OpenSSH Tricks

Tom Ryder
tom@sanctum.geek.nz
https://sanctum.geek.nz/

What is SSH?

- Short for **Secure SHell**
- Run commands on other computers:
  - With **encryption**
  - With **authentication**
- Successor to unencrypted tools like `rlogin`
  - Designed as mostly-drop-in replacement

What is OpenSSH?

- OpenBSD's *implementation* of both client and server of the SSH protocol
- Free software (BSD license); also the default on GNU/Linux
- Very popular, but *not* the only SSH implementation

Typical SSH usage 1/2

- Run a user’s default shell on remote host with same username:

```
local$ ssh remote
```

- Doesn’t have to be a shell; other commands work too:

```
local$ ssh -t remote top
```

- For interactive use, a shell is the most convenient program.

Typical SSH usage 2/2

- Connect to a remote server with a different username:

```
local$ ssh -l friend remote
```

- Connect to a remote server running on a different port

```
local$ ssh -p 2222 remote
```

Installing the OpenSSH server

- **dpkg (Debian)**:
  ```
  remote$ sudo apt install openssh-server
  ```
- **rpm (Red Hat)**:
  ```
  remote$ sudo yum install openssh-server
  ```

Generally works out of the box; some security considerations with the defaults

Passwords are a pain

- Prompted for passwords on every connection:

```
local$ ssh remote
Password:
```

- This gets tiresome to type out every time.
- There are also some security issues with it...
  - *Especially* if you turn `StrictHostKeyChecking` off, or ignore the host key warnings...

    (plz dont)

Public/private keys

- Generate an SSH key:
  ```
  local$ ssh-keygen
  ```
  - Lots of options, but the defaults are OK
  - Decent strength RSA key (>= 2048 bits)
  - Named `user@local`
  - We are prompted for a **passphrase**...

Passphrase vs password

- The **passphrase** is used to encrypt the key, *not* to authenticate to the remote server.
- The remote server doesn’t see the passphrase.
  - This is already somewhat safer.

SSH agent 1/2

- Run an SSH agent for managing the key:

```
local$ eval "$(ssh-agent)"
Agent pid 2518
```

- Add your key:

```
local$ ssh-add
Enter passphrase for /home/ssh-demo/.ssh/id_rsa:
Identity added: /home/ssh-demo/.ssh/id_rsa (/home/ssh-
demo/.ssh/id_rsa)
```

- Check it’s ready:

```
local$ ssh-add -l
2048 SHA256:3v1Rf6ua3eTnjKQWbaSTWJJkXyK7dRgxAAVySGXuQKM
/home/ssh-demo/.ssh/id_rsa (RSA)
```

SSH agent 2/2

- Good to put into your `~/.profile` or `~/.bash_profile`

```
eval "$(ssh-agent)"
```

- Also check out **Keychain**: https://www.funtoo.org/Keychain

Copy the key 1/2

- Easy way:

```
local$ ssh-copy-id remote
```

- Provide the password (for the last time!)
- Connections after that point will use the key from the agent, without needing a password:

```
local$ ssh remote
remote$
```

Copy the key 2/2

- Under the hood, all `ssh-copy-id` does is:
    1. Retrieve the public key loaded by `ssh-agent`
    2. Add it to the end of `remote:.ssh/authorized_keys`
- That’s all!
- You can do this manually, but it’s a bit inconvenient and error-prone (especially for long keys)

Go key-only

- Now we can turn off password authentication in `remote:/etc/ssh/sshd_config`:

  ```
  PasswordAuthentication no
  ```

- Now only keys will be accepted (careful!)
- You don’t have to worry about being brute-forced anymore
- Well, except for robots filling up your logs...

  Hint: `apt install fail2ban`

Client config files 1/3

- Remembering or typing out connection details can be a pain:

```
local$ ssh remote1

local$ ssh -l friend remote2

local$ ssh -p 2222 remote3

local$ ssh remote4.ssh-servers.annoyingly-long-
domain-name.net.nz
```

Client config files 2/3

- Put the details in `~/.ssh/config` instead:

```
Host remote1
Host remote2
  User friend
Host remote3
  Port 2222
Host remote4
  HostName remote4.ssh-servers.annoyingly-long-domain-name.net.nz
```

Client config files 3/3

- Ahh, much better!

```
local$ ssh remote1
local$ ssh remote2
local$ ssh remote3
local$ ssh remote4
```

- Way better than copy-pasting commands or searching `history`...

Port forwarding 1/3

- Forward TCP traffic on a local port to a remote port accessible to the SSH server.

  ...that is to say...

- Make a port on your computer act like a port on the other end; *masquerade* as the remote host.

Port forwarding 2/3

- My MariaDB server only listens locally.
- I can’t just connect to it straight from the internet (good!)
- But I want to connect to it from a GUI program on my trusted computer...

Port forwarding 3/3

```
local$ ssh -L 3306:localhost:3306 remote
remote$
```

Then, in another terminal:

```
local$ mariadb
MariaDB>
```

SOCKS5 proxy 1/4

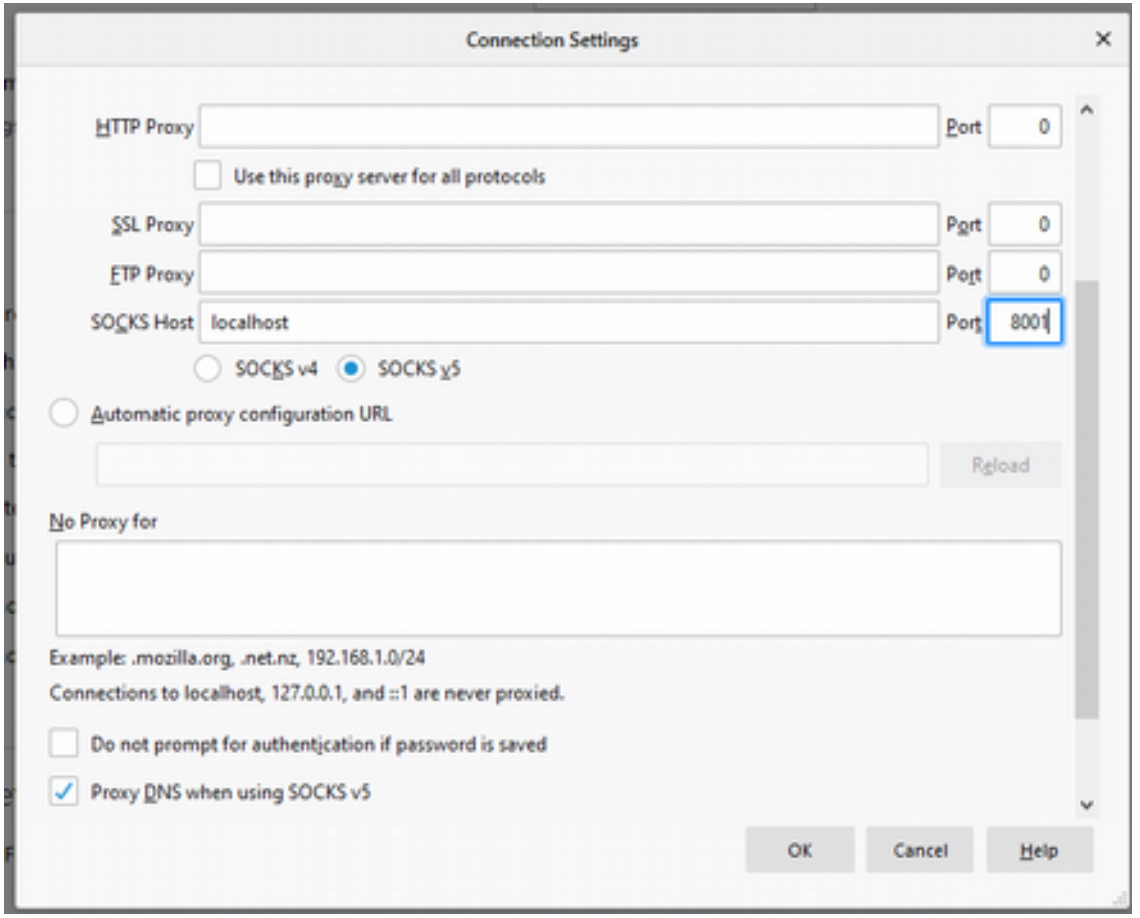
- Tom’s favourite trick!
- Pass *all* traffic for an application through the SSH tunnel
- Allows you to treat any SSH server to which you have access as a browser proxy server
  - **Hint:** Hosting providers generally leave it on!

SOCKS5 proxy 2/4

```
local$ ssh -D8001 remote
remote$
```

- SOCKS5 proxy port is now listening on localhost port 8001

SOCKS5 proxy 3/4



SOCKS5 proxy 4/4

Web and DNS traffic now goes via the SSH server!

- Yes, **including HTTPS**
- Super handy when your designated VPN into a remote site breaks

  (long story)

Questions?

- OpenSSH project site: https://www.openssh.com/
- Manual pages: https://www.openssh.com/manual.html
- More about SSH keys:
  https://sanctum.geek.nz/arabesque/gnu-linux-crypto-ssh-keys/
- More about `~/.ssh/config`:
  https://sanctum.geek.nz/arabesque/uses-for-ssh-config/

Email: tom@sanctum.geek.nz
Website: https://sanctum.geek.nz/
Twitter: @tejrnz
Fediverse: @tejrnz@soc.fglt.nl