Secure WLAN Networks via Encrypted OpenVPN Tunnels

Secure Tunnels

Wireless networks may be practical but they are also quite dangerous. Integrated WEP encryption is no real problem for attackers who can snarf and manipulate data or even inject packets. An encrypted tunnel that uses OpenVPN to protect your data provides a secure solution. BY ACHIM LEITNER, DANIEL COOPER, OLIVER KLUGE

Wireless LANs allow attackers to war drive their victims' premises and grab all the data packages travelling across the network with a little help from the WLAN cards installed in their lap tops. You can compare this with a victim installing a network socket at the nearest bus stop and hoping nobody will bother plugging in to it.

In urban areas the risk is extremely high even for private users. Wardrivers constantly search for WLANs that allow them to gatecrash internet accounts, snarf data or hack into large enterprise file servers possibly causing denial of service conditions.

Whereas an attacker would need access to a network socket or the wire to hack a wired LAN, a WLAN pays little attention to walls and fences. To provide a modicum of protection even the earliest wireless LANs used the "Wireless Equivalent Privacy" approach.

WEP aims to provide a level of security equivalent to that available in wired environments and uses its own encryption algorithms to do so. Key lengths or 40 bits were originally envisaged, but today's devices use 128 bits. Unfortunately, the algorithm used here is quite weak: 40 bit keys can be cracked in a matter of minutes, and even 128 bit keys will tumble within a few days. In other words WEP provides very little protection.

Encryption

A Virtual Private Network (VPN) that receives the traffic, encrypts it, transmits it across the wireless LAN, and decrypts it on the other side is normally the best solution. A VPN uses the traditional WLAN, but looks like an additional network – a virtual one – from the client's point of view. Figure 1 demonstrates this principle using the OpenVPN[1] VPN package. The laptop and the desktop are connected via a WLAN and can reach each other's true IP addresses on the wireless LAN.

The VPN assigns an additional IP address to both the laptop and the desktop. The VPN encapsulates any data sent to the virtual addresses and transmits it to the real address of the host at the other end of the connection. The host on the receiving end will then decapsulate this traffic and treat it as though it had arrived via its virtual IP, thus creating a tunnel between the laptop and the desktop.

Additional firewall rules allow both computers to receive any data arriving through the tunnel. So, any packages an attacker inserts into the WLAN have no chance of getting through.

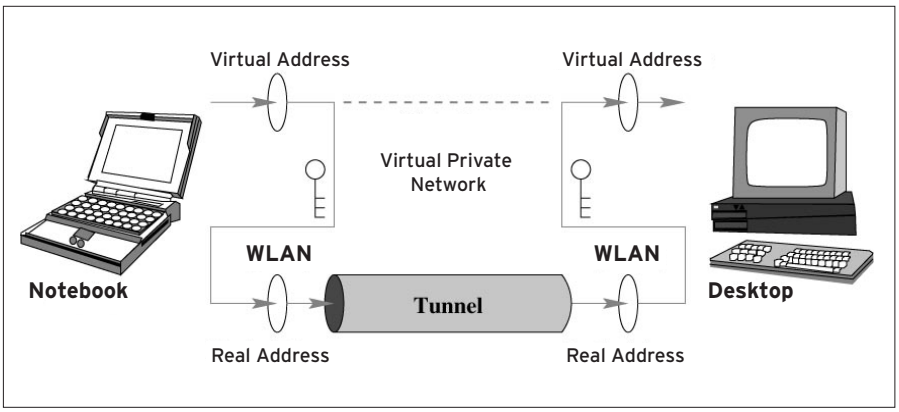

Figure 1: The Virtual Private Network is tunneled along a path that starts and ends at the real IP addresses of the laptop and desktop computers

The VPN uses cyptographic techniques to protect the tunnel. In contrast to the insecure WEP technology, tried and tested algorithms are used to provide a high level of security here. The tunnel thus protects any data sent through it from uninvited guests, at the same time ensuring that nobody can spoof a legitimate laptop and transmit data through the tunnel – the tunnel walls are solid.

OpenVPN

The VPN principle has been implemented in various protocols, products, and projects. OpenVPN is a stable and simple implementation that does without manipulating the kernel or the IP stack.

Installation

To install OpenVPN you will probably want to download the source package, *openvpn-1.3.1.tar.gz*, from [1], and then unzip and install it (you need root priveleges).

```
tar -xvzf openvpn-1.3.1.tar.gz
cd openvpn-1.3.1
../configure --disable-lzo
make
make install
```

Note that we used the *--disable-lzo* flag with configure in order to disable compression. However, you can optionally install the LZO library[3]. You will definitely need the OpenSSL library and developer files. SuSE users require two separate packages, for example: *openssl* and *openssl-devel*.

Installation is easier for Debian users – just type the following to install OpenVPN:

```
apt-get install openvpn
```

The OpenVPN developers also provide RPM packages for Red Hat 7.2 and 7.3.

At both ends of the tunnel it collects traffic destined for the other end, encrypts this data using a locally stored key, and transmits the packages secured in this way through to the other end of the tunnel.

The receiving end decapsulates the transmission and checks its origin. Only data secured with the correct key (i.e. the secret common to both ends) will be decapsulated and forwarded – any other data is rejected. This allows you to tunnel data packed in secure containers through a maze of insecurity.

The following example assumes that the wireless network is attached to *wlan0*. The desktop is also equipped with a traditional, wired network interface card, referred to as *eth0*. This network provides access to other computers in the local network and to the Internet.

First Steps

If you have not already installed it, you will need to install the OpenVPN package first (see the "Installation" box). The simple procedure described below assumes a static IP address – so your computers will need fixed addresses that do not change after every reboot.

The procedure is more complex if you use a DHCP server to assign dynamic addresses. OpenVPN does not modify the kernel, instead using the TUN/TAP driver [2] to ensure the forwarding of data packages.

This step is quite simple as the required kernel module has been part of the major distributions kernel trees for some while now. The next step is to load the module. Ensure that you have superuser (root) privileges and type the following command:

```
modprobe tun
```

In order to provide secure functionality OpenVPN will need keys. The simplest case assumes that both computers will be working with a *shared secret*. The command is then

```
openvpn --genkey -secret ➥
secret.key
```

will create a key and store it in the *secret.key* file. Only the two computers involved should know this key, which should be readable for root only – anyone who knows the key can easily crack the tunnel.

The key also needs to be copied to the second computer – make sure that this step is secure. Somebody might already be listening in on your wireless network, so why not use a floppy, which you would then reformat. If you have already installed a program such as OpenSSH, PGP, GnuPG, or similar, you can also use this program.

Digging the Tunnel

Now let's get that tunnel up and running. For this step OpenVPN will need the (static) IP address of the target computer, the name of the tunnel device (*tun0* by default), and both virtual IP addresses for the VPN.

TUN Device

The tunnel device is available in the current kernel, and from [2] for older versions. If you want to compile the current kernel yourself, you will find the TUN module under "Universal TUN/TAP device driver support" in the "Network device support" section of *make xconfig*.

You can compile and install this module individually at any time without needing to replace the entire kernel. After configuring the kernel simply type:

```
make modules
make modules_install
```

You will now need to create the device file, */dev/net/tun*. If the */dev/net/* does not exist, type *mkdir /dev/net/* before creating the device:

```
mknod /dev/net/tun c 10 200
```

And don't forget the file with the key, of course. The commands on the laptop are as follows:

```
openvpn --dev tun0 ➥
--remote [Real_DesktopIP] ➥
--ifconfig [Virtual_LaptopIP] ➥
[Virtual_DesktopIP] ➥
--secret secret.key
```

You need to be superuser (root) to run this and any following commands. The commands for the desktop are as follows (the IP addresses just need to be rearranged, of course):

```
openvpn --dev tun0 ➥
--remote [Real_LaptopIP] ➥
--ifconfig [Virtual_DesktopIP] ➥
[Virtual_LaptopIP] ➥
--secret secret.key
```

You can use more or less any IPs for the virtual addresses, however, they will need to be **private addresses**. Your virtual addresses should be in a different block from your real addresses to allow simpler **routing** – the real network should be easy to distinguish from the virtual network.

Address Assignments

As a practical example, let's assume that the real IP address 172.16.0.1 has been assigned WLAN adapter in the laptop, and that the desktop answers to 172.16.0.2. The VPN will need to use addresses in the private address space, for example 10.0.0.1, as the virtual IP address for the laptop, and 10.0.0.2 for the desktop. In this case, the command for the laptop is as follows:

```
openvpn --dev tun0 ➥
--remote 172.16.0.2 ➥
--ifconfig 10.0.0.1  10.0.0.2 ➥
--secret secret.key
```

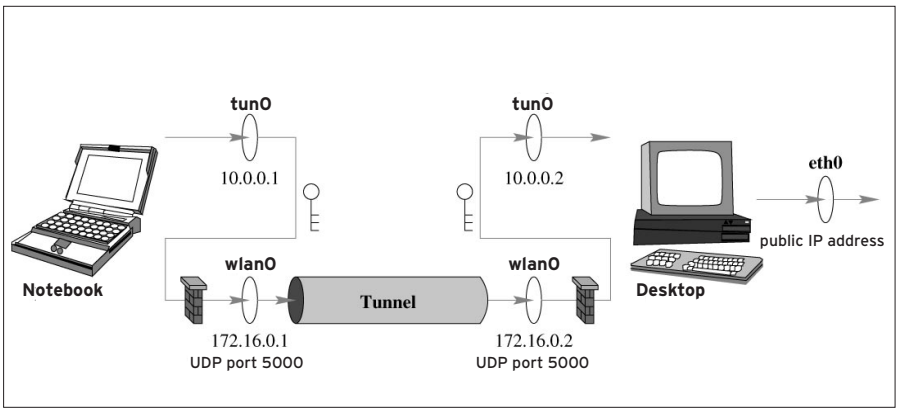

Figure 2: Firewall rules can prevent outsiders entering your WLAN. Only the OpenVPN tunnel is allowed to transmit on the WLAn interface

And for the desktop:

```
openvpn --dev tun0 ➥
--remote 172.16.0.1 ➥
--ifconfig 10.0.0.2  10.0.0.1 ➥
--secret secret.key
```

You can then use ping to test the connection. On the laptop *ping 10.0.0.2* should do the job, and demonstrate that the virtual IP address of the desktop is then reachable.

If everything turned out ok, you can now launch the OpenVPN daemon, allowing OpenVPN to run in the background and use Syslog for logging. Use the *--daemon* flag when you launch OpenVPN to do so, but make sure that you supply the absolute pathname for the file containing the secret key.

On the Right Track

The tunnel is up and running, and traffic is travelling happily back and forth – but your laptop and desktop still need to know *what types* of packages you want to allow through the tunnel. If you use the virtual IP address of the other end of the tunnel for the commands involved, this should be no problem. The OpenVPN call will define the route to use exactly this address. Any other addresses will be routed past the tunnel, just like they were previously.

The route from the desktop to the laptop will work perfectly, provided you use the new virtual address when you want to talk to the laptop. The real addresses assigned to the WLAN adapters in the laptop and the desktop only serve one useful purpose now: they are the endpoints of the tunnel. However, they will no longer be accessed by normal connections.

You will need to put a few finishing touches to the route from the laptop to the desktop and thence to the other computers on your local network and the Internet, as the default route needs to be redefined. The following command allows the laptop to direct all of its traffic through the tunnel:

```
route del default
route add default gw 10.0.0.2
```

Of course, the default route does not apply to packets destined for the real WLAN IP address of the desktop (172.16.0.2). And this is a good thing, as the tunnel is bound to this address. So now the desktop just needs to know that it may need to forward some of the packets that it decapsulates. Use the following command:

```
echo "1" > /proc/sys/net/ipv4/ ➥
ip_forward
```

Fireproof

That nearly completes the job at both ends. Both the laptop and the desktop are using the tunnel, your traffic is

GLOSSARY

Private address: *Normal, public IP address are globally unique, and need to be so, for packages to find their way to a target. In contrast, private IP addresses are valid only on local networks and are not routed on the public Internet. This allows multiple networks to use the same private addresses. Various IP address blocks have been reserved for this purpose: 10.x.x.x and 192.168.z.z, and 172.16.y.y through 172.31.y.y.*

Routing: *Path selection for IP packets. Linux uses a routing table to select an interface that will permit a packet to get closer to its final target. Stand alone computers do not have many options: 127.0.0.1 uses the loopback device, lo, and everything is transmitted via the default route, eth0, or similar. Routers with multiple network adapters need to make more complex decisions.*

secure and nobody can listen in. However, it is still possible to inject packets, and this would allow an attacker to hijack your desktop's Internet connection.

Even if you have a flat rate, you will probably want to avoid giving bandwidth away. Network services provided by clients and servers (such as Web, SSH or FTP servers), are vulnerable from within the WLAN. And if you run an internal network there is another danger to consider: Any packages injected into your WLAN will sidestep a firewall positioned between the Internet and your internal network. However, you can modify your firewall configuration [4] to remedy this situation.

The OpenVPN distribution also contains a sample script for your firewall. However, you will need to add a few additional rules for your WLAN tunnel combination. Figure 2 shows where you should apply these rules.

OpenVPN uses UDP to transmit encrypted packets to port 5000 at the other end of the tunnel, and uses the WLAN to do so. This means you will need to allow UDP port 500 on your *wlan0* interface. The following command allows you to receive data:

```
iptables -A INPUT -i wlan0 -p ➥
udp --dport 5000 -j ACCEPT
iptables -A INPUT -i wlan0 ➥
-j DROP
```

The last line prevents the computer from receiving any other data via the WLAN. The first ingress rule could be even stricter and use *-s real_IP* to define the IP addresses from which traffic is allowed to originate. This would be the real IP address of the other end of the connection in this case, that is *-s 172.16.0.2* on the laptop.

You will also need to restrict transmitting and forwarding of traffic:

```
iptables -A OUTPUT -o wlan0 ➥
-p udp --dport 5000 -j ACCEPT
iptables -A OUTPUT -o wlan0 ➥
-j DROP
iptables -A FORWARD -i wlan0 ➥
-j DROP
```

The endpoints of the tunnel only forward packages that originate from known partners who have access to the correct (secret) key.

This means you can trust packets that originate from a tun device, and will want to accept and handle them. You will also want to enable traffic through the tunnel. Use the following commands to enable incoming and outgoing traffic:

```
iptables -A INPUT -i tun0 ➥
-j ACCEPT
iptables -A OUTPUT -o tun0 ➥
-j ACCEPT
```

This completes the configuration for your laptop. The laptop is not attached to any other networks, and thus does not need to forward any traffic. The desktop will still need a forwarding rule and should also use masquerading to allow the laptop to send its data onward to the outside world:

```
iptables -A FORWARD -i tun0 ➥
-j ACCEPT
iptables -t nat -A POSTROUTING ➥
-o eth0 -j MASQUERADE
```

Limitations

One hitch with the method described in this article is the fact that you can only use it to secure PCs and laptops. It will not work for a network printer with a WLAN interface.

WLAN aware printers only provide WEP encryption, and often only WEP-40. At first sight, it might seem fairly useless to misuse a printer, as attackers would have no way of collecting their printed output. But it is still a chink in your security armour.

The network is only as secure as the computers attached to it. If an unauthorized person can access the OpenVPN laptop, she automatically has access to the key, and thus to your LAN. Wireless devices are thus particularly prone to theft.

The passwords you select for the services on offer in your WLAN are also important. You might find it annoying having to type those passwords, but having an intruder is definitely a lot more troublesome. ■

INFO

[1] OpenVPN: *http://openvpn.sourceforge.net*

[2] TUN/TAP drivers: *http://vtun.sourceforge.net/tun/*

[3] LZO library: *http://www.oberhumer.com/opensource/lzo*

[4] Marc André Selig: Paketfilter-Firewall, LinuxUser 05/2002, S. 30.

General Terms

Access Control List (ACL): *(in this context) A list containing the non-editable hardware addresses (MAC addresses) of the cards allowed to log onto the network – normally stored on access points and access routers. However, there are some techniques that allow you to spoof other hardware addresses, and this prevents the ACL from providing any real protection for your network – although it certainly is another hurdle the attacker will need to take.*

Station (STA): *Any WLAN device, i.e cards, access points or access routers.*

Wired Equivalent Privacy (WEP): *Using encryption technologies to achieve a security standard equivalent to the standard achievable in "wired" networks for data transferred via wireless LAN that can otherwise be sniffed by anybody interested in doing so. The WEP-40 (40 bit key length), and WEP-128 (104 bit key length) algorithms are somewhat trivial, however, and can be cracked within minutes. This means paying particular attention to security measures in wireless networks, such as ACLs, for example.*

Access Point (AP): *A central node in a wireless network. A participating node will transfer data to the AP which relays it to the receiver. Today's APs normally have an Ethernet port allowing them to be connected to a wired network.*

Basic Service Set (BSS): *A group of stations (STA) with the same identification (BSSID).*

Independent Basic Service Set (IBSS): *Also referred to as an ad hoc network where the participating hosts transmit data directly to each other without accessing a central node. There is no easy way of connecting a wireless ad hoc network to a wired network.*

Distribution System: *Connects multiple wireless (BSS) and/or wired networks to form an ESS.*

Extended Service Set (ESS): *A group comprising multiple wireless networks (BSS) with the same (E)SSID that together comprise a larger, logical network (BSS).*

(Extended) Service Set ID ((E)SSID): *The ID or name of a network.*

Basic Service Set ID (BSSID): *The hardware address (MAC address) of the central node in a network. In the case of ad hoc networks, this is the address of any given participant, in networks with access points (APs) the address of the AP.*