Using SSH for remote logins

Basic cryptography

Hans Peter Verne

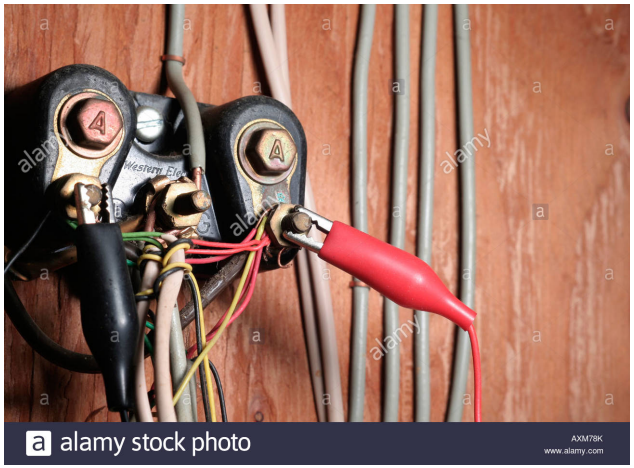
Geofag/UIO

2019-01-09

This presentation is an adaption from
http://www.mn.uio.no/geo/english/services/it/help/using-linux/ssh-tips-and-tricks.html

The Problem

Wiretapping



- Old Telephone System: A headset and crocodile clips.
- An intruder can listen in on the conversation and can impersonate either part.
- Internet: Only slightly more complicated.

SSH basics

SSH - the Secure Shell

- Enables secure communication.
- Drop-in replacment for the "remote shell" rsh (from the 80's).
- To log in:
  `$ ssh username@remote.host`
- To log in with X11 forwarding, assuming same username:
  `$ ssh -X remote.host`
- To copy a local file to the remote host:
  `$ scp localfile remote.host:remotefile`

Note the use of `$` as a shell prompt, and `remote.host` etc. as a placeholder.

Symmetric Cryptography

Disclaimer: I'm not a cryptographer.

All data can be assumed to be (sequences of) whole numbers.

- Function f is used to encrypt a message to crypto-text:

$$c = f_k(m)$$

- The inverse function g is used to decrypt:

$$m = g_k(c)$$

- The functions f and g are not secret.
- The index k is secret. We call it a *key*.
- k is typically 128–256 bits (ca. 10^{38} – 10^{77}).

Historical example, the Caesar shift

- Each letter in *m* is shifted *n* places up the alphabet.
- key is *n*.
- pseudocode, excluding rollover:

```
f(m,n) = foreach char x in m { print (x + n) }
g(c,n) = foreach char x in c { print (x - n) }
```

- Not a very good cryptographic algorithm.

Asymmetric Cryptography

- Two functions with different index:

$$c = p_i(m)$$
$$m = q_j(c)$$

- Inverse functions: $x = q_j(p_i(x)) = p_i(q_j(x))$
- Functions are not secret.
- Large indices (i, j), typically 2048 bits (ca. 10^{616}).
- The indices (i, j) can be generated as a pair, but you *cannot compute one from the other!*
- (More precise: You cannot compute i from j.)

Alice and Bob

- Alice wants to send secret messages to Bob.
- We suspect that some third party will listen.
- We suspect that some third party will try to impersonate Alice.
- (or Bob).

Session key

Alice and Bob can communicate with symmetric cryptography.
Remember: $c = f_k(m)$ and $m = g_k(c)$

- Alice and Bob need a secret key k.
- They can't just send the key in the clear...
- They want a one-time key (session key).

Session key negotiation (simplified)

Remember: $x = p_i(q_j(x))$ (asymmetric cryptography)

- Alice calls up Bob: «Hi, it's me, Alice!»
- Bob generates a key pair (i, j), sends Alice j.
- Alice chooses a random session key k, computes $K = q_j(k)$, and sends the result K back to Bob.
- Only Bob knows i and can compute the result, $k = p_i(K)$.
- Both Alice and Bob now have a session key k.

(ssh uses a more complex algorithm, but the idea is similar.)

Password authentication

Remember: $c = f_k(m)$

- Alice can now use k and encrypt any message and send to Bob.
- Bob can't know yet if he's really talking to Alice.
- Alice can send her password.

This is basic functionality in SSH, and also https (secure http).

Public key authentication

Getting rid of passwords all together!

- Alice generates a set of (i, j).
- i is her *private key*, and j is her *public key*.
- She keeps her private key (i) secret!
- She hands her public key (j) to Bob.

Public key authentication cont.

Remember: $x = q_j(p_i(x))$

- Alice calls up Bob:
  «Hi, it's me, Alice!»
- (Alice and Bob negotiate a session key k)
- How can Bob know it's really Alice?
- Bob picks a random number r, computes $R = q_j(r)$.
- Bob: «Hi, Alice. Here's R. Please compute $p_i(R)$?»
- Alice computes $r' = p_i(R)$, sends it back to Bob.
- If $r' = r$, Bob knows he's talking to Alice.

Putting it all together

Enough math!

- `ssh-keygen`
- Creates a key pair, stores them in
  `~/.ssh/id_rsa` and
  `~/.ssh/id_rsa.pub`
- Text files!
- The private key *must* be protected!
  `ssh-keygen` will prompt for a passphrase to encrypt it.
  Otherwise, anyone with access to the file can read your key.

Putting it all together, II

- Transfer the public key to the remote host:
  `$ scp .ssh/id_rsa.pub remote.host:tmp.key`
- Log in to the remote host:
  `$ ssh remote.host`
- Create the .ssh directory (if needed):
  `remote$ mkdir .ssh`
- Add the public key to the file .ssh/authorized_keys:
  `remote$ cat tmp.key >> .ssh/authorized_keys`
- Protect the files:
  `remote$ chmod go-rwx .ssh/ .ssh/authorized_keys`
- Remove the tmp file:
  `remote$ rm tmp.key`

Putting it all together, III

Now try

```
$ ssh remote.host
Enter passphrase for key (...):

Welcome to remote.host!
remote$
```

Problem: You are still asked for a password (of sorts).
Solution: Load the key in computer memory!

The agent

- The ssh agent is a computer program, running in the background.
- Your private key is an encrypted file.
- You decrypt the private key and upload it to the agent.
- The agent will provide the key to other programs that ask for it – ie. ssh.

The agent cont.

Do we have a running agent?

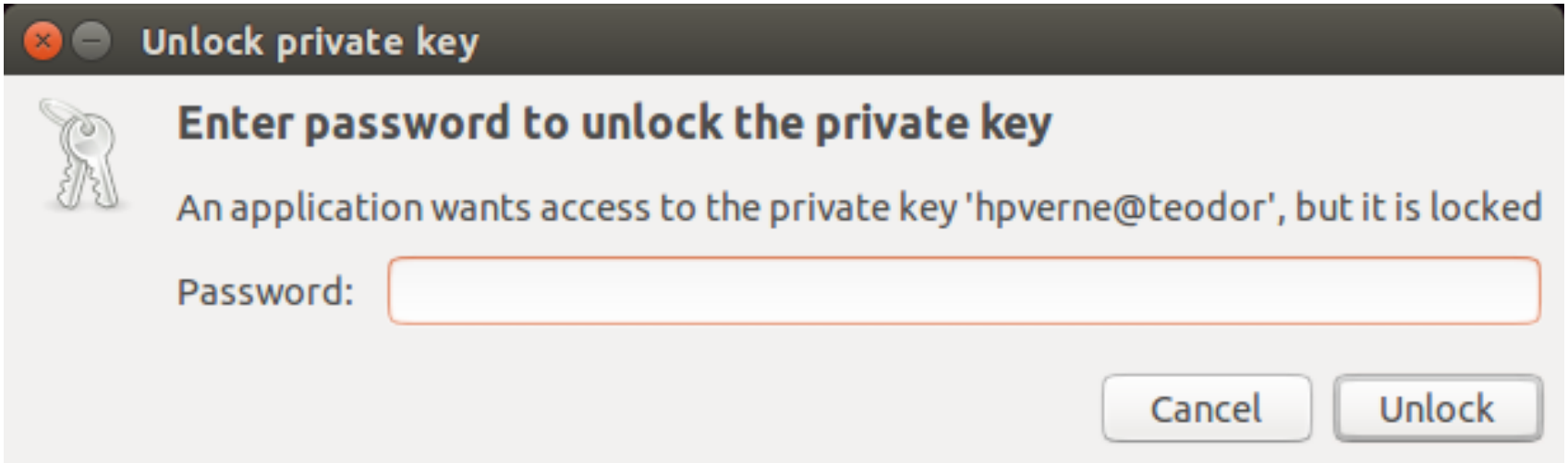
```
$ ssh-add -l
```

Possible replies:

- Could not open a connection to (. . .) agent.
  - The agent is not running.
- The agent has no identities.
  - The agent is running, but has no keys.
- 2048 4d:d0:c7:5e:(. . .) hpverne@tai.uio.no (RSA)
  - OK!

The agent II

In a desktop environment (Gnome/KDE/Mac), you probably already run a keyring manager. Just run ssh to upload the key.



(The «Password» prompt is misleading.
Type your key passphrase here.)

The agent III

- Otherwise, to start the agent, run:
  ```
  $ eval `ssh-agent`
  ```
- and, to upload the key:
  ```
  $ ssh-add
  Enter passphrase for key (...):
  ```

Once uploaded to the agent, you can log in without typing your password or passphrase again!

Class Discussion

- Q: You forgot the passphrase for the private key. What to do?
- Q: Your private key is compromized! What must be done?
  A: *Generate a new key pair, replace public key in* `.ssh/authorized_keys`.
- Q: Under what circumstances would an unencrypted private key be acceptable?
  A: *Other security measures, like whole-disk encryption? Also, depends on other restrictions on key?*

Key forwarding

```
$ ssh -A remote.host

remote$ ssh different.host
```

The ssh server (`sshd`) will act as an agent.

Only use it when you trust root on `remote.host`.

Host keys

- Private/public key pairs, used to identify hosts.
- Public host key transferred at first connection. You are asked to accept it.
- Messy when machine is reinstalled!

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED! @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

Is it plausible that the key should have changed?

Solution: ssh-keygen -R hostname -f known-hosts-file

Windows

Download putty from www.putty.org (actually `http://www.chiark.greenend.org.uk/%7Esgtatham/putty/`)

Choose the Windows Installer version, or standalone binaries:

- puTTY : ssh client and terminal emulator.
- PSCP : secure copy.
- PuTTYgen : Key generator.
- Pageant : ssh agent.

HOWTO:

`https://www.howtoforge.com/ssh_key_based_logins_putty`

Other Windows ssh clients:
tectia, MobaXterm, MTPuTTY, . . . Google it!

sshfs

Using ssh to access ("mount") a remote directory.

- Filesystem in userspace (fuse)
- Linux/Mac only. Additional packages required.

```
$ mkdir mountpoint
$ sshfs remote.host:/directory/ mountpoint/
$ fusermount -u mountpoint/
```

Do not use a mountpoint on a network drive
(like your network home directory)!

ssh tunnel: Ports

To access a remote TCP service, you need to know the port number. (Often implicit, or listed in `/etc/services`.)

Some commonly-used ports:

Service	Port	Description
ssh	22	Secure shell
http	80	Hyper-Text Transfer Protocol, ordinary web access
https	443	Secure HTTP
rdp	3389	Remote Desktop Protocol

Sometimes you can specify a different port, e.g.

```
$ xfreerdp -t 4000 server
$ ssh -p 443 login.uio.no
```

(real example!)

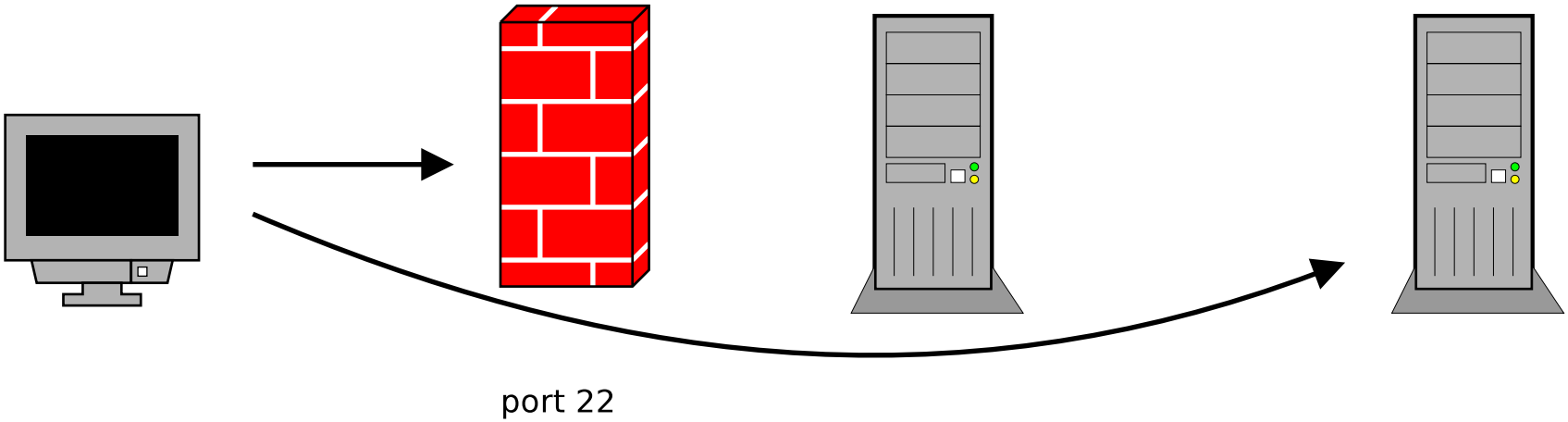
ssh tunnel: Firewall

The server (or the network equipment) might deny the connection to some ports, based upon the IP address (network) of the client.

```
home$ xfreerdp geo-all-hiperf.uio.no
unable to connect to geo-all-hiperf.uio.no:3389
```

ssh tunnel : port forwarding

There is perhaps a machine open for SSH traffic.



We want ssh to forward the traffic to another host.

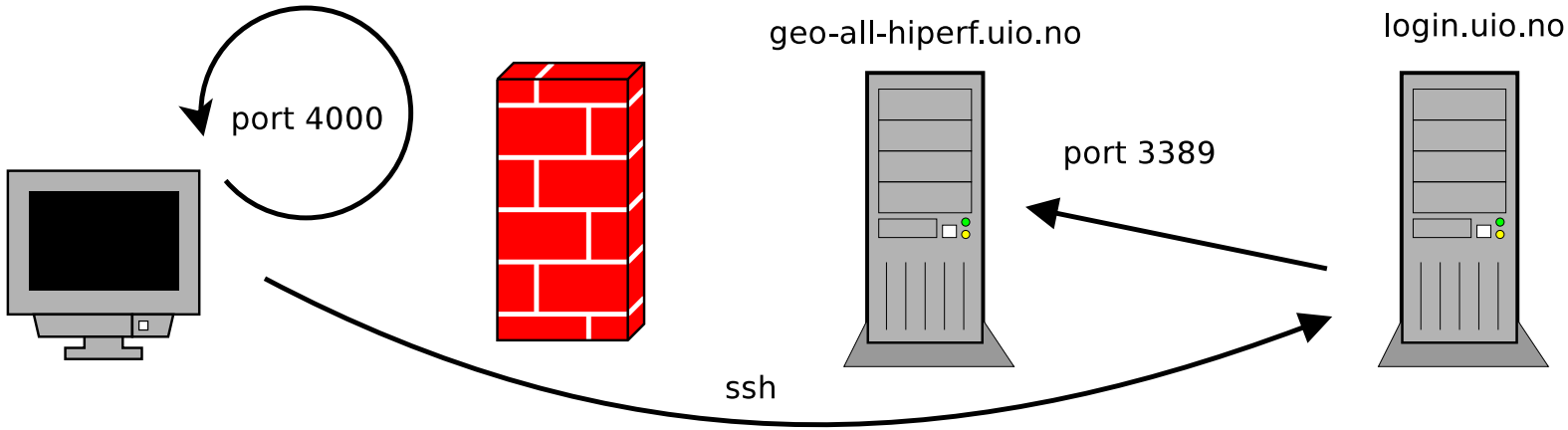
ssh tunnel setup

- Log in to sauron with ssh, ask ssh to forward local port 4000 to port 3389 («remote desktop») on geo-all-hiperf.uio.no:

```
$ ssh -L 4000:geo-all-hiperf.uio.no:3389 login.uio.no
```

- Now, in an other shell (terminal window):

```
$ xfreerdp -t 4000 -u username localhost
```



ssh tunnel warning

Caveat : services which do host authentication will protest!

```
$ xfreerdp -t 4000 -u hpverne localhost
connected to localhost:4000
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ WARNING: CERTIFICATE NAME MISMATCH! @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The hostname used for this connection (localhost)
does not match the name given in the certificate:
geo-all-hiperf.uio.no
```

ssh tunnel limits

You must be root on the local machine to forward ports < 1024.

```
$ ssh -L 80:www.uio.no:80 login.uio.no
Privileged ports can only be forwarded by root.
```

Java applets in the browser will likely not work.

Multiple port forwarding

- `$ ssh -L 8000:www.somewhere.no:80 \`
  `-L 4430:www.somewhere.no:443 login.uio.no`
- `http://localhost:8000/`
- `https://localhost:4430/`

Links within the document will perhaps not work...

SOCKS5

Better solution: Set up ssh as a SOCKS5 proxy:

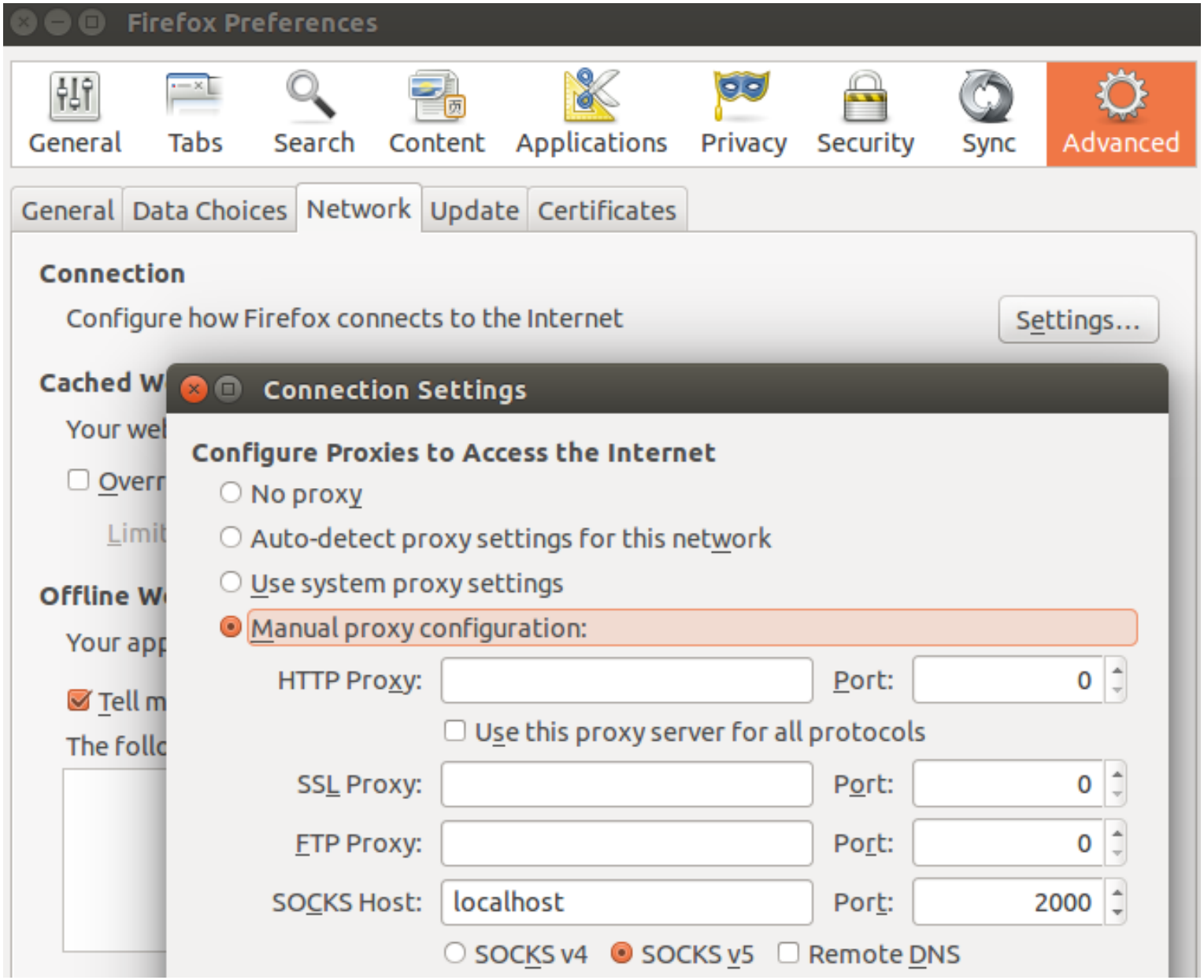
- `$ ssh -D2000 login.uio.no`

You must configure your client to use port 2000 on localhost as SOCKS5 proxy.

Google Chrome:

```
$ google-chrome -proxy-server="socks5://localhost:2000"\
-host-resolver-rules="MAP * 0.0.0.0 , EXCLUDE localhost"
```

Firefox SOCKS5 config



A final ssh tip

Annotate keys in `~/.ssh/authorized_keys`

```
ssh-rsa AAAAB3Nz.....GiWQe Your comments here
```

Remove no-longer needed keys the next time you edit the file.

Thank you!

Further reading:

- The manual pages,
  `ssh(1)`, `ssh-agent(1)`, `ssh-keygen(1)`, `ssh-add(1)`
- SSH config file `.ssh/config`,
  `ssh_config(5)`

- Questions? Discussion?