Securing Transmission with SSH

Abstract:

Guarantying secure transmission between hosts across the internet is the main object of web security providers. It has come to mutual agreement by almost all professionals that SSH- Secure Shell Protocol is the absolute alternative to Telnet and protocols like Remote Shell (RSH), reason being weak host-based authentication and transmission of clear-text passwords over the internet by Telnet and vulnerability of spoofing attacks on RSH.
The purpose of this paper is to explore the features of a secure protocol and present brief introduction to the techniques used to overcome security vulnerabilities by implementing protocols that use encryption like SSH.

General Terms: Cryptography/Encryption.

Keywords: Authentication, Port forwarding, Secure Command Shell, Secure File Transfer, Secure Tunnelling, and OpenSSH.

Introduction:

With most of the information in the world being available on the internet due to low publishing costs and wider availability a need for a protocol has arisen that provides data integrity, encryption and authentication for a network communication to be secure.
The solution was found with advent of Secure Shell protocol (will be referred as SSH all along the paper) that solves the problem of securing data over a public network.

History and Evolution Of SSH:

The SSH protocol was created by uber-hacker Tatu Ylonen a Finnish computer professional in 1995 to replace the non-secure "r-commands" of UNIX like rlogin, rsh, and rcp. The version 2 (SSH2), was then submitted as draft by the Internet Engineering Task Force in 1997. The purpose of the latter was to addresses some of the more serious vulnerabilities in SSH1 and also provides an improved file transfer solution. (1)
The problem with SSH version 1 was that it relied mainly on RSA, that encumbered (patented) technology causing any application that uses it to be licensed with the exception of noncommercial use. In September 2000 RSA's US patent expired making
Tatu earn a living. By the time RSA became less encumbered, SSH version 2.0, unlicensed/free commercial use was no longer permitted and failed to become an Internet standard, as it did not meet the requirement according to which at least one free implementation be available.
OpenBSD, is the ultra-secure offshoot of NetBSD, a free version of BSD UNIX wanted to include SSH in OpenBSD 2.6 but were cautious of SSH's various encumbrances and instead joined with the Swedish programmer Bjoern Groenvall who had released an improved version of

SSH, 1.2.12. Work on updating and adapting it for a larger audience began and OpenSSH became part of OpenBSD ever since making it portable to most versions of UNIX.
OpenSSH was built on OSSH, that was created by Groenvall and is still available, adding support for later versions of the SSH protocol and modularizing its cryptographic mechanisms in such a way that it's possible to compile OpenSSH without any patented algorithms like ssh v.1 protocols, which depend on RSA. The OpenBSD team also innovated by converting OpenSSH code-base into simple, platform-independent and a portable version, that can be compiled for both OpenBSD and a variety of versions of UNIX.
The latest innovation on the protocol was done by professionals working for Linux to make it secure and compatible for Linux machines. This was done by ensuring that the software package is fundamentally stable and secure with portability enhancements added.
Considering the quality of the end product even if everyone one was enhancing Ylonen's and Groenvall's packages it is a worth while achievement. (2)

Tools in the SSH suite: (2)

- sshd—secure shell daemon that performs server functions for all other commands
- ssh—primary end-user tool: remote shell, remote command, and port-forwarding sessions
- scp—tool used for automated file transfers
- sftp— is basically an ftp client with strong authentication and encryption embedded on it. This tool used when interactive file transfers are required i.e. it is only available in F-Secure's commercial versions of ssh version 2—I.
- ssh-keygen— private-public key pair generator to use in RSA and DSA authentication (including host keys)
- ssh-agent—Client's RSA/DSA authentications are automated using this daemon
- ssh-add—Used to load private keys into ssh-agent process
- ssh-askpass—provides ssh-add with X interface.

How does SSH perform its functions:

Secure Shell works by encrypting channels using generic “host keys” or with digital certificates that can be verified by a trusted certificate authority for example VeriSign. The cryptographical functions used by OpenSSH are much similar to the way Secure Sockets Layer’s web transactions.
The way connections are built can be summarized as follows:

1. Public host keys are exchanged between the client and the server

2. An option of accepting un-trusted key is prompted to the user if the public key was never encountered by the client machine before.
3. A session key is then negotiated that via a block cipher such as Triple-Des (3DES) encrypts all subsequent session data.
4. RSA or DSA certificates are then used by the server to authenticate the client.
5. If this fails standard username and password prompts are displayed to the client.
6. When the client is successfully authenticated the session is initiated over the encrypted tunnel. (2)

Public Key Encryption over others during Authentication:

As SSH caters authentication systems the following can be used to serve the purpose: (3)

- Password to be echoed in the script.
- Rhosts,
- SSH agents and
- Public keys.

Password:

As standard input is not read by SSH, therefore echoing password in the script is not an easy task as it requires advanced scripting techniques. Not only that the password has to be either placed in the script or a file present in the same filesystem. For a determined intruder this would be like viewing the password on the screen.

Rhosts:

In a host based authentication users are granted access by executing the rsh command. Password is not required in this scheme as access is granted or denied to the users based on the host. This is prone to an attack if an intruder pretends to be another host, where as the IP can be spoofed and the DNS can be disrupted. This is usually carried out where security concerns are light.

SSH agents:

It uses the scheme of public key cryptography instead of using plain text password. Upon connection the client proves its identity to the server based on algorithms.
The following steps show how a set of public and private SSH keys are generated to permit user named scripts to log in from hostA to hostB, assuming the user exists on both hosts:

1) **Generate the keys:**

```
[scripts@hostA]$ssh-keygen  -t dsa
Generating public/private dsa key pair.

Enter file in which to save the key (/home/scripts/.ssh/id_dsa):
Created directory '/home/scripts/.ssh'.
Enter passphrase (empty for no passphrase): XXXX
Enter same passphrase again: XXXX
Your identification has been saved in /home/scripts/.ssh/id_dsa.
Your public key has been saved in /home/scripts/.ssh/id_dsa.pub.
The key fingerprint is:
```

41:03:aa:dc:cc:b9:39:50:65:bc:ee:7b:36:d2:64:7a scripts@hostA

2) Copy public key to hostB from hostA:

scp /home/scripts/.ssh/id_dsa.pub \
hostB:/home/scripts/.ssh/authorized_keys

Coping of files by scp is similar to that of rcp but as the files are encrypted the procedure is carried out in a secured manner.
Users that can log in to the account by using public key authentication use authorized_keys file containing the public identities or public keys. Every user maintains their own personal authorized_keys file, that resides in the hidden .ssh directory of the user's home directory. Security restrictions to public keys can be confiqured here by the users.
When ssh-keygen is run for the first time to create public and private keys it does not create the authorized_keys file. After running the script for about 600 times (as recommended) ,the userA is now able to log in to hostB using public key technology without a password. There still is a problem as the passphrase has to be echoed into the script owing to the fact that SSH does not take input from standard text input. SSH comes with ssh-agent so that the need to retype your password continually is eliminated. ssh-agent is used in combination with ssh-add as follows:

```
[scripts@hostA scripts]$ ssh-agent bash
[scripts@hostA .ssh]$ ssh-add  id_dsa
Identity added: id_dsa (id_dsa)
```

The shell is passed to the ssh-agent, that inherits the keys added with ssh-add. The need to type the passphrase is now reqiured and the default key (id_dsa.pub) can be authenticated.
The way SSH is called is important when using multiple keys in an interactive session with SSH. An example, of which can be a case where three private keys are created: the default key, id_dsa, and two other keys called backup and monitor that are to used for different task. For this the SSH will be called with the -i parameter. This is done to ensure that the new key is being used while logging in to the remote SSH server:

[scripts@hostA]$ssh -i backupkey hostB

For interactive users great time is saved when using ssh-agent. But when implemented in scripts it still requires the passphrase to be typed in at least once when the machine boots.

Public Key:

For the jobs to be totally automated and secured, using public keys is the only alternate. The following techniques can be implemented using public keys:

(1) To secure setup the **from=** directive is used in the authorized_keys file. Syntax:

from="host1,host2" KEY

It allows only users from host1 or host2 to be authenticated against the public key matching KEY. If logins are to be restricted from only hostA and hostB for user scripts, the authorized_keys syntax would be:

from="hostA,hostB" ssh-dss AAAAB..Aqbcw= scripts@hostA

It is still possible to spoof an IP and pretend to be another host. But then there is restriction added by a security layer that increases the effort needed to compromise the host.

(2) In order to secure script setup the **command =""** directive can also be used. Syntax:

```
command ="command",  KEY
```

The SSH runs the ***command*** and then exits limiting the ability to run commands on a remote server. Combination of both hosts can be used in authorized_keys file;

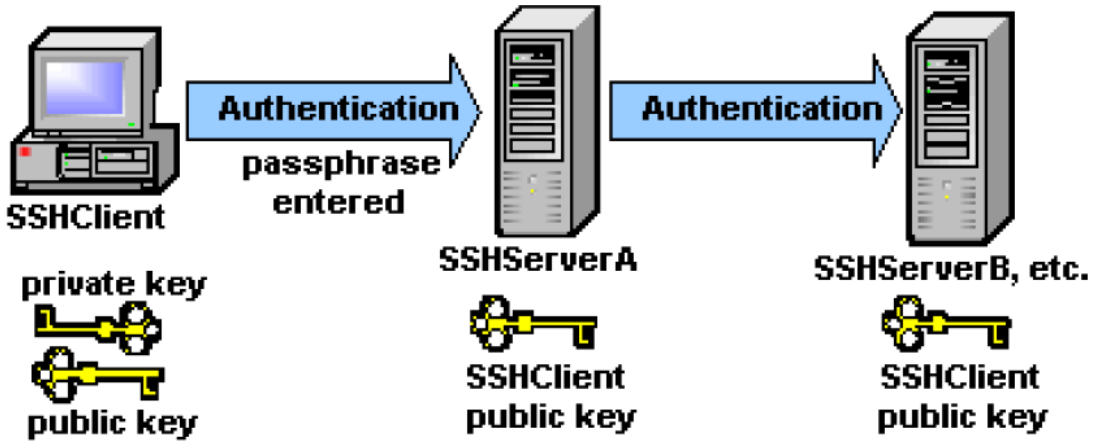
```
from="hostA,hostB",command="/bin/df -k"  ssh-dss AAAAB3N...Aqbcw= userA@hostB
```

The worst that can be done to this scheme is that the list of free disk space can be retrived from the remote host.

(3) The ability to forward traffic is limitted. To do this when this key logs on the ability to forward TCP/IP ports is denied as forwarding ports is a great way to bypass firewalls and is a potential security threat; hence, the ability to run only the commands necessary are allowed for the scripts.
The SSH process is then disabled therefore it is restricted from forwarding any X11 connections upon login. Error is returned if an attempt to do so is made as this is a way used by the intruder to exploit hosts.
The ability to forward ssh-agent and stored keys to another host is denied to the key by the no-agent-forwarding in the authorized_keys reduceing the complexity and taking away another avenue for a potential intruder to trespass.
The no-pty option, restricts allocation of a pseudo-tty when logging in while the non-interactive commands continue to work using the associated key; however, the interactive session can no longer issue commands effectively ensuring that the intruder cannot issue interactive commands to do any damage.
The final authorized_keys has the following syntax:

```
from="hostA,hostB",command="/bin/myscript.sh",no-port-forwarding,no-X11-
forwarding,no-agent-forwarding,no-pty
ssh-dss AAAAB3....o9M9qz4xqGCqGXoJw= scripts@hostA
```

It should be ensured that a layered security approach is implemented to make the servers less vulnerable to attacks and proper options are set for each particular key. To reduce the potential damage done during a successful attack least possible privileges should be set. With these techniques embedded in the system the data and networks stay efficent and secure.
Authentication Agent Forwarding: Authentication agent is not required to run on both servers if one is trying to another using public key authentication. The authentication request is automatically passed on by the SSH from other servers to the agent running on the localhost. By doing this it prevents from passing the secret key to the remote computer. After performing authentication on the localhost it sends the results to the remote computer.



Agent Forwarding passes authentication from the first SSH connection to the next, re-authenticating each time.(1)

ssh –A can be run to setup authentication agent forwarding or add ForwardAgent yes to the config file. Authentication agent forwarding should only be used if the adminstrators of the remote computer can be trusted.(4)

Functions of SSH:

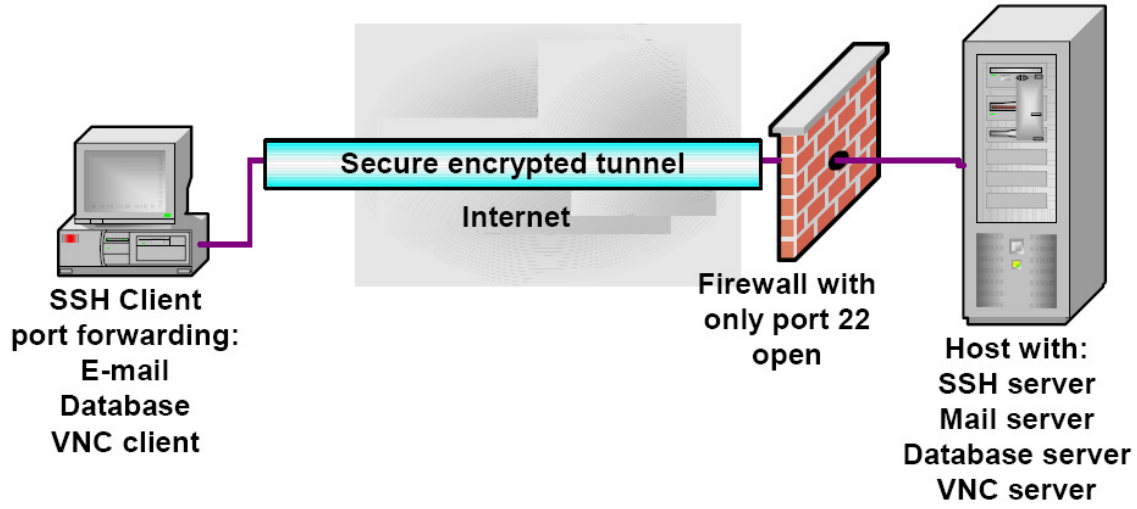
The following are the functions of SSH:

X11 Forwarding:

When running X on the local computer, the remote X applications automatically appears on the local screen as the DISPLAY environment variable on the remote system is set. Not only this the X sessions are also encrypted over the SSH.
X11 forwarding can be turned on with ssh -X host. X11 forwarding should only be used for remote computers where the administrators can be trusted. Otherwise, the system is prone to X11-based attacks.(4)
In order to connect to remote UNIX systems and to configure systems so that it displays graphics from remote systems, Windows users have to install an X server other wise they cannot make use X applications. To improve access to X applications the X server and communications software should be run either by without installing them or with little or no configuration.(5)

Port Forwarding:

Port forwarding is a powerful tool that allows single secure connection to be shared amongst multiple TCP/IP applications such as e-mail, sales and customer contact databases, and in-house applications.
SSH reroutes traffic from the client and sends it across the encrypted tunnel after port forwarding has been set up, it then transmits it to the server. A single multiplexed channel can handle transmissions of multiple applications without opening additional vulnerable ports on a firewall or router.
A secure remote command shell is not sufficient and graphical remote control is necessary for some applications. (1)



SSH client and Host Port Forwarding (1)

Using SSH Port Forwarding to print at Remote Locations

SSH using Port Forwarding allows an arbitrary port of a local system to connect to a port on the remote system or vice versa. To run a browser on a local machine to look at an intranet, hosted on the server at example.com but is not available to the public. ssh can be used to forward the local port 8080 to example.com's port 80 with the command:

```
ssh -L 8080:example.com:80 example.com
```

After authentication you get a shell on example.com. Open up a browser on the localhost, and enter the URL http://localhost:8080/.
The browser sends a request to the local port 8080, which gets passed over to port 80 at example.com. This request is sent and on the same channel receives the answer displayed on the browser. A relay program is now required that runs between the two different lpr systems. For this client/server system can be used.
If the server has a print job waiting, it sends it back to the client, that passes it off to the local lpr.
The localhost lpr system should be up and running, and should send a file to print on the remote system (example.com):

```
lpr mytestfile.txt
```

At this point the status of the print job with `lpq` checked along with the shell where rlclient is running.(6)

Compression and Ciphers:

Compression can be turned on if `Compression yes` is put in your config file or with with `ssh -C`. It is a great idea when forwarding X sessions on a dial-up or slow network. Gzip compression can be applied on any connection by the SSH. The default compression level is equivalent to approximately 4× compression for text..
Changing the encryption cipher can also speed up the process. The default cipher used by older systems is triple DES (3DES), which as compared to Blowfish and AES is slow. But then Blowfish is used by new versions of OpenSSH. To change the cipher to blowfish type ssh -c blowfish.
To apply Cipher changes to the config file use Cipher blowfish if connecting with SSH1 or Ciphers blowfish-cbc,aes128-cbc,3des-cbc,cast128-cbc,arcfour,aes192-cbc,aes256-cbc if connecting to SSH2. (4)

Encryption:

Protocols such as POP3, IMAP, SMTP and NNTP are used by applications to send to send data and passwords but both are sent as clear text. These connections can be transparently encrypted by SSH. An e-mail client that connects using the POP3 port (110) on example.mail.com cannot SSH it directly to example.mail.com but with a shell login present at shell.mail.com it can instruct SSH to encrypt traffic from say port 9110 on the local computer and send it to port 110 on example.mail.com using the SSH server at shell.mail.net:

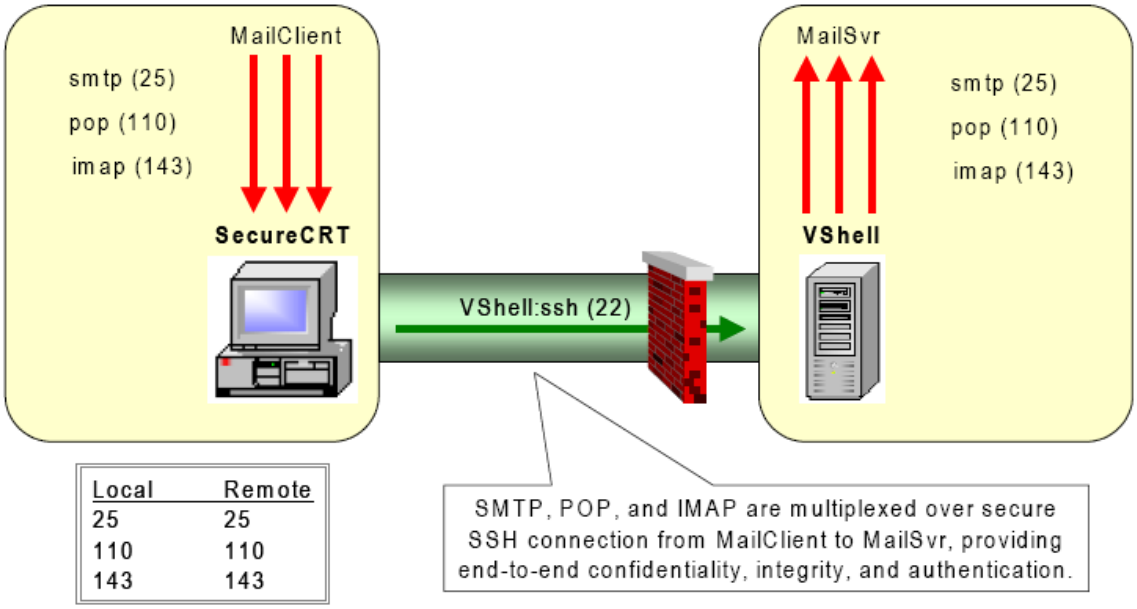
```
ssh -L 9110:example.mail.com:110 shell.mail.com
```

It then sends local port 9110 to example.mail.com port 110, using the SSH connection to shell.mail.com.
When the e-mail program connects to port 9110 on the localhost the data is encrypted, transmitted to shell.mail.com using the SSH port. It is then decrypted and passed to example.mail.com over port 110. All this time the POP3 dæmon on mail.example.net has an illusion that it is accepting traffic from shell.mail.com. (4)

Tunnelling with SSH: (7)

SSH Tunneling will described in the context of local port forwarding tunnels by a SSH client rather than a server. By forwarding individual TCP ports application streams are tunneled over the SSH.
The SSH client (SecureCRT) listens to the TCP port on the local host when a local port is forwarded. This allows the TCP connection to open for the remote host where the actual server application is running by the SSH server (VShell).
Staying with the convention localhost refers the application client's host, if the local host is not specified it defaults to SecureCRT host while remotehost will be refers the application server's host that defaults to VShell host.
The port that the application client sends to and SecureCRT listens to is referred by the localhost that can have an unused port known as the localport while the port that VShell sends data to and the application server listens to is referred by the remoteport which is a IANAassigned listening port for the tunneled application.

The client application has to be reconfigured to connect to localhost: localport in order to use port forwarding.
The SecureCRT intercepts and encrypts packets sent by the client to the localhost: localport. These packets are then tunneled through the SSH connection to the VShell or another SSH server. These packets are then decrypted to be relayed as clear text to the remotehost: remoteport via the TCP connection.



SSH Tunneling (7)

The traffic between VShell and the remote host is usually not cryptographically protected as the VShell is present behind a firewall inside the network boundary. The application protocols are restricted and only the SSH is permitted by the firewall's configuration.
The firewall is configured in away that it protects the clear text traffic in the servers of a trusted LAN. The VShell can run on the same machine if the LAN is not trusted or Intranet servers are at a premium.

If the SecureCRT is running on the same PC as the client application then it is appropriate to use local port-forwarding initiating outbound TCP connections to the server application.
An advantage of using SSH is also that it forwards individual TCP connections but not the port ranges which weaker protocols like FTP do.

Disscusions:

SSH has a protocol layered on it known as the SFTP or the Secure File Transfer. This has advantages over the old FTP as it adopts the encryption technique which secures both the Username/password and the text by making it look like garbage. As it uses the same port as the SSH it eliminates the need for another port on the firewall or the router. Previously Network Address Translation was an issued faced by the professional using FTP, with SFTP this is also resolved. (8)

With all these features of security by encryption there is one flaw in the SSH protocol that was reported in Dai (2002) (9). The point that was raised was that if such a transmission is intercepted the current message or packet of data will be encrypted and will not be understandable to the attacker but if the attacker gets control of the entire first block of the input into SSH-IPC's underlying scheme, that will enable him to know the initialization vector or the IV of the next message before it is encrypted. As a block cipher is deterministic the attacker can generate information about the previouly encrypted message.
To over come this attack what can be done is (9) that the underlying encryption scheme is replaced with randomized CBC mode in which for every new message a new random IV is chosen that is sent with the cipher text.
There are other techniques that can be used instead of the SSH if the above seems to be a problem depending on the situation of the network use. A paradigm known as the Encrypt-then-Mac can be used. (9). In this scehme the message is first encrypted with an underlying encryption scheme and then with an underlying message authentication scheme the cipher text is MACed.
Another technique that can be used is the Dedicated Authenticated Encryption Scheme. This is a symmetric key based technique that carries out the authenticated encryption functions. Such a scheme is considered to be more potentially efficent as compared to the schemes based on a black-box composition of off-the-shelf encryption and MAC components.

Conclusion:

In this paper an attempt was made to bring the reasons why SSH has gained popularity and why the popularity is just. There is work done on other techniques to replace SSH but for now if SSH is kept for transmissions it will be not a deal. The attack on a SSH transmission is not all that easy as mentioned in the 'Discussion' earlier. To plant an attack, it is necessary to control the next message that a user encrypts after seeing the cipher text collision. It will not be practical to attack if a lot of data is quickly being encrypted by the session while transferring. As a result the attacker will not have the time to detect collision. (9)
Keeping all of these above aspects forward on the table I believe SSH deserves the popularity it has If major Software Giants really open there minds to what SSH provides they should provide full support to present a platform that not only is fully compatible with the protocol but a forum should be established to enhance the features and compatibility of SSH. All this will require is the thought to have a secure environment rather than who leads in profiting from it.

References:

1. An Overview of the Secure Shell (SSH), White Paper presented by Vandyke Software Inc.Copyright2005, Source: http://www.vandyke.com/solutions/whitepapers.html
2. Paranoid Penguin: The 101 Uses of OpenSSH: Part 1, by Mick Bauer. Year of Publication: January 2001, Linux Journal, Publisher: Specialized Systems Consultants. Source ACM Digital Library.
3. Paranoid Penguin: Managing SSH for scripts and cron jobs by John Ouellette. Year of Publication: September 2005, Linux Journal, Volume 2005, Issue 137. Publisher: Specialized Systems Consultants. Source: ACM Digital Library.
4. Eleven SSH Tricks by Daniel R. Allen. Year of Publication: August 2003. Linux Journal, Volume 2003, Issue 112. Publisher: Specialized Systems Consultants. Source: ACM Digital Library.
5. Easy access to remote graphical UNIX applications for windows users by Richard R. Repasky, Indiana University, U.S.A., Publisher: ACM Press New York. 2004. Source: ACM Digital Library.
6. Using SSH Port Forwarding to Print at Remote Locations by Rory Krause. February 2002, Linux Journal, Volume 2002, Issue 94. Publisher: Specialized Systems Consultants. Source: ACM Digital Library.
7. Tunnelling with Secure Shell (SSH), White Paper by Vandyke Software Inc, Copyright 2003. Source: http://www.vandyke.com/solutions/whitepapers.html

8. Secure File Transfer: White Paper by Vandyke Software Inc, Copyright 2003. Source: http://www.vandyke.com/solutions/whitepapers.html
9. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm: by Mihir Bellare, Tadayoshi Kohno of University of California, San Diego, La Jolla, CA and Chanathip Namprempre of Thammasat University, Patumtani, Thailand. **ACM** Transactions on Information and System Security (TISSEC) archive
Volume 7, Issue 2 (May 2004). Publisher ACM Press, New York. Source: ACM Digital Library.

Bibliography:

1. An Overview of the Secure Shell (SSH), White Paper presented by Vandyke Software Inc.Copyright2005, Source: http://www.vandyke.com/solutions/whitepapers.html
2. Paranoid Penguin: The 101 Uses of OpenSSH: Part I of II, by Mick Bauer. Year of Publication: January 2001, Linux Journal, Publisher: Specialized Systems Consultants. Source ACM Digital Library.
3. Paranoid Penguin: The 101 Uses of OpenSSH: Part II of II, by Mick Bauer. Year of Publication: February 2001, Linux Journal, Publisher: Specialized Systems Consultants. Source ACM Digital Library.
4. Paranoid Penguin: Managing SSH for scripts and cron jobs by John Ouellette. Year of Publication: September 2005, Linux Journal, Volume 2005, Issue 137. Publisher: Specialized Systems Consultants Inc. Source: ACM Digital Library.
5. Eleven SSH Tricks by Daniel R. Allen. Year of Publication: August 2003. Linux Journal, Volume 2003, Issue 112. Publisher: Specialized Systems Consultants. Source: ACM Digital Library.
6. Easy access to remote graphical UNIX applications for windows users by Richard R. Repasky, Indiana University, U.S.A., Publisher: ACM Press New York. 2004. Source: ACM Digital Library.
7. Using SSH Port Forwarding to Print at Remote Locations by Rory Krause. February 2002, Linux Journal, Volume 2002, Issue 94. Publisher: Specialized Systems Consultants. Source: ACM Digital Library.
8. Tunnelling with Secure Shell (SSH), White Paper by Vandyke Software Inc, Copyright 2003. Source: http://www.vandyke.com/solutions/whitepapers.html

9. Secure File Transfer: White Paper by Vandyke Software Inc, Copyright 2003. Source: http://www.vandyke.com/solutions/whitepapers.html
10. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm: by Mihir Bellare, Tadayoshi Kohno of University of California, San Diego, La Jolla, CA and Chanathip Namprempre of Thammasat University, Patumtani, Thailand. ACM Transactions on Information and System Security (TISSEC) archive
Volume 7, Issue 2 (May 2004). Publisher ACM Press, New York. Source: ACM Digital Library.
11. Take Command; ssh: Secure Shell, by Alessandro Rubini, November 1997, Linux Journal Published by Specialized Systems Consultants Inc, Source ACM Digital Library.
12. DMSEC session: HMM Profiles for Network Traffic Classification (Extended Abstract) by Charles Wright, Fabian Monrose, Gerald M. Masson of Johns Hopkins University, Information Security Institute. October 2004, Published by ACM Press. Source ACM Digital Library.
13. ACAS: Automated Construction of Application Signatures, by Patrick Haffner, Subhanbrata Sen, Oliver Spatscheck, Dongemi Wang of AT&T Labs-Research, Florham Park, New Jersey. Published August 2005. Source: ACM Digital Library.