Security Implications of SSH

Bill Pfeifer

Security Implications of SSH

GSEC Certification Version 1.4b (amended 29 August 2002)

By William Pfeifer
18 April 2003

Abstract

As an updated and secured version of older protocols such as Telnet and FTP, SSH is being installed and enabled on an increasing number of devices, from personal desktops to firewalls. There has been significant discussion on the whys and wherefores of the installation of SSH, the benefits of SSH over earlier protocols, and even how to set up tunneling to secure communication for other protocols via SSH. I have thus far seen little discussion on the potential security exposure offered by SSH and how to mitigate that exposure. This paper was written to provide a high-level discussion of some of the security considerations associated with SSH as well as some potential methods of addressing those considerations.

SSH is a useful protocol, but to be a true benefit to an organization, it must be implemented with some care. Evaluation of where in an architecture it is installed as well as some configuration suggestions can significantly increase the security of the overall implementation.

A Brief History of SSH

SSH-1 was created in 1995 by Tatu Ylonen, a university researcher in Finland, in response to a password-sniffing incident on his university network. He released SSH later that same year as freeware, and included the source code. At the end of the year, SSH Communications Security, Ltd. (SCS) was founded as a result of the widespread usage of the new SSH protocol suite. In 1996, SCS released SSH-2, a new and improved version of SSH-1 that corrected some deficiencies in the original product. However, SCS issued SSH-2 as a commercial product and not as freeware; it came with a more restrictive license than SSH-1 (SCS has since loosened the licensing restrictions). It also lacked some of the features of SSH-1 and was not backward compatible with the earlier protocol. As a result, SSH-1 is still more widely deployed than SSH-2. [1]

A common variant of SSH-1 and SSH-2 is OpenSSH, developed by the FreeBSD group from a derivative of an early version of SSH-1. OpenBSD v2.6 included the first release of OpenSSH (v1.2.2) in December 1999. OpenSSH v2.0, which shipped with OpenBSD v2.7 in June of 2000, included both SSH-1 and SSH-2 support in a single program. [2] OpenSSH does not carry the licensing restrictions included with the SSH-1 or SSH-2 protocols as published by SCS.

The OpenSSH site currently supports a server running the ScanSSH program written by Niels Provos. This server polls random IP addresses on the internet to scan for active instances of SSH. When it locates such a server, it records the version of both the SSH server and the SSH protocol that are running and then disconnects from the machine. The latest scan by the OpenSSH site's ScanSSH server was tallied in April of 2002; of 2.4 million SSH servers polled, 59% were running OpenSSH v1.99. The next most popular version was SSH v1.5 at 17.9%.

Total use of OpenSSH servers (versions 1.5, 1.99, and 2.0) was 66.8%. Total use of SSH servers (versions 1.3, 1.5, 1.99, and 2.0) was 28.1%. [3]

Please note that for the purposes of this paper, the term “SSH” refers to the protocol in general, unless a specific implementation is stated.

Convention

In a paper with as many examples as this one, those examples become cumbersome (not to mention boring) if the author consistently refers to “the system administrator”, “a given system”, “a network’s end users” and other such generic terms. For purposes of the examples in this paper, we will assume that we all work for InsecurCo. Our system administrator’s name is Bradmin, who reports directly to a VP who we’ll call Bossman. There is a malicious person trying to gain access to InsecurCo’s systems and/or private data whom we’ll name Johnny Hackster. Authorized users of InsecurCo’s network systems are Mary and Carlo. And let’s call the main file servers Server_1 and Server_2 for the sake of simplicity.

The Situation

As security awareness grows, more and more administrators and users are turning to SSH as an effective and easy way to secure their communications via encryption. In many cases, SSH tunnels are set up to encrypt protocols that do not include any encryption of their own; common examples of this are remote control of servers and desktops via VNC over SSH tunnels (see Appendix A for a sample setup) or remote database access utilizing SQLNet via SSH tunnels.

While SSH provides vastly improved security over the protocols that it was written to replace (specifically the “r” commands such as rlogin and rsh, but also Telnet and FTP), the unfortunate reality is that SSH is not a “silver bullet” capable of removing all dangers. Known exploits of SSH exist that can be used as attack vectors against a network. For example, the US Department of Energy put out notice in 2001 that “… many servers running SSH have been compromised…” via an exploit that allowed attackers to gain root-level access to vulnerable servers [5]. More recently, CERT updated a list of vulnerabilities with the key exchange and initialization processes for SSH implementations from multiple vendors. “In some cases remote attackers could execute arbitrary code with the privileges of the SSH process… Attackers could also crash a vulnerable SSH process, causing a denial of service.“ [6] Other such vulnerabilities exist, and more will be found as time goes on and attackers find new ways to penetrate secure systems.

As a further danger, SSH port-forwarding can be used to effectively bypass security measures. If Mary is permitted SSH access to Server_1, then a default installation of SSH will also allow her to set up port-forwarding tunnels. Because SSH port forwarding occurs through an SSH-encrypted tunnel, port-forwarded traffic will bypass most firewalls and/or IDS systems in the path of the tunnel.

Let's say for this example that Mary has set up an SSH session from her home PC that includes port forwarding through Server_1 to port 5900 (the standard VNC port) of Carlo's desktop at work. She fires up her VNC session and now has direct control of Carlo's work machine from her home network. That connection has bypassed all of InsecurCo's firewalls, which were set up to deny VNC sessions from external sources.

If Mary has malicious intent, she could hack the corporate network from home in ways that might be very difficult to detect and/or prove if necessary. Fortunately for us, Mary doesn't have a malicious bone in her virtual body. However if she forgets to close her SSH session, then when Johnny Hackster comes over for dinner (he's really nice once you get to know him), he could surf InsecurCo's network while Mary is busy stuffing the turkey. Yet another risk is that if Mary's PC had been compromised by a Trojan Horse or virus, that could provide an alternate avenue for Johnny Hackster to access the internal corporate network.

Threat Mitigation

Obviously, the best way to secure a system is simply to eliminate all external access to it. Unfortunately, such an approach is simply not practical, and so we must in all cases strive to find the proper balance between accessibility/useability and security. Brian McKenney began his article titled "Defense in Depth" as follows:

> A primary Information Assurance (IA) objective is to protect enterprise or enclave resources through Defense in Depth (DiD). A DiD strategy combines the capabilities of people, operations, and security technologies to establish multiple layers of protection--analogous to protecting a home with multiple defenses. These defenses may include a strong lock at the front door, secured windows, an electronic home security system, bright lights on the outside, a neighborhood watch program, and a dog that barks at people who walk near the home. An intruder must circumvent these defenses in order to gain unauthorized entry to the home. With DiD, the objective is to implement defenses at multiple locations so that critical enclave resources are protected and can continue to operate in the event that one or more defenses are circumvented. Managers must strengthen their defenses at critical locations and be able to monitor attacks and react to them with a coherent response. [7]

The point to this is that good security cannot be a single-layered approach. Multiple layers must be utilized to help ensure that, even if one layer of security should fail, an intruder's progress into your network will be arrested or at least logged by the next layer of security. In terms of SSH, the following progression is recommended:

1. Evaluate whether SSH is necessary on a given system
2. Evaluate which internal and external entities should be permitted SSH access to that system
3. Patch, patch, patch
4. Run SSHD as a non-root user
5. Run SSH in a CHROOT environment
6. Use TCPWrappers for additional granularity of control

Evaluate whether SSH is necessary on a given system
SSH is a very useful tool to allow secure remote console access to systems, to securely move files between systems, and to tunnel insecure protocols between systems. However, there are many situations in which SSH does not provide any significant benefit and is nothing but unnecessary overhead on a server, and a security liability to a network.

Let's say, for example, that Bradmin has a network appliance that is maintained via an SSL-encrypted browser-based GUI. All functions of the device can be controlled from a browser, so the only time that Bradmin needs console access to the appliance is when he is troubleshooting a hardware failure, and he has a direct console connection into it then anyway. Since running SSH on the system provides no benefit for administration and it might provide an avenue that an attacker could utilize to gain access to key resources, he simply disables SSH on the appliance.

More common would be a situation in which a device does require SSH but port forwarding is not utilized. In a case where SSH does provide a clear benefit for administration of the system but port forwarding is either not necessary or is not authorized, SSH should be run with port forwarding disabled. To do this for OpenSSH or SSH-1, you need to set the "no-port-forwarding" option on SSHD, the SSH daemon. I have thus far been unable to locate an equivalent option for SSH-2; this may require an alternate method of blocking port-forwarding if such is available.

Sample Man pages for the main versions of SSH can be located at the following sites:
SSH-1: http://www.employees.org/~satch/ssh/faq/manpages/sshd1_man.html
SSH-2: http://www.ssh.com/documents/32/sshd2man.txt
OpenSSH (current): http://www.openbsd.org/cgi-bin/man.cgi?query=sshd

Note that for earlier versions of OpenSSH, simply select the appropriate version of OpenBSD from the drop-down menu on the site referenced above.

Evaluate which internal and external entities should be given access to SSH on that system

Once you have made the decision that a system should, in fact, have SSH running, you should then look at what hosts require access to SSH on that machine. Many security administrators allow open access to SSH servers on port 22 (the standard SSH port) regardless of whether or not such open access is required because they consider SSH to be a “secure” protocol.

In many situations, SSH access can be restricted to just a few hosts rather than leaving access open to the world. For example, Bradmin installs SSH on Server_2; since external entities require access to Server_2 via SSH, he opens up port 22 (SSH) on his firewall to the world. A week later, Bradmin runs an external security audit on his network and finds that Server_2 is vulnerable to a new SSH exploit. He takes a moment to re-evaluate his firewall policy and realizes that SSH was installed on the system to allow for remote administration from his home office, and also so that InsecurCo’s partner company DataMiners, Inc. could run shell scripts on the server. Bradmin will patch the server as soon as he can schedule downtime, but he immediately restricts port 22 access to his home IP address and the address block owned by DataMiners, Inc. This simple step significantly reduces the exposure of the vulnerability and should allow him enough time to test the SSH patch before he has to install it on his production server.

Another option to consider is permitting external SSH access to one or two servers and allowing those servers SSH access to other internal servers; such servers are generally referred to as “bridgehead” servers. With a setup such as this, you can permit external access to those servers that are not business-critical while still providing authorized users with remote console access to critical servers via the bridgehead servers. Generally more than one server acts as a bridgehead so that users will still have access to required servers even if one of the bridgehead servers becomes unavailable due to scheduled maintenance or other causes.

Let’s say that Bradmin gets a request for SSH access to the InsecurCo financial server, Acctg. This server sits in its own secured area and for security reasons no external access has been permitted thus far. He doesn’t want to change that policy and permit external access of any sort, but this request has come from his boss Bossman who has made it clear that it is a priority item. Bradmin discusses the issues with Bossman, and they agree that allowing SSH access to Acctg from Server_2 does not pose a significant risk. When Bossman requires external access to Acctg, he will SSH in to the bridgehead server, Server_2, and then SSH again from Server_2 to Acctg. In this way, Acctg is not exposed directly to any external threats, but Bossman can freely access the box from wherever he goes.

Patch, patch, patch
If by now you are not a firm believer in keeping up with the latest patches to secure your systems, then you either have been hacked as a result or you have been very lucky (so far).

New exploits come out every day and it's up to each admin to determine which patches are critical for which systems. Exploits for services that are exposed to the open internet should be patched immediately for critical systems. Potentially compromised services on noncritical systems and/or systems with limited exposure may be able to wait for the next scheduled maintenance cycle if necessary, but always with an eye toward risk assessment. A non-critical system that gets taken over by a hacker might expose the heart of your network to malicious activity, and so might prove to be a larger liability than the single system would indicate.

An InsecurCo example of the above might involve Bradmin's SSH bridgehead server, Net_Console. Net_Console serves as the corporate SSH bridgehead server, allowing SSH access to InsecurCo from users all over the world. It sits in its own dedicated DMZ and has no business-critical processes or data resident on the server. In short, it has been set up as an SSH server that can be taken offline with relatively little impact. As new SSH vulnerabilities are found and patches released, the server is updated, but not very aggressively because the server is considered to be expendable.

Johnny Hackster comes along and notices that Net_Console has an open exploit in its SSH daemon. He already has an attack script that he downloaded from the internet and has been waiting to try out, and he thinks this is just the time. The attack script works, and Johnny Hackster has a root shell on Net_Console.

No big deal, right? Only if Net_Console has no access to any other InsecurCo servers, which would make it useless as a bridgehead server! If Net_Console is actively used as a bridgehead server, it probably has access (via SSH or other protocols) to almost every server InsecurCo runs, and pretty soon Johnny Hackster will too.

Any exposed server, and especially the bridgehead servers, must be kept up to date on patches for any exposed service. Some servers may be left unpatched because they are considered to be "expendable", but remember that those servers could be used as just another hop for an attacker to get into your network.

Run SSHD as a non-root user
In some cases, administrators may be able to restrict the access that SSH has to their systems. One possible method of restriction is running the SSH daemon (SSHD) as a non-root user.

Note that running SSHD as a non-root user creates some limitations with SSHD and so is not a very popular method of securing SSH. I mention it here because in certain situations it can be very useful to restrict access where only limited access is required.

By default, most daemons run as root; this provides them with all the rights that they could possibly require to a given system to ensure smooth operation. The unfortunate side effect of this is that, when that process is taken over by an attacker, the attacker may gain root-level access to your system. A process running as a non-root user with restricted system access, however, does not pose nearly so great a security risk in case of a program exploit. If a process is run as a user who has rights only to what that process requires, then an attacker taking over that process must continue looking for further avenues to gain full system access.

SSHD will start and run as a non-root user without issue. However, SSHD running as a given non-root user will only accept connections from that user; the daemon will not accept connections from other users. [8] There has been some discussion regarding the use of public key authentication to allow SSHD to run as a non-root user. The suggestion is that SSHD running as a non-root user would be unable to access /etc/passwd (the password file) for authentication. [9] This might be an interesting topic for further research, but is beyond the scope of this paper.

Run SSH in a CHROOT environment

A CHROOT environment for a process is frequently referred to as a “jail” or a “sandbox”. The CHROOT program is used to create a virtual system root at the specified directory. The purpose of this is to restrict access to a specific portion of the filesystem. A user or process running in a CHROOT environment cannot access files outside of the designated portion of the file tree because there is no way to address them – file system calls have no method of specifying a directory higher than the perceived root of the file system.

One word of caution about using CHROOT environments: since no access outside of the newly defined root directory is possible, any utilities, programs, scripts, or other files that are required must be accessible from the logical “root” directory.

For example, Bradmin has decided that Carlo only requires access to the files in his home directory (/home/carlo) and nothing else on the system. When Carlo logs in to the system, his login script includes the following:

chroot /home/carlo

When the chroot command runs, it redefines root (/) as being located at /home/carlo. As a result, Carlo can access any files located in his home directory

or any of its subdirectories, but cannot access anything outside of his home directory. For Carlo, any files outside his home directory are “above” the root directory. Note that any utilities such as ls, vi, or less that Carlo requires must be accessible from within his CHROOT environment or they will not function.

SSH-2 has its own helper application called CHROOT Manager to handle restricting users to their own home directories when they use SSH. They also include instruction on how to perform the same functions manually. [10]

There is also a SourceForge project related to setting up a CHROOT environment for SSH users. It can be found at http://chrootssh.sourceforge.net/. The project involves adding a patch to OpenSSH to allow the performance of CHROOT without simply adding the command to a shell script (as in the example above), which is relatively simple for an experienced attacker to circumvent. [11]

There is also a good overview of setting up various CHROOT environments at http://www.networkdweebs.com/chroot.html. [12]

Use TCPWrappers for additional granularity of control

TCPWrappers was written by Wietse Venema in 1990 to defend a Dutch university against a malicious hacker. It functions as a drop-in replacement for the inetd daemon, which handles incoming requests for such common network services such as FTP, Telnet, and Finger. Requests for controlled services come in to inetd (or tcpd, the TCPWrappers daemon), which then calls the appropriate network service daemon to respond to the request.

Since inetd has no security or logging functions built into it, Mr. Venema wrote TCPWrappers to replace the functions of inetd with the addition of tracking and security functions so that he could monitor and/or halt a hacker’s progress through his university’s network. [13]

TCPWrappers offers administrators the option of allowing or denying connections to many common network services via the /etc/hosts.allow and /etc/hosts.deny configuration files. Further, in those files can be specified additional shell commands to be run when certain criteria are met. This feature allows for some additional tracking of a suspected intruder by means of finger or other such programs. Finally, TCPWrappers can log its activity via the standard syslog service. [14]

One of the most significant advantages to using TCPWrappers from a security standpoint is that the tcpd daemon exchanges data with neither the network client nor with the server process. [13] This makes tcpd completely transparent to authorized users, who simply gain access to the network service as requested. Unauthorized users will be denied access to the requested service, but cannot directly address tcpd, and so have very little chance of successfully exploiting the service.

TCPWrappers is included in the default installation of current versions of Red Hat linux; configuration information can be found online at http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/ref-guide/ch-tcpwrappers.html. [15] SSH-2 also has setup information available at http://www.ssh.com/support/documentation/online/ssh/adminguide/32/Configuring_SSH_Secure_Shell_for_TCP_Wrappers_Support.html. [16]

Let's say that Bradmin has decided to set up TCPWrappers on his SSH bridgehead server, Server_2. Since it's a bridgehead server, he decides to leave it as a "mostly open" configuration, which means that his default is to permit connections through unless they're explicitly denied. He knows of a few folks who have actively tried to hack their way in before, such as Johnny Hackster, and so he'll block their known IP addresses via the /etc/hosts.deny file and keep watch to see what else he needs to deny.

Mary is traveling for work, and needs to access Server_2 via SSH. She sends her SSH request to Server_2. Tcpd, the TCPWrapper daemon, intercepts the request. It checks its access lists, doesn't see her on the deny list and so lets her pass. It hands off the request to sshd for normal processing. The whole procedure is completely transparent to Mary.

Later that evening, Johnny Hackster checks out the server. Since his IP address is on the deny list, the request comes in to tcpd and is not accepted. Tcpd does not call sshd to handle the connection, and Johnny Hackster is denied. Another victory for InsecurCo.

Summary

Reasonable steps must be taken in any SSH implementation to determine first whether SSH should be run, and then to ensure that it is as secure as possible. The following is a logical progression to follow with any implementation of SSH.

1. Evaluate whether SSH is necessary on a given system
2. Evaluate which internal and external entities should be permitted SSH access to that system
3. Patch, patch, patch
4. Run SSHD as a non-root user
5. Run SSH in a CHROOT environment
6. Use TCPWrappers for additional granularity of control

SSH is an invaluable protocol, and a great improvement over the old "r" commands that are its predecessors. It helps protect against snooping, and can provide an encrypted tunnel for less secure protocols. However, it is not a panacea to fix all security risks, and it must be implemented with some level of caution. Use it wisely as part of a comprehensive security plan.

References

[1] Barrett, Daniel J. and Silverman, Richard E. SSH – The Secure Shell. Sebastopol: O'Reilly & Associates, 2001. 10-12

[2] "OpenSSH – Project History and Credits". From the OpenSSH web site. http://www.openssh.org/history.html

[3] "ScanSSH – SSH Usage Profiling". From the OpenSSH web site. http://www.openssh.org/usage/index.html

[4] "ScanSSH – Statistics from the Current Scan Results". From the OpenSSH web site. http://www.openssh.org/usage/ssh-stats.html

[5] "CIACTech02-001: Understanding the SSH CRC32 Exploit". US Department of Energy Computer Incident Advisory Capability Technical Document number UCRL-MI-119788. 9 May 2002. http://www.ciac.org/ciac/techbull/CIACTech02-001.shtml

[6] "Vulnerability Note VU#389665 - Multiple vendors' SSH transport layer protocol implementations contain vulnerabilities in key exchange and initialization". Computer Emergency Response Team Coordination Center. 20 March 2003. http://www.kb.cert.org/vuls/id/389665

[7] McKenney, Brian. "Defense in Depth". February 2001. http://www.mitre.org/pubs/edge/february_01/mckenney.htm

[8] anne@ipsec.com and satch@employees.org. "The Secure Shell (SSH) Frequently Asked Questions – Section 3.3. Does it make sense to install SSH as a non-root user on UNIX?". November 1999. http://www.ldeo.columbia.edu/NETWORK/ssh/ssh-faq-3.html

[9] Friedl, Markus. Message from the Neohapsis OpenBSD mailing list. April 2002. http://archives.neohapsis.com/archives/openbsd/2002-04/0239.html

[10] "Using Chroot Manager (ssh-chrootmgr)". From the SSH Communications Security Corp web site. http://www.ssh.com/support/documentation/online/ssh/adminguide/24/Using_Chroot_Manager__ssh-chrootmgr_.html

[11] james@firstaidmusic.com. From the SSH CHROOT project on SourceForge. http://chrootssh.sourceforge.net/

[12] jonathan@networkdweebs.com. “Chrooting daemons and system processes HOW-TO”. 12 March 2003.
http://www.networkdweebs.com/chroot.html

[13] Venema, Wietse. “TCP WRAPPER - Network monitoring, access control, and booby traps”. 15 July 1992.
ftp://ftp.porcupine.org/pub/security/tcp_wrapper.txt.Z

[14] Arruda, Stacy M. “TCP WRAPPERS—What are they??”. 16 February 2001.
http://www.sans.org/rr/unix/TCP_wrappers2.php

[15] Red Hat Linux 9: Red Hat Linux Reference Guide / Chapter 15: TCP Wrappers and xinetd.
http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/ref-guide/ch-tcpwrappers.html

[16] “Configuring SSH Secure Shell for TCP Wrappers Support”. From the SSH Communications Security Corp online documentation.
http://www.ssh.com/support/documentation/online/ssh/adminguide/32/Configuring_SSH_Secure_Shell_for_TCP_Wrappers_Support.html.

Appendix A
SSH tunneling on MS Windows with Cygwin and Putty

When I started using VNC as a remote access solution for Windows machines, I searched for some time to find an SSH server for Windows that was free, stable, and looked like it was going to be around for a while. The only one I've found thus far that fits those criteria is Cygwin. It took me a while to create an easy installation and setup checklist, but the below has worked well for me (and my co-workers and friends) for a while now.

This text is intended as a basic checklist for setting up Cygwin and TightVNC on Windows, and establishing port-forwarded connections from Windows (via Putty) and Linux clients (via command-line SSH). I've had several people other than myself use this checklist to set up their machines without issue, so I expect that it's complete by now (until versions change).

Setup of Cygwin on the Windows server

Download and install Cygwin from www.cygwin.com. Install the base package along with the OpenSSH server/client package (under Network), the CygRunSrv program (under Administration), and Vim from the Editors section. I also generally include the cygwin-specific man pages from the Documentation section.

Go to My Computer -> Properties -> Advanced tab -> Environment Variables -> System Environment and add "CYGWIN=ntsec tty"; also append "c:\cygwin\bin" to your path.

Open a Cygwin console window and run "ssh-host-config –y". This will configure SSH to run on your server. When prompted with "cygwin=", type "ntsec tty".

To set up Cygwin as a service on your server, type "cygrunsrv -S sshd" in a Cygwin console window.

To create Cygwin users and groups based on local accounts, open a Cygwin terminal and type the following:
mkpasswd -l > /etc/passwd
mkgroup -l > /etc/group
Note that the "-l" is for Local user/group database. "-d" can be used to create users and groups based on Domain accounts, but I haven't had much luck with that thus far.

Set up TightVNC on the Windows server

Download and install TightVNC from www.tightvnc.com; a default install is fine.

From Start-> Programs-> TightVNC-> Administration-> Show Default Settings, enter your access password. Click on the Advanced button, and check "Allow Loopback Connections".

From that same page, you can also check “Allow Only Loopback Connections” if you want to restrict VNC to only accept connections that have tunneled directly to this box. If you also want to be able to VNC to this machine without tunneling, do NOT select “Allow Only Loopback Connections”.

Setting up Putty for SSH Port Forwarding for VNC connections
Download and install Putty from
http://www.chiark.greenend.org.uk/~sgtatham/putty/.

Set up a saved session to the Windows server configured above.

From the list in the left-hand panel, select SSH --> Tunnels --> Add new forwarded port
Set the source port to 5900
Set the destination IP and port to 127.0.0.1:5900
Click Add

Via the left-hand panel, go back to Session and click Save

Connect via SSH to the Windows server.

Setting up *nix for SSH Port Forwarding for VNC connections
At a command prompt, type the following:
ssh -L 5900:127.0.0.1:5900 <IP_address_of_Windows_SSH/VNC_server>

NOTE: format for the above command is: ssh -L
<local_port>:<host_IP_or_loopback>:<remote_port> <remote_ssh_server_IP>

To Connect via VNC
SSH to the remote PC via the Putty profile or SSH command line
Use your VNC client to connect to 127.0.0.1 on the default port of 5900
Because of the port forwarding that you set up above, the VNC connection will redirect through the SSH tunnel and arrive at the Windows SSH/VNC server.

Last Updated: September 25th, 2020

Upcoming SANS Training

Click here to view a list of all SANS Courses

SANS October Singapore 2020	**Singapore, SG**	**Oct 12, 2020 - Oct 24, 2020**	**Live Event**
SANS Community CTF	**,**	**Oct 15, 2020 - Oct 16, 2020**	**Self Paced**
SANS SEC504 Rennes 2020 (In French)	**Rennes, FR**	**Oct 19, 2020 - Oct 24, 2020**	**Live Event**
SANS SEC560 Lille 2020 (In French)	**Lille, FR**	**Oct 26, 2020 - Oct 31, 2020**	**Live Event**
SANS Tel Aviv November 2020	**Tel Aviv, IL**	**Nov 01, 2020 - Nov 06, 2020**	**Live Event**
SANS Sydney 2020	**Sydney, AU**	**Nov 02, 2020 - Nov 14, 2020**	**Live Event**
SANS Secure Thailand	**Bangkok, TH**	**Nov 09, 2020 - Nov 14, 2020**	**Live Event**
APAC ICS Summit & Training 2020	**Singapore, SG**	**Nov 13, 2020 - Nov 21, 2020**	**Live Event**
SANS FOR508 Rome 2020 (in Italian)	**Rome, IT**	**Nov 16, 2020 - Nov 21, 2020**	**Live Event**
SANS Community CTF	**,**	**Nov 19, 2020 - Nov 20, 2020**	**Self Paced**
SANS Local: Oslo November 2020	**Oslo, NO**	**Nov 23, 2020 - Nov 28, 2020**	**Live Event**
SANS Wellington 2020	**Wellington, NZ**	**Nov 30, 2020 - Dec 12, 2020**	**Live Event**
SANS OnDemand	**OnlineUS**	**Anytime**	**Self Paced**
SANS SelfStudy	**Books & MP3s OnlyUS**	**Anytime**	**Self Paced**