White Paper

Tunneling with Secure Shell (SSH)

VanDyke Software

4848 tramway ridge dr. ne
suite 101
albuquerque, nm 87111

505 - 332 -5700

www.vandyke.com

Tunneling with Secure Shell

Remote access to network resources is increasingly a business requirement, but external network threats must be neutralized. A Secure Shell (SSH) capability called port forwarding allows nonsecure TCP/IP data to be tunneled across public and private networks through a secure, encrypted connection. The benefits of port forwarding are illustrated by a series of concrete examples. VanDyke Software's clients and servers provide an end-to-end tunneling solution to secure client/server applications, which may serve as a lightweight alternative to a Virtual Private Network (VPN).

With today’s increasingly mobile and distributed workforce, providing remote access to travelers and teleworkers is no longer a “nice to have” option. In many corporations, remote access to business applications has become mission critical. At the same time, Internet access is now cheap, fast, and readily available. Leveraging the Internet to extend the local area network (LAN), provide real-time communications, and immediate file transfer and sharing is a scalable, cost-effective solution for corporate network remote access.

However, Internet-based remote access also adds significant risk. Sensitive data can be intercepted, modified, or replayed anywhere between remote workers and the corporate firewall. Broadcast access technologies like cable and wireless are especially vulnerable. Whenever a computer is connected to the Internet, it becomes a potential target for intruders. "Always on" broadband greatly increases this exposure by giving intruders a fixed target to attack repeatedly over time. Unless appropriate measures are taken, allowing remote access over the Internet can compromise usernames, passwords, proprietary data, traveler laptops, teleworker PCs – even the corporate network itself.

Secure Shell (often referred to as SSH) can help to neutralize these threats and make the most of *secure* Internet-based remote access. This standard protocol employs authentication and encryption to ensure the privacy and integrity of data exchanged between clients and servers. To learn more about Secure Shell protocols, authentication methods, and cryptography, refer to our Secure Shell Overview.

Secure Shell can tunnel data from any TCP application with a predefined listening port. Commonly known as “port forwarding”, Secure Shell tunneling makes it easy to secure applications that would otherwise send unprotected traffic across public networks. Application messages relayed from one end of a Secure Shell connection to the other are protected by the cryptographic measures negotiated for that connection. Because several applications can be multiplexed over a single Secure Shell connection, firewall and router filters can be tightened to just one inbound port: the Secure Shell port (22).

The VanDyke Software® VShell® server and SecureCRT® client enable Secure Shell tunneling on Windows®, macOS, and Linux. Cross-platform tunneling is made possible by compliance to the SSH protocol. For the full list of platforms supported by VShell and SecureCRT visit www.vandyke.com.

This paper shows how VanDyke’s VShell server and SecureCRT provide a comprehensive, end-to-end solution to secure client server applications. This paper:

- Examines threats addressed by tunneling over the public Internet or a company Intranet
- Explains how Secure Shell port forwarding, authentication, and access control features work
- Illustrates common applications like email, file sharing, and screen sharing as they are tunneled over residential broadband and WiFi networks
- Considers security implications and where tunneling is best used.

Note: IEEE 802.11 standards have been changed since this article was written in 2006. However, the details regarding tunneling are still accurate.

Tunneling over the Internet

Conference attendees at public PCs. Travelers using a hotel or airport wireless LAN. Day extenders logging back into work at night. Teleworkers conducting business from home. All of these workers can increase business efficiency by leveraging the public Internet to stay connected. But what are the risks?

Consider a teleworker using the Internet to access email (Figure 1). When the worker's client sends mail, messages are relayed to an SMTP server. When the client reads mail, message headers and bodies are downloaded from a POP or IMAP server. Anyone anywhere in this path through the Internet can use a sniffer to capture not only cleartext message bodies, but also email addresses, usernames, and passwords.

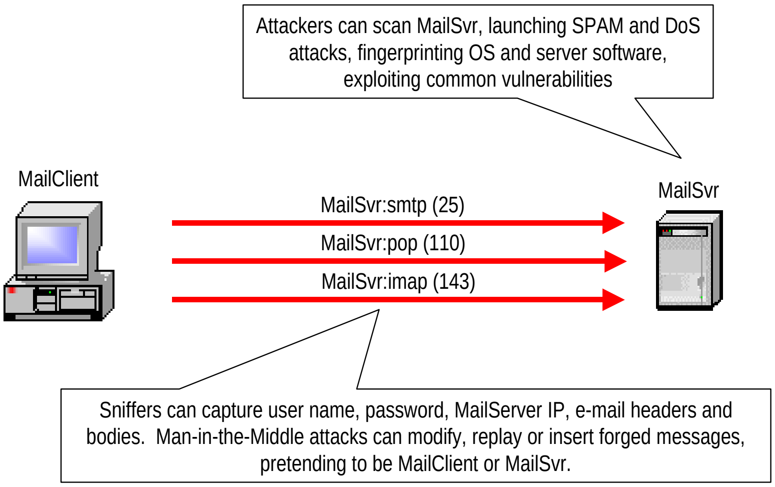

Attackers can scan MailSvr, launching SPAM and DoS attacks, fingerprinting OS and server software, exploiting common vulnerabilities
MailClient
MailSvr
MailSvr:smtp (25)
MailSvr:pop (110)
MailSvr:imap (143)
Sniffers can capture user name, password, MailServer IP, e-mail headers and bodies. Man-in-the-Middle attacks can modify, replay or insert forged messages, pretending to be MailClient or MailSvr.

Figure 1: Typical Remote Access Security Risks

Armed with this stolen data, a passive attacker can replay original or modified messages, even send them to other destinations. By actively masquerading as a legitimate email client or server, a "man in the middle" (MitM) attacker can intercept and drop messages, or insert new forged messages.

Mail-specific security measures like PGP and S/MIME encrypt and digitally sign message bodies, but leave cleartext message headers. Furthermore, they do nothing to protect the mail server from attack. Mail servers listening to well-known SMTP, POP, and IMAP ports are easily discovered by port scans. Hackers can use an open server to relay spam or tie up the server with denial-of-service (DoS) attacks. By "fingerprinting" the server, they can exploit known vulnerabilities in the server's operating system or email software.

Leaving this mission-critical resource wide open to Internet access is clearly unwise. Tunneling with Secure Shell can help by eliminating open ports, blocking unauthorized users, and ensuring the privacy and integrity of all SMTP, POP, and IMAP traffic exchanged between mail clients and servers.

Tunneling over the Intranet

This section primarily applies to WiFi networks that use WEP encryption.

In the past, companies tended to think about "us" and "them," using firewalls to establish a secure perimeter between untrusted outsiders and trusted insiders. This view is increasingly giving way to layered perimeters that enforce more granular security at workgroup, system, and user levels. These policies are commonly implemented with operating system access controls – for example, file and printer sharing privileges extended in a Windows® domain, based on login authentication through the Primary Domain Controller.

However, authentication and access control alone are insufficient. Intranet client/server applications that exchange sensitive data – for example, a payroll system – must be protected from insider abuse. Ethernet

LANs are a broadcast medium. Any PC on the LAN can capture traffic passively without detection. Using readily available hacker tools, insiders can easily perform MitM attacks on cleartext LAN traffic, modifying and inserting packets.

Companies that trust Ethernet LANs need to reexamine this policy when adding wireless LANs (WLANs). WLAN access points are often incorrectly deployed behind the corporate firewall, treating all stations on the WLAN as trusted. Doing so is a blanket invitation to intruders. WLANs based on IEEE 802.11b WiFi broadcast radio signals hundreds of feet in every direction - even beyond the physical premises. Furthermore, WiFi shared key authentication and Wired Equivalent Privacy (WEP) encryption often go unused because they are difficult to administer and have serious flaws.

As a result, visitors in the lobby or a "war driver" in the parking lot can easily use freeware like NetStumbler or AirSnort to discover a WLAN. By recording packets with WEPCrack, hackers can break WEP keys and decipher WLAN traffic. At that point, the WLAN becomes vulnerable to the same Ethernet LAN attacks previously discussed. If the wireless access point is inside the firewall, nothing stands between the intruder and the corporate network.

Tunneling with Secure Shell can protect corporate Intranet traffic by defeating WLAN exploits like AirSnort, NetStumbler, and WEPCrack, as well as passive eavesdropping and active MitM attacks that can be performed on any unprotected LAN. Furthermore, combining Secure Shell with proper placement of the wireless access point and a single access rule on the corporate firewall can prevent would-be intruders from penetrating the corporate network.

Tunneling To Shared Resources

Today, many companies share networked resources. File shares on UNIX servers are mounted on remote systems using the Network File System (NFS) and SAMBA protocols. Databases like Microsoft Access and SQL Server interface with ODBC drivers to answer queries issued by ODBC clients. Users remotely access Concurrent Versioning System (CVS) source code repositories using terminal emulators and GUI front-ends like WinCVS.

Each shared resource is a business asset that must be protected from Denial of Service (DoS) attacks, loss, malicious modification, and unauthorized access. OS security measures – Windows and *NIX file system read/write privileges, usernames, and passwords – control access. However, they do nothing to preserve data privacy and integrity when shares are accessed remotely.

A common example is the corporate teleworker with cable modem Internet access. A teleworker that uses the built-in Client for Microsoft Networks to share files between home and office PCs unwittingly exposes these shares to every neighbor on the same cable passing. Because cable is an "always on" technology, would-be attackers have plenty of time to perform a dictionary attack, discovering share user names and passwords. Thus armed, the attacker can break into shares and servers on the corporate network that are accessible with the same credentials.

Another resource shared or accessed remotely is the home or office desktop. Screen sharing can be accomplished with remote control software like TeamViewer, GoToMyPC, Windows Quick Assist, Microsoft Remote Desktop client, and RDP (Terminal Services). Unauthorized remote control has long been a security concern for enterprise administrators. Because these solutions are free/inexpensive and easy to deploy, workers install them for convenience without first addressing the inherent risk to their computers and the network.

Secure Shell tunneling can provide strong uniform authentication, access control, and privacy for shared files and desktops. Instead of leaving RDP or VNC ports open for exploit, tunneling multiplexes these nonsecure streams onto a single Secure Shell session. User credentials can be checked and access granted at the one place completely under the enterprise administrator's control: the Secure Shell server.

How Secure Shell Tunneling Works

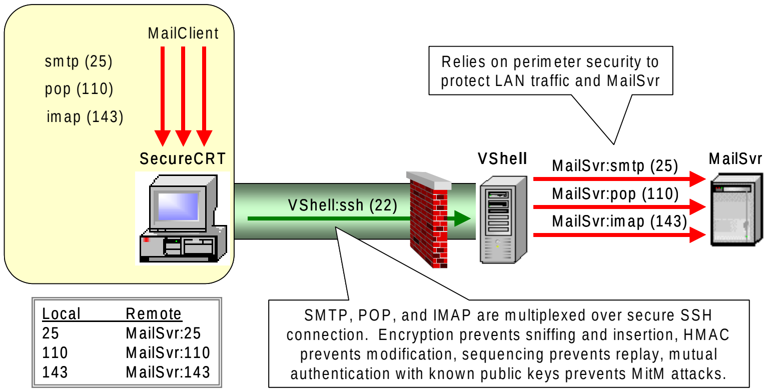
Application streams are tunneled over Secure Shell by forwarding individual TCP ports. In this paper, we focus on *local port forwarding*: tunnels initiated by the Secure Shell client. This direction is far more common than *remote port forwarding*: tunnels initiated by the Secure Shell server (see Appendix A).

When a local port is forwarded, SecureCRT (the Secure Shell client) listens to a specified TCP port on the local host. VShell (the Secure Shell server) opens a TCP connection to the remote host where the server application is actually running. By convention:

- The *localhost* refers to the application client's host; *remotehost* refers to the application server's host. Typically, if *localhost* is not specified, it defaults to the SecureCRT host. If *remotehost* is not specified, it defaults to the VShell host.

- The *localport* refers to the port that the application client sends to and SecureCRT listens to. The *remoteport* refers to the port that VShell sends to and the application server listens to. In most cases, the *localport* can be any arbitrary, unused port on the *localhost*. The *remoteport* must be the IANA-assigned "well-known" listening port for the application being tunneled.

To use the port forward, the client application must be reconfigured to connect to *localhost:localport* instead of *remotehost:remoteport*. Packets sent by the client to *localhost:localport* are intercepted by SecureCRT or another SSH client, encrypted, and tunneled through the Secure Shell connection to VShell or another SSH server. On receipt, VShell decrypts these packets, relaying them as cleartext through the TCP connection to the server at *remotehost:remoteport*. Local port-forwarding for e-mail is illustrated in Figure 2.



Relies on perimeter security to protect LAN traffic and MailSvr

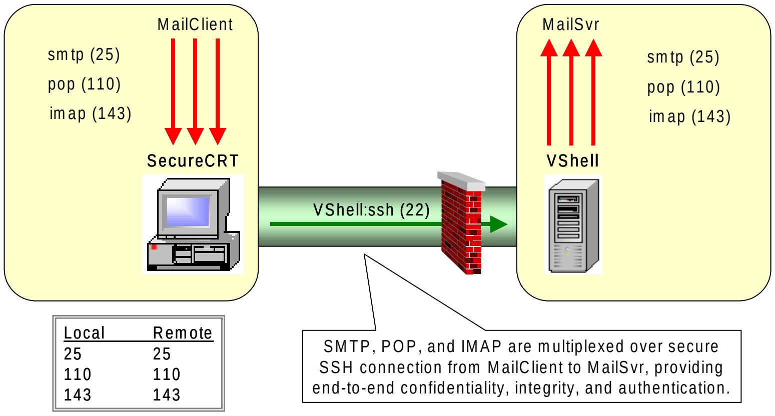
Local	Remote
25	MailSvr:25
110	MailSvr:110
143	MailSvr:143

SMTP, POP, and IMAP are multiplexed over secure SSH connection. Encryption prevents sniffing and insertion, HMAC prevents modification, sequencing prevents replay, mutual authentication with known public keys prevents MitM attacks.

Figure 2: Local Port Forwarding

Traffic in transit between SecureCRT and VShell is cryptographically protected. However, traffic between VShell and the remote host is not. Typically, VShell is located inside the network perimeter, behind a firewall. The firewall is configured to permit Secure Shell, but not the tunneled application protocols (in this example, SMTP, POP, and IMAP). In essence, this configuration relies on the firewall to protect cleartext traffic and inside servers on the trusted LAN.

When the LAN cannot be trusted or Intranet servers are at a premium, VShell can run on the same machine as the server application (see Figure 3). In this case, there is no need to specify a remote host in the port forward – SecureCRT and VShell interact with client/server applications on each local host. Application packets are protected end-to-end; cleartext is never sent over the network.



Local	Remote
25	25
110	110
143	143

SMTP, POP, and IMAP are multiplexed over secure SSH connection from MailClient to MailSvr, providing end-to-end confidentiality, integrity, and authentication.

Figure 3: Local Port Forwarding to Application on VShell Server

Local port forwarding is appropriate when SecureCRT is running on the same PC as the client application, initiating outbound TCP connections to the server application. Occasionally, users need to accept TCP connections initiated in the reverse direction by an application on the Secure Shell server-side. This can be accomplished with remote port forwarding, described in Appendix A.

These examples illustrate the broad power and flexibility of Secure Shell tunneling. But it is also important to bear in mind:

- Secure Shell forwards individual TCP connections, but not port ranges. Multi-connection applications like FTP that use ephemeral ports do not lend themselves well to port forwarding. To transfer files securely over Secure Shell, it is better to use SFTP or SCP protocols, supported by VanDyke VShell server, SecureFX® file transfer client, and the SecureCRT VCP utility.

- Although conceptually possible, standard Secure Shell does not forward UDP datagram services. However, RPC-based UDP protocols like NFS can be tunneled over Secure Shell using freely available extensions like SNFS.

Authentication And Access Control

In each of these examples, a perimeter firewall protects VShell. Leaving the Secure Shell port open on the firewall effectively delegates control over tunneled applications to VShell. Doing so creates a single, integrated point of control over remote user authentication, resource access rights, and tunneled applications.

Before any tunneling can occur, the SecureCRT user is authenticated by VShell, combining strong two-factor and public-key methods with Windows workgroups, computers, and user accounts. It also enforces authentication retry and timeout limits.

VShell filters can be created to allow or deny Secure Shell connections from individual IPs, hosts, subnets, or entire domains. Windows users and groups can be given access to local or remote port forwarding without granting command shell or SFTP privileges. Forwarded hosts and ports can be controlled at more granular levels by creating filters that allow or deny forwarding to IPs, hosts, subnets, domains. For example, forwarding can be allowed to/from *.corp.com, for any port or selected ports.

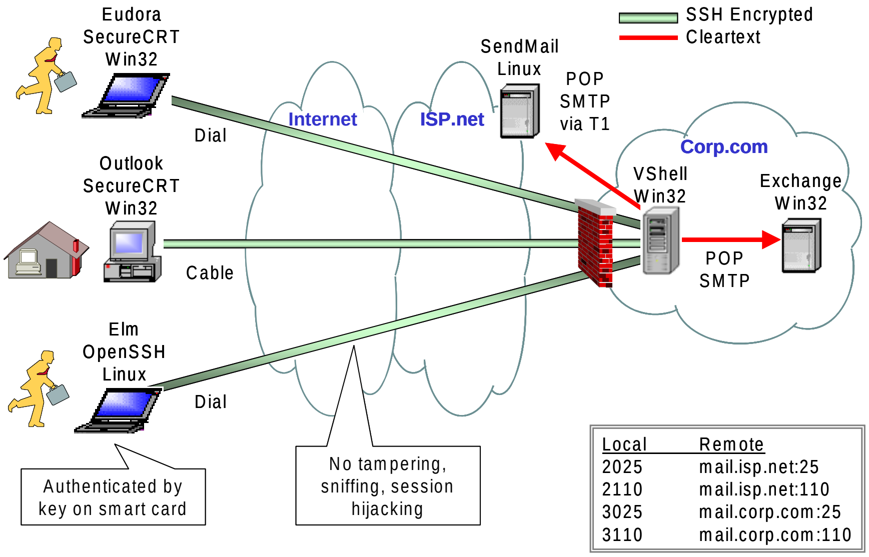
Port forward mappings are actually defined by each Secure Shell client. When a Secure Shell connection is established, VShell accepts or rejects the requested port forwards, based on the authenticated user’s privileges and port forward filters. By default, SecureCRT allows port forwarding to and from the localhost, but these client-side Access Control Lists (ACLs) can also be customized.

To more fully appreciate how port forwarding is configured, where authentication and encryption occur, and the threats addressed by these measures, let's take a closer look at some common applications that can be tunneled over Secure Shell.

Secure Email For Travelers And Teleworkers

Travelers who access email from a hotel or conference center and teleworkers accessing email from home over residential broadband need to secure POP and SMTP. Failing to do so, workers can inadvertently disclose sensitive and confidential data, including user names, passwords, message text, and attachments. Legitimate messages can be recorded, modified, and replayed to others, with consequences ranging from embarrassing to disastrous. Tunneling email is an easy way to ensure the privacy and integrity of all mail sent and received by authenticated workers through company POP, IMAP and SMTP servers.

To tunnel email, each worker configures a SecureCRT or other Secure Shell client with a local port forward; mapping ports on the *localhost* to the well-known ports listened to by mail servers. Figure 4 illustrates this, expanding on the local port forwarding configuration described in Figure 2.



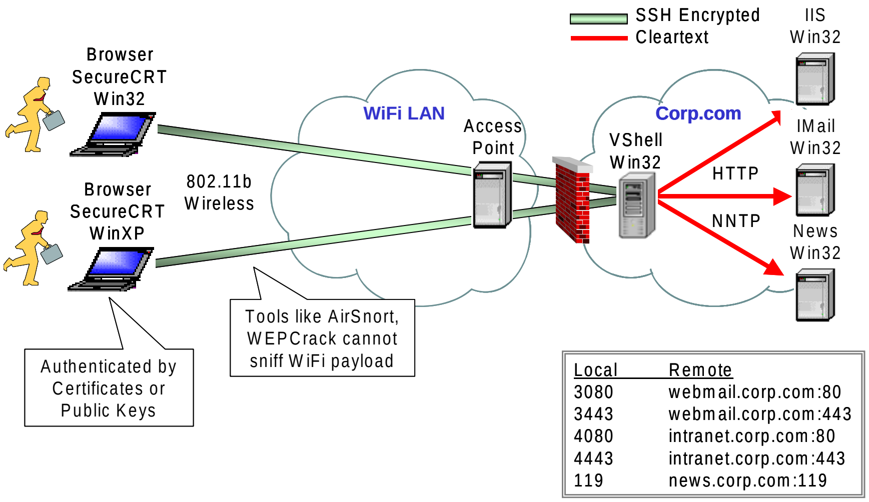
Local	Remote
2025	mail.isp.net:25
2110	mail.isp.net:110
3025	mail.corp.com:25
3110	mail.corp.com:110

Figure 4: Secure Email for Travelers, Teleworkers

Two alternatives are illustrated here. An external SendMail server that is located at this company's ISP is reached through arbitrary local ports *2025* and *2110*. An internal Exchange server within the corporate network is reached through local ports *3025* and *3110*. In both cases, mail traffic is protected on the public Internet, but forwarded as cleartext to the mail server. This prevents eavesdropping, modification, and session hijacking as e-mail passes through the public Internet. Only authenticated users can gain access to these mail servers (in this example, key pairs stored securely on smart cards). Users should know VShell's host public key in advance to be confident that they are reaching an authentic destination.

Secure Wireless Access To Corporate LANs

Figure 5 expands on a scenario described earlier in this paper: securing WLAN traffic destined for intranet servers on the corporate LAN. Employees using WiFi-enabled laptops in a conference room, cafeteria, or other public space can increase business efficiency by accessing their company's internal network resources. To prevent sniffing by AirSnort or WEPCrack, each laptop uses SecureCRT to forward ports on the *localhost* to ports *80 (HTTP)*, *443 (SSL)*, and *119 (NNTP – News)* listened to by these servers.



Local	Remote
3080	webmail.corp.com:80
3443	webmail.corp.com:443
4080	intranet.corp.com:80
4443	intranet.corp.com:443
119	news.corp.com:119

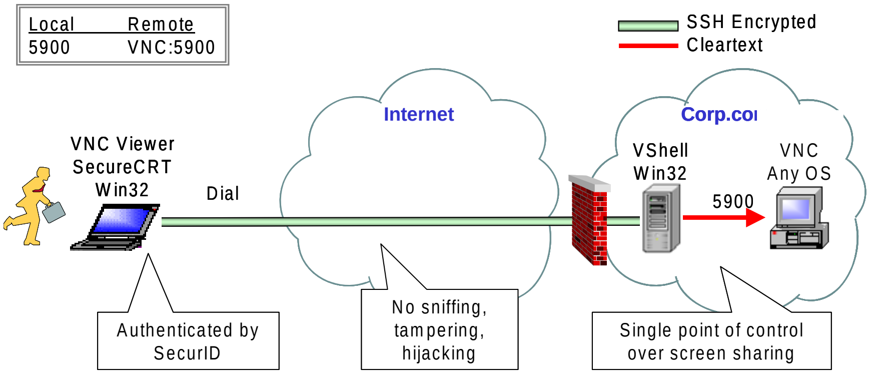
Figure 5: Secure Wireless Access to Corporate LANs

An IMail server with browser-based mail access is reached with the URL *http://localhost:3080*. An IIS server is reached with the URL *http://localhost:4080*. In this example, different local ports are assigned to forward the same application to different remote hosts. Because we have just one NNTP server, we can simply map local port *119* to remote port *119*. As the user navigates these server's web pages, only URLs relative to forwarded hosts (*webmail.corp.com* and *intranet.corp.com*) will be accessible.

Since HTTP can be encrypted with SSL (*443*), why tunnel this over Secure Shell? In this example, only users with known public keys (including those extracted from laptop certificates) may access these Intranet servers. The firewall between the 802.11b Wireless Access Point (WAP) and VShell protects the corporate LAN from the WLAN. Therefore, the only wireless traffic that can penetrate this LAN are authenticated, authorized applications tunneled over Secure Shell. On the other hand, simply opening 443 on this firewall would give any application a free ride into the LAN through this port, reaching any destination without authentication. Finally, multiplexing applications over Secure Shell reduces the total number of TCP connections, optimizing firewall performance.

Secure VNC Screen Sharing

VNC stands for Virtual Network Computing. VNC is a remote display system which allows you to view a computing desktop environment not only on the machine where it is running, but from anywhere on the Internet and on a wide variety of operating systems. Figure 6 illustrates secure VNC screen sharing, implemented through SecureCRT local and remote port-forwards. This traveler uses a VNC viewer on his laptop to remotely control his desktop back at the office. To do so, he creates a local port-forward, mapping port *5900* on the *localhost* to *5900* on the remote desktop running VNC.



Local	Remote
5900	VNC:5900

Figure 6: Secure VPN Screen Sharing

Although there are many programs that enable screen sharing, VNC is convenient because it runs on multiple platforms: Windows, Linux, UNIX, and Mac. Because VNC provides only weak logon password authentication and no encryption, tunneling VNC over Secure Shell is critical. Using Secure Shell products like SecureCRT and VShell give the network administrator granular control over remote screen sharing. Workers can be strongly authenticated with public keys, certificates, or two-factor authentication methods like SecurID. VShell filters control which desktops can be accessed through VNC ports, and which workers have permission to do so. The firewall can block VNC ports, while allowing Secure Shell to reach the VShell server. The VShell server acts as a single point of control over VNC access to this corporate network.

Security Implications

In addition to those benefits already discussed, tunneling over encrypted Secure Shell protects against IP spoofing (attackers masquerading as legitimate hosts by using a known IP address), DNS spoofing (forged DNS records that trick clients into connecting to an attacker's own server), and IP source routing (a method used by hackers to pretend that arriving packets originate from elsewhere).

No security measure – including Secure Shell tunneling protects against every possible attack. As these examples illustrate, end-to-end security involves not just protecting data in transit, but system security at the tunnel endpoints (SecureCRT and VShell), firewalls, and on any trusted server receiving forwarded cleartext. For this reason, locking down the Secure Shell server platform is essential. If a hacker penetrates a misconfigured firewall, then exploits a weak administrator password to log onto the Secure Shell server, secure tunneling cannot prevent application data from falling into the wrong hands.

When outfitting travelers, teleworkers, or partners with Secure Shell clients, document "best practices" that must be used. For example, most Secure Shell clients let the user accept and save the server's host public key on first access. This may be convenient, but doing so blindly is wrong. SecureCRT displays the host key "fingerprint." Users should be instructed to visually verify this string before accepting any unknown host key. Alternatively, supply users with host keys in advance, instructing them never to accept an unknown host key.

Permitting encrypted Secure Shell tunnels through the corporate firewall means that the firewall can no longer inspect the forwarded application data. Each company must assess the benefits and risks of Secure Shell tunneling. As discussed previously, the firewall is delegating responsibility to the Secure Shell server. If implemented correctly, this has its advantages. Content inspection products – especially e-mail and web anti-virus scanners – can be deployed on the Secure Shell server, application server, and/or client. If content inspection at the firewall is mandated by company security policy, the Secure Shell server can also be placed on a firewall DMZ or sandwiched between two firewalls.

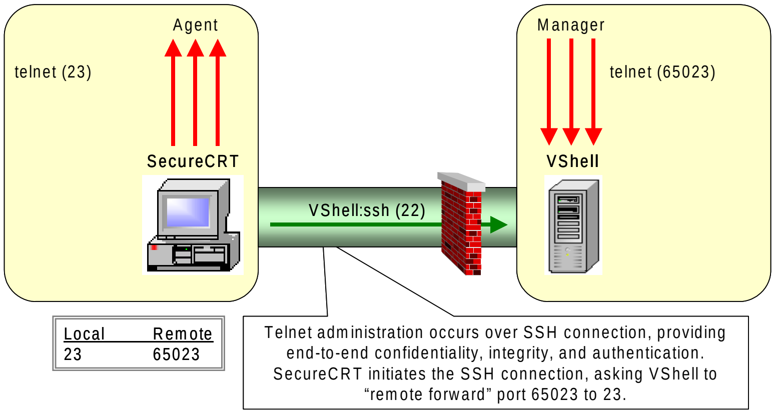
Conclusion

Compared to other link, network, and application security measures like IPsec, WEP, and PGP, installing and configuring Secure Shell is relatively quick and easy. By deploying VShell and SecureCRT, companies create a comprehensive general-purpose tunneling platform that can be used to implement a wide variety of security policies, ensuring the privacy, authenticity, and integrity of many different applications. This paper illustrates several common business applications, but the possibilities are endless. Anyone using a client to reach a single TCP port on a single remote server should seriously consider tunneling this application over Secure Shell.

Appendix A: Remote Port Forwarding

Remote port forwarding may be used if there is a need for applications to connect, through the Secure Shell server, to an application that resides on the Secure Shell client-side.

When a remote port is forwarded, SecureCRT (the Secure Shell client) requests that VShell (the Secure Shell server) listen to an arbitrary, unused TCP port on the Secure Shell server. When a connection is requested to this port on the Secure Shell server, the Secure Shell server opens another port to the Secure Shell client to relay the forwarded traffic. Packets received at *remotehost:remoteport* are intercepted by the Secure Shell server and re-directed to the Secure Shell client at *localhost:localport*.



Local	Remote
23	65023

Telnet administration occurs over SSH connection, providing end-to-end confidentiality, integrity, and authentication. SecureCRT initiates the SSH connection, asking VShell to "remote forward" port 65023 to 23.

Figure 7: Remote Port Forwarding

In this case, forwarded traffic can be seen as "flowing" between some independent client (the application that accesses the reverse-forwarded port), the Secure Shell server (*remotehost*), the Secure Shell client (*localhost*), and a destination server (the application that consumes the reverse-forwarded data). Figure 7 illustrates remote port forwarding to a Telnet server on the *localhost*.

With remote port forwarding, the server application is typically co-located with SecureCRT. The server can also run on a trusted host near SecureCRT – for example, a SOHO LAN gateway that is remotely administered through Telnet. When configuring remote port-forwards, unique listening ports must be assigned to each SecureCRT. In Figure 7, VShell can forward Telnet sessions to several different SecureCRTs – provided that each uses a different remote port.