Practical Lab Exercises

Lab- Internet and Web

Web Programming (F28WP)

Introduction

In this lab, you'll build upon your previous work to further develop your understanding of Internet and Web.

You'll need to be familiar with Linux (as it one of the most popular OS for web development) this includes remotely managing files and resources remotely. The following tutorial shows you how to login and remotely manage the Linux servers at Heriot-Watt University in Edinburgh (e.g., remotely managing web files for testing).

1.1 Remote Logins under Linux

The first part of this remote login exercise assumes you have direct login access to Unix hosts on a shared LAN. It uses the MACS Linux service on the Edinburgh campus for illustrative purposes. However, similar things can also be done on other LANs of Unix hosts. Skip to section 1.2 for the advice about PuTTY if you only have remote access to such a service from a Windows host.

Use an ssh client to log securely into another Unix (Linux) host on the LAN via SSL/TLS. To do this, run a virtual terminal client under X Window first. Then use your **own** login name and password instead of *humbert* and *Quilty* to login into *anubis*:

```
linux% ssh anubis -l humbert
humbert@anubis's password: Quilty
```

You can omit the "-l login_name" part if your login name on both hosts is the same. Now logout of *anubis* with the command *logout* or by issuing *Control D* together.

Now try the same thing with another host in the Linux lab. They have names like *linux19* in the range *linux01* to *linux86*. Not all of them may be switched on.

Remote login clients like *ssh* encrypt communications between the client and a remote ssh server using SSL/TLS. Those connections are secure so long as you are reasonably confident you are communicating with the host you think you are communicating with and not some attacker masquerading as it.

You can set up the means to identify hosts and be identified by them using ssh-keygen. The benefit will be not having to supply passwords every time you use *ssh* (or *scp*). The command *ssh-keygen* generates a public/private key pair and stores key data in a *.ssh* directory under your home directory. You only need to do this once. Please accept all the default options when running this command.

```
linux% ssh-keygen
```

The public key is put in *~/.ssh/id_rsa.pub* and the private key in *~/.ssh/id_rsa*. The "~" symbol stands for your home directory i.e. a path such as "/home/cs2/bozo" for the 2nd year Computer Science user *bozo*. The private key file *must* be owned by you, readable and writable by you and not readable or writable by anyone else.

```
linux% cd ~/.ssh
linux% ls -l id_rsa
-rw------- 1 bozo cs2 668 Sep 20 2017 id_rsa
```

If the file doesn't have these permissions, use *chmod 600 id_rsa* to give them to it. Now copy the public key file *id_rsa.pub* in the same directory to a file called *authorized_keys* in the same directory. It should be readable by anyone but only writable by yourself.

```
linux% cp id_rsa.pub authorized_keys
linux% ls -l authorized_keys
-rw-r--r-- 1 bozo cs2 668 Sep 20 2017 authorized_keys
```

From now on, whenever you try to log from host A into another departmental Unix machine host B with *ssh*, host A's *ssh* client will present this public key to the *sshd* daemon on host B. Then the daemon will mount your home directory, access *authorized_keys* and compare the presented public key to the keys in *authorized_keys* for a match. It will also check that a message from host A's *ssh* client signed by your private key decrypts with your presented public key. If all this works, *sshd* will run a shell command interpreter on host B that is connected by an SSL/TLS encrypted socket connection back to the *ssh* client on host A.

However, there remains the issue of you on host A authenticating host B. Is it host B or is it some attacker masquerading as host B in order to become a *man in the middle* snooping on your *ssh* interactions with host B. Authentication with *ssh* is done by asking you on host A to check that the fingerprint of the public key presented by the purported host B is correct. The *ssh* client on host A also checks that the presented public key can decrypt a message's signature from the host purporting to be B. Successful decryption proves that the message must be from you on host B as only you on host B should know the private key used to generate the signature.

After that first interaction *ssh* uses its stored record of that public key, you have vouched for, to check the authenticity of the host it is talking to. You no longer need be involved unless host B changes its public/private keypair.

For example the fictional user *humbert* might get the following interaction when he logs into *linux42* for the first time.

```
 linux% ssh linux42
 The authenticity of host 'linux42 (137.195.15.42)' can't be established.
 RSA key fingerprint is b7:cb:8e:74:de:84:01:55:0c:a6:0b:89:39:ac:db:07.
 Are you sure you want to continue connecting (yes/no)? yes
 Warning: Permanently added 'linux42,137.195.15.42' (RSA) to list of known hosts.
 humbert@linux42's password: Quilty
 Last login: Mon Jan 9 09:09:50 2017 from amaterasu
 linux42%
```

Here *humbert* has contributed to authenticating the host by accepting the given RSA key fingerprint as identifying that host's public key.

Now list the local hosts on your LAN known to the local NIS service with the command

```
linux% ypcat hosts | more
```

You can filter out servers with a command such as

```
linux% ypcat hosts | grep server
```

The MACS service on the Edinburgh campus doesn't let you log into most of these, but you can log into servers with names such as *amaterasu*, *anubis*, *jove*, *osiris*, *sif*, *thor* and ordinary hosts with names from *linux01* to *linux82*.

Once you are remotely logged in, prove this by running the command *hostname*. You can also try out commands such as *uname -a*, *who*, *top*, *ps -ax*, *df*, *uptime* which give information on the local host. To find out what these commands do, use the online manual *man* e.g.

```
linux% man uptime
```

Once you have set up *ssh* for use without passwords, you can also use scp to copy files securely between machines without supplying a password. Later in the semester you will find that a directory has been set up for you on *anubis* (aka *www2*) in the directory */var/lib/tomcat/webapps* so you can use the Tomcat web server. Its name is the same as your login name and you are its owner.

A MACS user *bozo* can copy files from their home directory to this anubis directory using *scp* as follows:

```
linux% cd ~/public_html
linux% scp page.html anubis:/var/lib/tomcat/webapps/bozo
```

The *scp* command copies the file *page.html* from their web folder under their home directory across to their directory on *anubis* if the file permissions are set correctly. If the copied file is readable by the Tomcat web server, the file after copying can be obtained over the web with the URL *http://www2.macs.hw.ac.uk:8080/bozo/page.html*.

Note the explicit use of a host name as well as the fully enumerated path in the *scp* command. Either or both of the 2 arguments to the *scp* command can be specified with a host name in this way to copy files between hosts on a LAN. For example you could copy a file *gubbins* from one Linux host's */tmp* directory to another as follows.

```
linux% scp linux15:/tmp/gubbins linux39:/tmp
gubbins 100% 0 0.3KB/s 00:00
Connection to linux15 closed.
```

The copying operation works because Unix users are normally allowed to write to the "/tmp" directory on any Unix host they can log into. However, you can't overwrite someone else's file in "/tmp".

Try using *scp* to copy files between Linux hosts. Remember that you must have permission to write to the destination directory as well as permission to login to the destination host.

You can even use *scp* to transfer files across the Internet from one Unix host to another. The receiving host must run an *sshd* daemon and be configured to accept such transfers. The following command run on a MacOS, Linux or other Unix host on the Internet supporting *scp*

```
linux% scp me.html bozo@jove.macs.hw.ac.uk:/home/cs2/bozo
password for bozo@jove.macs.hw.ac.uk: *******
me.html 100% 0 0.3KB/s 00:00
Connection to jove.macs.hw.ac.uk closed.
```

can copy a file *me.html* from elsewhere to the user *bozo*'s home directory on the host *jove* on the MACS LAN after authentication as *bozo*.

A valid login and password for the MACS LAN will be required. Also the host's *sshd* daemon on port 22 must be visible through the university firewall. Port 22 is open to the Internet for *osiris*, *amaterasu* and *jove* which all run *sshd* but closed for nearly every other MACS host.

1.2 Remote Logins under Windows

Under Windows you can remotely log into a (Unix) host that runs a secure shell daemon *sshd* using PuTTY.

Under the CS department Windows service at the Edinburgh campus PuTTY can be run from the start menu by selecting PuTTY followed by PuTTY. On other Windows hosts you can use PuTTY by first installing it from the given link and then running it.

Once PuTTY has popped up its start panel, type a computer name such as *amaterasu.macs.hw.ac.uk* into the Host Name box and make sure the Port box contains 22 and the Connection type is SSH. Then start a login session by clicking on the Open button. Supply your Linux login name in response to the "login as:" prompt and your Linux password to the subsequent prompt.