. As of 2.0 this goes away. You will now be able to run multiple connections over a single UDP port, using a single TUN/TAP interface, and a single configuration file. Your config file will obviously be a bit more complex, but management will be dramatically improved with this enhancement.

Pseudo DHCP improvements

Using the --ifconfig-pool, --push, and --pull options you can send out remote addresses, and push or pull many configuration options to further simplify set up and maintenance of remote machines. This is a huge advantage for users with many external clients, or road warriors who are connecting from changing IP addresses.

Other SSL VPNs

The current state of commercial SSL VPNs is disturbing at best. The term "SSL VPN" is in my opinion being badly distorted by almost all the vendors claiming to offer the product. What they are really offering is an SSL gateway which is quite a different product. Again the definition of a VPN is a device that provides a site-to-site encrypted tunnel between two end point hosts or networks that allows arbitrary traffic to pass between them. For the most part, the SSL VPN products on the market fall short of this mark while claiming they meet it.

The term VPN carries with it an expectation of the highest level of security. When someone looks at a product labeled a VPN, they believe that the protections we have been discussing thus far are in place to some degree. Many commercial SSL VPNs are carrying this label without including the above protections. The architectures they are suggesting have serious security problems.

The Four Horsemen of SSL VPNs

SSL gateways provide access to corporate applications on an application by application basis, which violates are definition of a VPN right from the start. They use four methods to do this: proxying, application translation, port forwarding, and network extension. Network extension is the only one of these methods that actually creates a VPN. Few of the SSL VPNs provide this feature, and fewer still provide a working version of it.

The business push on commercial SSL VPNs is their ability to function without a client. This is confusing at best and irresponsibly deceptive at worst. What these vendors are really claiming is that you can access corporate applications using the universal client, the Web browser. They are trying to sell simplicity and flexibility, eliminating client installation for remote users, or (gasp) allowing mobile users to "VPN" to the corporate network from public machines like kiosks. What they aren't in a hurry to tell you is that Web browsers only work on the first two levels of access, proxying and application translation. In order to do port forwarding, and particularly network extension, you need a client, which often requires administrative access to the machine you are using. So much for clientless VPN and so much for using public kiosks.

Proxying

Proxying is simply providing an intermediary between an external and internal application. This intermediary usually pretends to be the end point for both sides of the connection and accepts the client request, rewrites it and sends it to the server. Return traffic is handled the same way. ***Application Level Gateway*** (ALG) is a common name for devices that do this, or in simpler situations, just a

Web Proxy. Many ALG's make sure the client request is well formed before they forward traffic to ensure proper resource usage. Regardless, the application request is completed and sent back to the client over the SSL connection. This method of mediating traffic is slightly slower than normal as the gateway must decode/encode the packets an extra time as well as inspect the contents. It works well with Web based protocols but struggles beyond that. A common use would be to couple the proxy with authentication and allow access to a private intranet website for remote users. It also requires a special proxy for each and every protocol. This system does not provide site-to-site connectivity for arbitrary traffic and requires new coding for any addition protocols that come up.

Application Translation

Some applications, like FTP and other file sharing services, can adapt to translation. This means the internal protocol is translated to HTTP and HTML for delivery to the client's Web browser. This works in some circumstances, but is hard to get right with some protocols, like the black magic that is Windows file sharing. It also destroys the look and feel of applications as they are limited to the display capabilities of HTML. This translation needs to be done on a protocol by protocol basis and can not handle many services. It requires a special translator for each and every protocol. This system does not provide site-to-site connectivity for arbitrary traffic and requires new coding and analysis for any addition protocol that come up.

Port Forwarding

Port forwarding is what firewalls do. Traffic to a port on one IP address (usually your gateway) is simply redirected to the same (or sometimes different) port on another machine. If the packet qualifies, the gateway simply passes the traffic without inspecting its contents. This works well for some common services that use predictable ports. However, many protocols do not use a fixed port, instead using a range of ports or random ephemeral ports. It also requires individual forwarding for each service or port, one at a time. This system does not provide site-to-site connectivity for arbitrary traffic and requires new coding for any addition protocols that come up. It also requires that software be installed on the client machine; if you think that sounds like a "client" you are correct! For this client to work correctly, it requires administrative access on the box. Again, so much for clientless VPN, and so much for public kiosks.

Network Extension

Of The Four Horsemen of the SSL VPNs, the only one that provides true VPN service is network extension. As with traditional IPSec VPNs, the devices that handle network extension create a site-to-site tunnel that can handle arbitrary traffic. No surprise, this configuration requires a client in all cases. On top of that, it requires administrative access to the host or gateway machine, which you

are not going to get on public machine (I hope). This is going to sound like a broken record, but so much for clientless VPNs and so much for public kiosks. Few of the commercial devices claim to do network extension and of those that claim to do it, few actually do it correctly [Sny04]. For those select few (Checkpoint, NetScreen, OpenVPN), we reserve the name SSL VPN.

Hybrids

Some of the devices listed as SSL VPNs actually just provide SSL access to web based applications, and then use IPSec access for network extension (Cisco). I would suggest that labeling this device an SSL VPN is a misnomer at best.

Security Issues

For all the rest of the commercial SSL VPN market, shame on you. I don't know if this phrase is already claimed, but if it isn't, I'm labeling it ***Hosner's Lament***:

> "The one thing worse than bad security is bad security that creates the illusion of good security"

VPNs are an extension of your network. Hence the term network extension above. They represent your LAN at a location outside your company. Most of the commercial SSL VPN products focus heavily on "clientless VPN" as an obvious marketing feature. The following quote is taken directly from the marketing material of one of these vendors:

> "The biggest difference between SSL VPNs and traditional IP Security remote access VPNs is that the IPSec standard requires installation of client code on the end user's system, while SSL VPNs focus on making applications available through any Web browser."

This is not only deceptive, it is dangerous. Either these companies lack an understanding of security and VPNs, or they are unethically presenting a product to unknowing users that gives them the impression of safety with the well know label of VPN without the security features that are traditionally associated with this label. Here is another marketing snippet that affirms this strategy:

> "We ruled out changes to client systems as unacceptable and not in the spirit of SSL VPNs' goal of security with ease of use."

That philosophy would be fine if it were true. The real situation is security is being compromised for the sake of ease of use. Imagine this, one of these SSL

VPNs allows "clientless access" from public kiosk machines. Your user goes to one of these kiosks where a 14 year old has installed a keystroke logger. Your user connects to your “VPN”, authenticates to your network with their username and password, and accesses a bunch of material. Your 14 year old now knows your users username and password plus a large amount of data about your internal applications. Or worse yet a trojan on the kiosk starts a worm attack on your network through the SSL tunnel. Now you have a worm hammering the soft underbelly of your network through a trusted tunnel. Public clients are by definition untrusted clients. VPNs work on the foundation that both sides of the connection are trusted. If you are providing an application gateway, maybe this is acceptable, but if you are using a VPN, you must have trusted and authenticated hosts on both ends of the connection. The following quote says it all.

> “Your VPN--IPSec or SSL--is only as secure as the laptops, PCs or PDAs connected to it.”[Phi03]

How about cookies, temporary files, browser history, and session information? Many of these vendors claim to have programs that clean these areas up after use. Unfortunately, to use these features, you need to have administrative rights on the machine. If a kiosk lets you have admin access to it, you can bet it is already infected with a keystroke logger, remote control software and enough bugs and viruses to make the Amazon jungle jealous. This feature is most important on public machines, which are the most unlikely to provide the permissions necessary to allow this sort of scrubbing.

Ever heard of a man in the middle attack? How do these so called SSL VPNs defend against that? Do the users remember a 1024-bit certificate that they type into the password box at authentication challenge? How do we know the server is really the machine we want to connect to and not some hostile intermediary? What we have here is a cotton shirt claiming to be suit of armor. Both devices provide access to internal corporate resources. The difference is only devices that deal with trusted and authenticated end points using installed client software are actually capable of creating a VPN. All other current claimants are charlatans and very dangerous ones at that. These companies put their name and the label of VPN on these products which give the unknowing IT manager a sense of security. This sense will stay until a terribly damaging breach occurs that could wipe out the company, and definitely the IT manager’s career. Will these “SSL VPN” corporations bear this responsibility? Of course not. I imagine if you look at their user license agreement you will find text absolving them of any responsibility for compromise related to their product. Use commercial SSL VPN solutions with extreme care!!! Just to give you a taste of the lack of security knowledge some of these vendors have, here is a quote that represents their level of understanding, again taken directly from marketing material. Company name removed to protect the guilty.

> "The XXX Server is protected against any kind of web server, password or other hacking attempts."

Anyone who has spent more than 5 minutes in the computer security field knows the above statement is ridiculous and a common indicator that we are dealing with an entity with little experience in security. For anyone interested in more information on these products plus some excellent comparisons and product information please see the reference at the end of this document [Sny04].

Conclusion

IPSec VPNs are unnecessarily complex. The IPSec protocol is dense and confusing providing many opportunities to compromise its security by implementing it incorrectly. As Bruce Schneier says:

> "We strongly discourage the use of IPSec in its current form for protection of any kind of valuable information …… however, we …. recommend IPSec when the alternative is an insecure network."[SF99]

Vendor IPSec packages are expensive and most open source variations are hard to install and configure. But now we have a choice.

OpenVPN is an SSL/TLS-based user-space VPN that provides industry tested security with tremendous ease of use. It is available on most modern operating systems and gives you the flexibility to work in a variety of modes that are easy to understand and hard to make insecure.

Glossary

Application Level Gateway – (ALG) Usually a component of a firewall, an ALG acts as an intermediary between an external client and an internal service.

Asymmetric Encryption – System for encrypting/decrypting and digitally signing messages. Uses different keys (public/private) on either side of the connection. Does not perform well for bulk encryption when compared to symmetric encryption.

Authentication – Determining an entity's identity and possibly the level of access they are allowed to have.

Authentication Header – (AH) A component of IPSec that allows masking and protection of the original source address of packets. Its value is questioned as ESP and NAT do most of its functions with the exception of source address verification, which has limited use.

Broadcast repeater – A networking device that regenerates OSI level 2 broadcasts onto a different network segment

Certificate – An electronic data structure that contains identification information on an entity as well as that entity's public key. This public key is usually signed by a certificate authority.

Certificate authority – (CA) An entity that does physical validation of other entities and then signs these other entity's keys to prove they are who they claim to be.

Cipher Block Chaining – (CBC) A method of randomizing cipher text to reduce the amount of available cipher text encrypted using a common key.

Confidentiality – Keeping information private between only those who need to know it.

Cryptographic Primitive – Refers to the basic building blocks of a crypto system. Symmetric ciphers, Asymmetric ciphers, message digests and digital signatures are all primitives.

Digital Signature – Using a private key to sign a cryptographic hash of a message to guarantee authorship and integrity of the message.

Ebuild – Package management system used by Gentoo Linux for distributing and installing applications.

Encapsulating Security Payload – (ESP) A component of IPSec that enables the encryption and protection of a message.

Hosner's Lament – The one thing worse than bad security is bad security that creates the illusion of good security.

Initialization Vector – (IV) Random or hard to predict string of characters used to modify the first block of a Cipher Block Chaining mode encryption session or any other encryption requiring pseudo-random seeding.

Integrity – A guarantee that the information sent is the same as the information received.

Internet Engineering Task Force – (IETF) Group organized to design and disseminate technical standards for the Internet. The keepers of the fabled RFC's.

Internet Key Exchange – (IKE) Handshake protocol used by IPSec. Uses a 6 message format to authenticate tunnel end points and exchange session keys to build and maintain an IPSec VPN tunnel.

IPSec – Internet Protocol Security. IPSec is a standard for creating VPNs. Primarily defined by RFC's 2401-2412. IPSec has received much criticism for its complexity.

Key Agreement – A process by which two entities can calculate the same key over a public network without eavesdroppers also being able to calculate the key. The oldest and most wide spread algorithm for key agreement is Diffie-Hellmann.

Key Distribution – System for moving cryptographic key material between entities over a secure medium.

Key Exchange – System for transferring cryptographic key material between entities, usually over an insecure medium. RSA and Diffie-Hellmann are examples.

Message Digest – A cryptographic summary of a message. If any character in a message is changed, all of the message digest will change.

Network Address Translation – (NAT) Usually a component of a firewall. NAT allows multiple internal clients to share external addresses. NAT (actually PAT) maps private internal addresses to ports on an external address allowing public address space conservation and source address masking.

Network Extension – This refers to extending the boundary of the corporate network to include remote machines that are connected over the public Internet. This is the primary purpose of VPNs.

Non-repudiation – The inability of the sender to later deny they sent the message. Achieved using digital signatures.

OS Ring Architecture – Operating system design philosophy that breaks the OS into rings of privilege starting with Ring0. Kernel processes and all essential system components run at Ring0 or Ring1. User applications run at Ring3. The philosophy states that processes at higher rings can not interfere with processes at lower rings thus creating a more secure, stable, multi-user environment.

Perfect Forward Secrecy – This philosophy states that if a key is compromised for one section of an encrypted communication, it will only allow access to that section. Future sections of the communication are protected differently (different key) and will not be compromised.

Polynomial Time – Means the function can be computed in a reasonable time. This is in comparison to exponential time meaning as the function adds complexity, the time to crack it increases exponentially, quickly surpassing the reasonable time measure.

Pretty Good Privacy – (PGP) A system for protecting information. Uses the IDEA cipher to encrypt data and incorporates public key cryptography using a web of trust. Common system for protecting email communications between trusted entities.

Public Key Cryptography – System of using public and private key pairs to protect data and authenticate entities. Includes certificate authorities.

Redhat Package Management – (RPM) Package management system used by Redhat Linux for distributing and installing applications.

Request For Comment – (RFC) Standard system for communicating technical standards for the Internet. RFC's are created and maintained by the IETF.

Secure Socket Layer – (SSL) Protocol and crypto libraries used to protect communication over the Internet. Used primarily in e-commerce, SSL is making headway into link encryption environments like VPNs. Originally developed by Netscape in the early 1990's.

Symmetric Encryption – System for encrypting/decrypting traffic using the same key on both sides of the connection. Very fast when compared to asymmetric encryption.

TAP Virtual Driver – A driver interface that allows Ethernet bridging. The TAP interface communicates with the actual physical interface eliminating some complexity and rigidity.

Transparent Bridging – Using a device that connects two different subnets together and allows traffic to pass between them without routing. Stores MAC addresses of each network in a table and uses this data to bridge networks.

Transport Layer Security – (TLS) To condense the code trees of SSL and centralize management and development of the protocol, the IETF developed TLS. TLS is essentially the latest version of SSL and is really just SSLv3 with some minor improvements. Often abbreviated SSL/TLS.

TUN Virtual Driver - A driver interface used for IP traffic. The TUN interface communicates with the actual physical interface eliminating some complexity and rigidity.

Virtual Private Network – (VPN) A device that is capable of creating a semi-permanent encrypted tunnel over the public network between two private machines or networks to pass non-protocol specific, or arbitrary, traffic.

Windows Internet Name Service Server – (WINS) Device that maps NetBIOS names to IP addresses allowing machines on different subnets to still use services like Windows file sharing. Samba is able to act as a WINS Server.

Work Cited

[Cou04] Courtois, Nicholas T (2004). Is AES a Secure Cipher? Retrieved August 1st, 2004 from http://www.cryptosystem.net/aes/

[Dum97] Dumon, Pieter (1997). OS Kernels: A Little Overview and Comparison. Retrieved August 3rd, 2004 from http://tunes.org/~unios/oskernels.html#rings

[Free04] FreeS/WAN. FreeS/WAN home page. Retrieved July 28th, 2004 from http://www.freeswan.org/

[LDH03] Law Y., Doumen J., Hartel P. (2003) Survey and Benchmark of Block Ciphers for Wireless Sensor Networks. Retrieved August 4th, 2004 from http://www.ub.utwente.nl/webdocs/ctit/1/000000eb.pdf

[Phi03] Phifer, Lisa (2003). VPN: Tunnel Vision. Information Security Magazine Online. Retrieved July 26th, 2004 from http://infosecuritymag.techtarget.com/ss/0,295796,sid6_iss21_art83,00.html

[Res01] Rescorla, Eric (2001). SSL and TLS: Designing and Building Secure Systems. Indianapolis, IN: Addison-Wesley.

[RSA96] RSA Laboratories (1996). The Status of MD5 After a Recent Attack. CryptoBytes. Retrieved July 23rd, 2004 from ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto2n2.pdf

[RSA04] RSA Laboratories (2004). Retrieved August 2nd, 2004 from http://www.rsasecurity.com/rsalabs/node.asp?id=2103

[SF99] Schneier, B Ferguson, N. (1999). A Cryptographic Evaluation of IPSec. Retrieved July 10th, 2004 from http://www.schneier.com/paper-ipsec.pdf

[Sny04] Snyder, Joel (2004). SSL VPN Gateways. Network World Fusion Online. Retrieved July 25th, 2004 from http://www.nwfusion.com/reviews/2004/0112revmain.html

[Yon04] Yonan, James (2004). OpenVPN Source Forge home page. Retrieved August 1st, 2004 from http://openvpn.net

Bibliography

Kolesnikov O., Hatch B. (2003). Building Linux Virtual Private Networks. Indianapolis, IN: New Riders.

Loshin, Peter (2002). Big Book of IPsec RFC's. San Diego, CA: Academic Press.

Northcutt, S et al. (2003). Inside Network Perimeter Security. Indianapolis, IN: New Riders.

Rescorla, Eric (2001). SSL and TLS: Designing and Building Secure Systems. Indianapolis, IN: Addison-Wesley.

Ruixi Y., Strayer T. (2001). Virtual Private Networks: Technologies and Solutions. New York, New York: Addison Wesley.

Schneier, Bruce (1996). Applied Cryptography: Protocols, Algorithms, and Source Code in C. New York, New York: John Wiley and Sons, Inc.

Viega J., Messier M., Chandra P. (2002). Network Security with OpenSSL. Sebastopol, CA: O'Reilly.