Security with SSH

pre SANOG VI Workshop

Hervey Allen

Topics

- Where to get SSH (Secure SHell)
- How to enable and configure SSH
- Where to get SSH clients for Windows
- Authentication of the server to the client (host keys)
- Issues to do with changing of the host key
- Password authentication of the client to the server
- Cryptographic authentication of the client to the server (rsa/dsa keys)
- hostkey exchange, scp, and sftp labs

Cryptographic Methods and Apps

Previously we had mentioned the following practical applications apply to the following methods:

• At the link layer	PPP encryption
• At the network layer	IPSEC
• At the transport layer	TLS (SSL)
• At the application layer	SSH, PGP/GPG

SSH Application Layer Security

In this section we will go over SSH at the application layer to do both authentication and data encryption.

We are going to largely ignore SSH Version 1 issues with RSA 1 Keys as RSA1 and SSH version 1 and 1.5 are no longer secure.

Main Security Concerns

SSH applies directly to dealing with these two areas of security:

- Confidentiality
  - Keeping our data safe from prying eyes
- Authentication and Authorization
  - Is this person who they claim to be?

Where to Get SSH

First see if SSH is installed on your system and what version. Easiest way is:

```
ssh -V
```

If you want or need an updated version of OpenSSH (current version is 3.8.x) you can go to the following places:

/usr/ports/security/openssh-portable/
http://www.openssh.org/
http://www.ssh.com/

We recommend using OpenSSH for FreeBSD. Default version installed in FreeBSD 5.3 is OpenSSH Portable version 3.8.1p1

Enable and Configure OpenSSH

On our machines this is already done, but if you did somehing like:

```
/usr/ports/security/openssh-portable/make install
```

- You should make sure that /etc/rc.conf is set:

```
sshd_enable="YES"
```

- Take a look at /etc/ssh/ssh_config and /etc/sshd_config. In sshd_config you might be interested in:

```
PermitRootLogin yes/no
```

and in /etc/ssh/ssh_config (this could cause problems):

```
Protocol 1,2
```

There are *many* options in ssh_config and sshd_config. You should read through these files to verify they meet your expecations.

Where to Get SSH Clients for Windows

There are several free, shareware, and commercial ssh clients for Windows:

See http://www.openssh.org/windows.html for a list.

A few that support protocol version 2 include:

- Putty: http://www.chiark.greenend.org.uk/~sgtatham/putty/
- OpenSSH for Windows (using Cygwin):

  http://www.networksimplicity.com/openssh/
- Secure Shell from ssh.com (free for personal use):

  http://www.ssh.com/products/ssh/download.cfm

And F-Secure at http://www.f-secure.com/products/ssh/ is a nice product if you are willing to pay.

Some Useful SSH References

- If you want a great SSH RSA/DSA key overview Daniel Robbins CEO of gentoo.org has written a 3-part series hosted on the IBM Developer Works pages.

- **The three papers and URL's are:**

OpenSSH Key Management, Part 1
http://www-106.ibm.com/developerworks/library/l-keyc.html
OpenSSH Key Management, Part 2
http://www-106.ibm.com/developerworks/library/l-keyc2/
OpenSSH Key Management, Part 3
http://www-106.ibm.com/developerworks/library/l-keyc3/

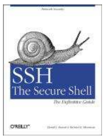
More SSH References

For a comparison of SSH Version 1 and 2 see:

http://www.snailbook.com/faq/ssh-1-vs-2.auto.html

An excellent book on SSH is:

SSH, The Secure Shell
The Definitive Guide
By Daniel J. Barrett &
Richard Silverman
January 2001
ISBN: 0-596-00011-1



SSH Connection Methods

Several things can happen when using SSH to connect from your machine (client) to another machine (server):

- Server's public host key is passed back to the client and verified against known_hosts

- Password prompt is used if public key is accepted, or already on client, or

- RSA/DSA key exchange takes place and you must enter in your private key passphrase to authenticate (assuming you have one).

SSH Quick Tips

You have a choice of authentication keys - RSA is the default (dsa is fine as well).

The files you care about are:

/etc/ssh/ssh_config
/etc/ssh/sshd_config
~/.ssh/id_dsa and id_dsa.pub
~/.ssh/id_rsa and id_rsa.pub
~/.ssh/known_hosts
~/.ssh/authorized_keys
And, note the rsa/dsa host-wide key files in /etc/ssh

Be *sure* that you do “man ssh” and “man sshd” and read the entire descriptions for both the ssh client and ssh server (sshd).

SSH Authentication

Private key can be protected by a passphrase
- So you have to give it each time you log in
- Or use "ssh-agent" which holds a copy of your passphrase in RAM

No need to change passwords across dozens of machines

Disable passwords entirely!
- /etc/ssh/ssh_config

```
#   PasswordAuthentication yes
```

Annoyingly, for historical reasons there are *three* different types of SSH keys
- SSH1 RSA, SSH2 DSA, SSH2 RSA

Man in the Middle Attacks

The first time you connect to a remote host, remember its public key
- Stored in ~/.ssh/known_hosts

The next time you connect, if the remote key is different, then maybe an attacker is intercepting the connection!

Or maybe the remote host has just got a new key, e.g. after a reinstall. But it's up to you to resolve the problem

You will be warned if the key changes.

Exchanging Host Keys

First time connecting with ssh:

```
ssh username@pc1.presanog.org.bt
The authenticity of host 'pc1.presanog.orb.bt (202.144.139.1)' can't be
established.
DSA key fingerprint is 91:ba:bf:e4:36:cd:e3:9e:8e:92:26:e4:57:c4:cb:da.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'pc1.presanog.orb.bt, 202.144.139.1' (DSA) to
the list of known hosts.
username@pc1.prsanog.org.bt's password:
```

At this point the client has in the file ~/.ssh/known_hosts the contents of pc1.presanog.org.bt's /etc/ssh/ssh_host_dsa_key.pub.

Next connection:

```
[hallen@hallen-lt .ssh]$ ssh usrname@pc1.presanog.org.bt
t1@pc1.presanog.org.bt's password:
```

Now trusted - Not necessarily a good thing...

Exchanging Host Keys Cont.

Command	**Key Type Generated**	**Public File**
ssh-keygen -t rsa	**RSA (SSH protocol 2)**	**id_rsa.pub**
ssh-keygen -t dsa	**DSA (SSH protocol 2)**	**id_dsa.pub**

- Default key size is 1024 bits
- Public files are text
- Private files are encrypted if you use a passphrase (still text)

Corresponding file on the host for host key exchange is “known_hosts”.

Exchanging Host Keys Cont.

How does SSH decide what files to compare?

Look in /etc/ssh/sshd_config. For OpenSSH version 3 the server defaults to protocol 2 .

By default OpenSSH version 2 client connects in this order:
- RSA version 2 key
- DSA version 2 key
- Password based authentication (even if RSA version 1 key is present)

Pay attention to the “HostKeyAlgorithms” setting in /etc/ssh/ssh_config to help determine this order - or use ssh command line switches to override these settings.

SSH - “Magic Phrase”

Basic concept to understand how an SSH connection is made using RSA/DSA key combination:
- Client X contacts server Y via port 22.
- Y generates a random number and encrypts this using X's public key. X's public key must reside on Y. You can use scp to copy this over.
- Encrypted random number is sent back to X.
- X decrypts the random number using it's private key and sends it back to Y.
- *If the decrypted number matches the original encrypted number, then a connection is made.*
- The originally encrypted random number sent from Y to X is the “Magic Phrase”

We'll try drawing this as well...

SSH - Lab

We will now practice the following concepts:

- The use of known_hosts files
- SSH connection with password authentication
- RSA version 2 protocol key generation
- Public key copying
- Connecting with private key passphrase using key-based authentication
- Using scp with RSA key authentication
- Some ssh "hacks" without passwords.

*Technically you are still challenged (even if that is a bad pun in English).

SSH - Lab Cont.

The use of known_hosts files

Connect to the machine next to your machine using ssh:

```
     ssh root@pcN.presanog.org.bt
```

If this is your first connection to this machine you should see (example uses host1 connecting to host2):

```
pc1# ssh root@pc2.presanog.orb.bt
The authenticity of host 'pc2.presanog.orb.bt (202.144.139.2)' can't be
established.
RSA1 key fingerprint is 60:f7:04:8b:f7:61:c4:41:6e:9a:6f:53:7d:95:cb:29.
Are you sure you want to continue connecting (yes/no)?
```

Go ahead and answer "yes" here, but we'll discuss the implications of this in class. Are there ways around this? Could this be a "man in the middle" attack? What file is created or updated? Why?

SSH - Lab Cont.

ssh connection with password authentication

At the prompt below when you answered yes, you were asked to enter in the root password for pc2.presanog.org.bt:

```
host1# ssh root@pc2.presanog.orb.bt
The authenticity of host 'pc2.presanog.orb.bt (202.144.139.2)' can't be
established.
RSA2 key fingerprint is 60:f7:04:8b:f7:61:c4:41:6e:9a:6f:53:7d:95:cb:29.
Are you sure you want to continue connecting (yes/no)? yes
```

And, this is what you should have seen:

```
Warning: Permanently added 'pc2.presanog.org.bt' (RSA2) to the list of known
hosts.                  [/etc/ssh/ssh_host_key.pub]
root@pc2.presanog.org.bt's password:
```

Now you are "securely" connected as root to pc2.presanog.org.bt - We will discuss what happened during this connection.

SSH - Lab Cont.

rsa1/rsa2/dsa Key Generation

We will now generate a single RSA SSH protocol 2 key of 2048 bits. To do this, issue the following command. If you are logged in on the other machine, logout first!

Before continuing: you may need to edit /etc/ssh/ssh_config and make sure that the "Protocol" option is set either to "Protocol 2,1" or "Protocol 2"

```
     ssh-keygen -t rsa -b 2048
```

You will be prompted for a file location for the key as well as for a passphrase to encrypt the key file. Be sure to enter a passphrase. Private key files without passphrases are a security hole, or maybe not... We'll discuss this as we complete this excercise. You can use a passphrase other than "san0g2k5" if you wish.

SSH - Lab Cont.

RSA 2 Key Generation

Here is the output from the command "ssh-keygen -t rsa -b 2048":

```
pc1# ssh-keygen -t rsa -b 2048
Generating public/private rsa key pair.
Enter file in which to save the key
(/root/.ssh/id_rsa): [enter]
Enter passphrase (empty for no passphrase): [pw]
Enter same passphrase again: [pw]
Your identification has been saved in /
root/.ssh/id_rsa.
Your public key has been saved in /
root/.ssh/id_rsa.pub.
The key fingerprint is:
0f:f5:b3:bc:f7:5b:c8:ce:79:d0:b1:ab:2c:67:21:62
root@pc1.presanog.org.bt
pc1#
```

SSH - Lab Cont.

Public Key Copying

Now that you have a public and private RSA(2) set of keys you can take advantage of them. We will copy the public key to the same host you connected to previously, save this to the files *known_hosts*, and then reconnect to the host and see the difference:

First you must copy the public key files to the host you used previously (pcn.presanog.org.bt):

```
cd ~/.ssh
scp id_rsa.pub root@pcn.presanog.org.bt:/tmp/.
```

You will be prompted for the password for the host *and* username you are connecting to. We continue with our example using pc1 connecting to pc2 as root.

SSH - Lab Cont.

Public Key Copying

The output from the command on the previous page looks like:

```
pc1# scp *.pub root@pc2.presanog.org.bt:/tmp/.
root@pc2.presanog.org.bt's password:
id_rsa.pub           100% |******************************|   408       00:00

pc1#
```

You now have the public key file sitting on the host that will need them to use RSA/DSA public/private key authentication with you. You next step is to place these keys in the appropriate files.

You need the RSA keys in *~/.ssh/authorized_keys*

You can try to figure this out, or go to the next slide for steps to do this:

SSH - Lab Cont.

Public Key Copying

To copy the public keys to the correct places do the following:

```
ssh root@pcn.presanog.org.bt
cat /tmp/id_rsa.pub >> ~/.ssh/authorized_keys
rm /tmp/id_rsa.pub
exit
```

If you are unsure of what these commands do they will they are explained in class. In addition, you can do this many different ways, and you could issue the commands differently as well. If you understand what these commands do and have a preferred method, then feel free to use it.

Go to the next slide to connect with your public/private keys!

SSH - Lab Cont.

Public/Private Key Connection

To connect using your RSA protocol 2 key simply type:

```
ssh root@pcn.presanog.org.bt
```

And, here is the output you should see (pc1 to pc2 example):

```
host1# ssh root@pc2.presanog.org.bt
Enter passphrase for RSA key 'root@pc1.presanog.org.bt':
```

This is actually pretty neat! You *did not* enter in the root password for the root account on pcn.presanog.org.bt, but rather you used the passphrase that you chose for your private RSA protocol 2 key when you issued the command "ssh-keygen -t rsa -b 2048" - This was used to decode the encoded random number exchanged between the hosts (remember "Magic Phrase?").

Why was the RSA protocol 2 key used? We'll discuss this in class.

SSH - Lab Cont.

SCP Public/Private Key Connection

First disconnect from the ssh session you previously made:

```
exit
```

Now, try copying a file from your machine to the other machine (pick a small file) using SCP (SeCure coPy):

```
scp filename root@pcn.presanog.org.bt:/tmp/.
```

What did you notice? You should have noticed that you no longer get a password challenge to this account on this node, but rather you need to provide your RSA protocol 2 private key passphrase.

This is expected. SCP and SSH are from the same package - OpenSSH and both use RSA and DSA keys in the same way.

SSH - Lab Cont.

Another SSH tool - SFTP

In addition to scp, ssh has a secure ftp tool called sftp. Give it a try:

Let's use sftp to get your neighbor's /etc/motd file and place it in your /tmp directory.

```
sftp root@pcN.presanog.org.bt
```

Once you are connected:

```
sftp> lcd /tmp       [change local directory to /tmp]
sftp> cd /etc        [change remote directory to /etc]
sftp> get motd       [download /etc/motd to /tmp/motd]
sftp> ?              [view summary help]
sftp> bye            [terminate connection]
ls /tmp/motd         [prove you got the file]
```

SSH - Lab Cont.

Now let's use the power of scp

Multiple file and directory copy:

Let's copy all the files and directories in /usr/ports/palm from your machine to your neighbor's machine using one command:

```
scp -r /usr/ports/palm/* root@pcN.presanog.org.bt/tmp/.
```

- "-r" for recursively copy
- "/tmp/." to place files in your neighbor's /tmp directory.

SSH - Lab Cont.

Now let's use the power of scp some more!

Copy a file from one remote machine to another.

Let's move /etc/fstab on your left neighbor's machine to /tmp/fstab.copy on your right neighbor's machine using a single command.

```
scp  root@pcLEFT.presanog.org.bt:/etc/fstab \
    root@pcRIGHT.presanog.org.bt/tmp/fstab.copy
```

- “\” for newline, not part of the command.
- If root password is the same on both you only enter it once.
- Did you notice we renamed the file as well?

SSH Conclusion

SSH, SCP, and SFTP are great tools for connecting between machines and copying data while helping to maintain a secure environment.

If you can, we recommend you remove telnet and FTP from your system. Or, at most, only allow anonymous FTP access.

You can use SSH to tunnel ports securely that would otherwise pass your information (username, password, and session data) in the clear.

Remember - Use the references for more detailed information. This includes “*man ssh*” and “*man sshd*” for much more information.

This slide left intentionally blank.

Additional Topics

We have made available additional or advanced topics for those who are interested:

- **No challenge connection**
- **ssh-agent and ssh-add**
- **ssh tunnels**
  - **Direct and indirect port forwarding**
- **Private key without passphrase**
  - **Remote command execution**

SSH - Lab Cont.

Example of a No Challenge Connection

We will now use ssh-agent and ssh-add to setup an environment on your machine where you can connect to your other machine, as root, without having to enter a password or passphrase at the time of the connection.

You will, however, have to enter your RSA protocol 2 private key passphrase once during this session. We'll discuss ***ssh-add*** and ***ssh-agent*** in class, but read “***man ssh-agent***” and “***man ssh-add***” for more details:

On the next slide you will setup your bash shell environment to contain your RSA protocol version 2 private key passphrase. This will allow you to connect, logout, reconnect, exit, connect again, and so on to root at the host you have chosen issuing your private key passphrase only *once:*

SSH - Lab Cont.

Example of a No Challenge Connection

Follow these steps to setup a “no challenge” connection:

```
        ssh-agent /bin/bash
        ssh-add
        ssh root@pcn.presanog.org.bt
```

What happened? You should have been prompted for your RSA version 2 protocol private key passphrase (remember, that's what is in ~/.ssh/id_rsa) when you typed ***ssh-add***. Then, when you connected you did not need a passphrase. (If you have an RSA 1 key, you will be prompted for the passphrase for ~/.ssh/identity).

Now for the fun part. Logout, and log back in to the same session:

```
        logout
        ssh root@pcn.presanog.org.bt
```

Now what happened?

SSH - Lab Cont.

No Challenge Connection Notes

- ssh-add and ssh-agent have some slightly different behavior than just using ssh.
- If you don't specify a passphrase for your private key files when you create them, then you can truly connect with no password challend of any type - This is dangerous!
- Note that ssh-add defaults to ~/.ssh/id_rsa first then id_dsa.

SSH - Lab Cont.

Additional Notes

- You can use ssh-agent to "wrap" other programs that may need to use RSA/DSA authentication, but that cannot deal with multiple passphrase (or password) requests.
- These lab slides contain a complete session with notes of using ssh-agent and ssh-add.

SSH - Lab Cont.

ssh-agent/ssh-add session*

```
host5# where bash                         [Find where bash resides]
/usr/local/bin/bash
host5# ssh-agent /usr/local/bin/bash      [Wrap bash in ssh-agent]
bash-2.05a# ssh-add                       [Add rsa1 private key by default]
Need passphrase for /root/.ssh/identity
Enter passphrase for root@pc5.presanog.org.bt:
Identity added: /root/.ssh/identity (root@pc5.presanog.org.bt)
bash-2.05a# ssh-add ~/.ssh/id_rsa         [Add rsa v2 private key explicitly]
Need passphrase for /root/.ssh/id_rsa
Enter passphrase for /root/.ssh/id_rsa:
Identity added: /root/.ssh/id_rsa (/root/.ssh/id_rsa)
bash-2.05a# ssh root@pc6.presanog.org.bt  [Login with no password challenge]
Last login: Tue May  7 02:47:24 2002 from pc5.presanog.org.bt
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
        The Regents of the University of California.  All rights reserve
FreeBSD 4.5-RELEASE (GENERIC) #0: Mon Jan 28 14:31:56 GMT 2002

-----------------------------------------------------------------------
pre SANOG VI Workshop - Thimphu, Bhutan
-----------------------------------------------------------------------
You have mail.
host6#
```

*Still relevant, but example is from May 2002 using SSH Version 3.1.

SSH - Lab Cont.

ssh-agent/ssh-add session

```
host6# exit                                          [Exit the shell session]
logout
Connection to pc6.presanog.org.bt.
bash-2.05a# ssh root@pc6.presanog.org.bt             [Log back in - No password!]
Last login: Tue May  7 03:00:53 2002 from pc5.presanog.org.bt
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
        The Regents of the University of California.  All rights reserve
FreeBSD 4.5-RELEASE (GENERIC) #0: Mon Jan 28 14:31:56 GMT 2002

-----------------------------------------------------------------------
pre SANOG VI Workshop - Thimphu, Bhutan
-----------------------------------------------------------------------
You have mail.
host6# exit                                          [Exit the session again]
logout
Connection to pc6.presanog.org.bt closed.
bash-2.05a#
bash-2.05a# ssh-add -l                               [Show rsa/dsa key fingerprints]
2048 7d:68:9b:55:0f:ba:6c:75:23:ab:36:fb:4c:a3:66:ea /root/.ssh/id_rsa (RSA)
bash-2.05a#
```

SSH - Lab Cont.

ssh-agent/ssh-add session end

```
bash-2.05a# ssh-add -d ~/.ssh/id_rsa                 [Remove a private key]
Identity removed: /root/.ssh/id_dsa (/root/.ssh/id_dsa.pub)
bash-2.05a# ssh-add -l                               [List remaining keys]
bash-2.05a#
bash-2.05a# exit
exit                                                 [Exit ssh-agent bash shell]
host5#
```

Don't forget to read up on this with "*man ssh-agent*," and "*man ssh-add*" for many more options and details about how to use these programs.

Tunneling with SSH

The Topic You've Been Waiting For...

- You can use SSH to tunnel insecure services in a secure manner.
- SSH tunneling services includes authentication between known_hosts, password challenge, and public/private key exchanges.
- You can even indirectly tunnel via an intermediary machine.

Tunneling with SSH Cont.

The basic concept looks like this:

- Connect from one machine to another as `username`.
- Use ssh options to specify the port number on the remote machine that you wish to forward to the port on your local machine.
- Your ssh connection will *"tunnel"* data securely across ssh from the remote machine to your local machine.
- There are several options to be aware of.

Tunneling with SSH Cont.

Tunneling by Example

Here is a sample tunnel command using SSH under FreeBSD:

```
ssh -C -f username@host.domain -L 1100:localhost:110 sleep 10000
```

What is happening here?

- The '-C' option specifies compress the data. Good if it works.

- '-f' means ssh goes to the background just before executing the specified command listed (in this case, "sleep 10000").

- '-L' forwards the port on the left, or client (1100) to the one on the right (110) or remote side.

Tunneling with SSH Cont.

Tunneling by Example Cont.

So, what does this command do?

```
ssh -C -f username@host.domain -L 1100:localhost:110 sleep 10000
```

- This "tunnels" your POP email from port 110 on the remote side through port 1100 on your local side.
- The process backgrounds for 10000 seconds (detaches and runs).
- This is done under the authority between yourself (client) and user@host.domain.

Diagram* of Tunneling *both* smtp and POP Services

```
                                                           Host.domain:110
localhost:1100  o-<+----------+ ~   ~   +----------+>--<<--o-------------+
                   |SSH Client|~ ~ ~ ~ ~|SSH Server|       | mail server |
localhost:2500  o->+----------+   ~   ~ +----------+>-->>--o-------------+
                                                           host.domain:25
```

*Thanks to http://www.ccs.neu.edu/groups/systems/howto/howto-sshtunnel.html

Tunneling with SSH Cont.

Tunneling by Example Cont.

Why use something like ports "1100" and "2500"?

- Ports up to 1024 can only be reset by the root user.
- If you are root you can forward 110 to 110, 25 to 25, and so on.
- Other popular tunneling tricks include tunnels for XWindows, IMAP, etc.
- On the client side you must set programs to use "localhost" - For example, for POP and smtp, your mail client must use "localhost" instead of host.domain (i.e. no more "mail.host.com").
- If you are not root, and your ports are changed, then your mail client must be able to set the smtp and POP ports as well.
- **We'll show this using Thunderbird under Linux right now...**

Tunneling with SSH Cont.

One More Tunneling Example

You can use SSH to do "Indirect Port Forwarding"

- What to do if your organization's email sits behind a firewall?
- Connect via an intermediary box (gateway).
  Here's a real world example:

```
Ssh -C -f hallen@gateway.turbolinux.com -L
2500:mail.us.tlan:25 -L 1100:mail.us.tlan:110 /bin/sleep
10000
localhost:1100  o-<+----------+ ~   ~   +----------+>--<<--o-------------+..
                   |SSH Client|~ ~ ~ ~ ~|SSH Server|       |   gateway   |..
localhost:2500  o->+----------+   ~   ~ +----------+>-->>--o-------------+..

                             host.domain:110
...>--<<--+----------+>--<<--o-------------+
          |SSH Server|       | mail.us.tlan|
...>-->>--+----------+>-->>--o-------------+
                              host.domain:25
```

Tunneling with SSH Conclusion

- Tunneling lets you securely access basic services such as POP and IMAP.
- You can securely tunnel ports using SSH.
- You can use /etc/services to verify you are not using a port that is already defined.
- Only root can redfine ports below 1024.
- You can tunnel ports directly between two machines, and indirectly with a machine in the middle.

Some ssh hacks

Ok, let's break the rules. Imagine if you *did not* generate a passphrase for your private ssh keys?...

These ideas are courtesy of *Linux Server Hacks* by Rob Flickenger and O'Reilly books.

OK, so you don't generate a password when generating your passphrase (hit enter twice). What are the security implications of this?

Bottom line: *Keep* your private key safe! :-)

If you were to lose your private key you would need to remove all your public keys from all servers and accounts where they reside!

Some ssh hacks cont.

If you have a private key without a passphrase then you can take advantage of this by writing a short script named “ssh-to” that looks like this:

```
#!/bin/sh
ssh `basename $0` $*
```

Place this in /home/userid/bin and try something like:

```
ssh-to pcN.presanog.org.bt
ssh-to pcN.presanog.org.bt uptime
```

Some ssh hacks cont.

Note that ssh passes your username to the server if you don't specify one. So, what happened? You connected with no challenge at all.

Even better, you can run remote commands on the server (remember the “$*” in our script?).

Now, to *really* speed things up do:

```
cd bin
ln -s ssh-to host1
ln -s ssh-to host2
ln -s ssh-to host3
```

Some ssh hacks cont.

Now you just have to type something like:

```
host3 uptime
```

Assuming you have your public_keys on this machine.

As a system administrator this clearly serves you well if you do this as root and you have root access on multiple boxes.

You'll need to make sure that “`PermitRootLogin yes`” is set in /etc/ssh/sshd_config on each machine you wish to connect with.

Now some ssh-agent hacks

Using ssh and ssh-agent it's possible to connect to host1 as user1 and from host1 connect to host2 as user1 again *without* needing a password!

ssh-agent will check with ssh-agent in your original shell if the “**`ForwardAgent yes`**” flag has been set in either ~/.ssh/config (if it exists), or in /etc/ssh/ssh_config. This must be set on all machines where you wish to do this.

ssh-agent hacks cont.

To continue first do:

```
eval `ssh-agent`
```

Then:

```
ssh-add
```

If you have passwords, then you'll have to enter them for each ssh private key you generated (if they are different). Otherwise your private keys will automatically be loaded in to memory.

ssh-agent hacks cont.

If no passwords were set on any of your private keys, your public keys are on host2, host3, and host4, and you have root access on each machine, then you can do this:

```
root@host1:$ ssh host2
root@host2:$ ssh host3
root@host3:$ ssh host4
```

And you will not be asked for a password at any time. Very cool, and seems kind of scary...

One final ssh-agent hack

What happens in a GUI environment with ssh-agent and terminals? Each time you open a terminal you must run "`ssh-agent`" and "`ssh-add`" again.

What if ssh-agent automatically spawns each time you open a terminal? This can be done with a short scripts that uses ~/.agent.env to point to the currently running ssh-agent.

Do this by adding code to your ~/.profile file. If you have passphrases on your private keys you will need to enter them the first time you open a terminal window.

One final ssh-agent hack cont.

Credit to Rob Flickenger, *Linux Server Hacks*, pp. 144, O'Reilly books:

Code not yet included.

Awaiting author approval.

But, if you buy the book... :-)