pentestmonkey

Taking the monkey work out of pentesting

SSH Cheat Sheet

SSH has several features that are useful during pentesting and auditing. This page aims to remind us of the syntax for the most useful features.

NB: This page does not attempt to replace the man page for pentesters, only to supplement it with some pertinent examples.

SOCKS Proxy

Set up a SOCKS proxy on 127.0.0.1:1080 that lets you pivot through the remote host (10.0.0.1):

Command line:

```
ssh -D 127.0.0.1:1080 10.0.0.1
```

~/.ssh/config:

```
Host 10.0.0.1
DynamicForward 127.0.0.1:1080
```

You can then use tsocks or similar to use non-SOCKS-aware tools on hosts accessible from 10.0.0.1:

```
tsocks rdesktop 10.0.0.2
```

Local Forwarding

Make services on the remote network accessible to your host via a local listener.

NB: Remember that you need to be root to bind to TCP port <1024. Higher ports are used in the examples below.

Example 1

The service running on the remote host on TCP port 1521 is accessible by connecting to 10521 on the SSH client system.

Command line:

```
ssh -L 127.0.0.1:10521:127.0.0.1:1521 user@10.0.0.1
```

~/.ssh/config:

```
LocalForward 127.0.0.1:10521 127.0.0.1:1521
```

Example 2

Same thing, but other hosts on the same network as the SSH client can also connect to the remote service (can be insecure).

Command line:

```
ssh -L 0.0.0.0:10521:127.0.0.1:1521 10.0.0.1
```

~/.ssh/config:

```
LocalForward 0.0.0.0:10521 127.0.0.1:1521
```

Example 3

In this example, 10.0.0.99 is a host that's accessible from the SSH server. We can access the service it's running on TCP port 1521 by connecting to 10521 on the SSH client.

Command line:

```
ssh -L 127.0.0.1:10521:10.0.0.99:1521 10.0.0.1
```

~/.ssh/config:

```
LocalForward 127.0.0.1:10521 10.0.0.99:1521
```

Remote Forwarding

Make services on your local system / local network accessible to the remote host via a remote listener. This sounds like an odd thing to want to do, but perhaps you want to expose a services that lets you download your tools.

NB: Remember that you need to be root to bind to TCP port <1024. Higher ports are used in the examples below.

Example 1

The SSH server will be able to access TCP port 80 on the SSH client by connecting to 127.0.0.1:8000 on the SSH server.

Command line:

```
ssh -R 127.0.0.1:8000:127.0.0.1:80 10.0.0.1
```

~/.ssh/config:

```
RemoteForward 127.0.0.1:8000 127.0.0.1:80
```

Example 2

The SSH server will be able to access TCP port 80 on 172.16.0.99 (a host accessible from the SSH client) by connecting to 127.0.0.1:8000 on the SSH server.

Command line:

```
ssh -R 127.0.0.1:8000:172.16.0.99:80 10.0.0.1
```

~/.ssh/config:

```
RemoteForward 127.0.0.1:8000 172.16.0.99:80
```

Example 3

The SSH server will be able to access TCP port 80 on 172.16.0.99 (a host accessible from the SSH client) by connecting to TCP port 8000 on the SSH server. Any other hosts able to connect to TCP port 8000 on the SSH server will also be able to access 172.16.0.99:80. This can sometimes be insecure.

Command line:

```
ssh -R 0.0.0.0:8000:172.16.0.99:80 10.0.0.1
```

~/.ssh/config:

```
RemoteForward 0.0.0.0:8000 172.16.0.99:80
```

Configuration Files

~/.ssh/config

It’s sometimes easier to configure options on your SSH client system in ~/.ssh/config for hosts you use a lot rather than having to type out long command lines.

Using ~/.ssh/config also makes it easier to use other tools that use SSH (e.g. scp and rsync). It's possible to tell other tools that SSH listens on a different port, but it's a pain.

```
Host 10.0.0.1
Port 2222
User ptm
ForwardX11 yes
DynamicForward 127.0.0.1:1080
RemoteForward 80 127.0.0.1:8000
LocalForward 1521 10.0.0.99:1521
```

The above lines are explained more fully in the other subsection on this page.

~/.ssh/authozied_keys

During a pentest or audit, you might want to add an authorized_keys file to let you log in using an SSH key.

The authorized_keys file lives in a user's home directory on the SSH server. It holds the public keys of the users allowed to log into that user's account.

Generate a public/private key pair like this:

```
ssh-keygen -f mykey
cat mykey.pub # you can copy this to authorized_keys
```

If you want to shortest possible key (because your arbitrary-file-write vector is limited), do this:

```
ssh-keygen -f mykey -t rsa -b 768
cat mykey.pub # copy to authorized_key.  Omit the trailing user@host if you need a shorter key.
```

Connect to the target system like this (you need to know the username of the user you added an authorized key for):

```
ssh -i mykey user@10.0.0.1
```

Caveat: The authorized_keys file might not work if it's writable by other users. If you already have shell access you can "chmod 600 ~/.ssh/authorized_keys". However, if you're remotely exploiting an arbitrary file-write vulnerability and happen to have a weak umask, you may have problems.

X11 Forwarding

If your SSH client is also an X-Server then you can launch X-clients (e.g. Firefox) inside your SSH session and display them on your X-Server. This works well with from Linux X-Servers and from cygwin's X-server on Windows.

Command Line:

```
SSH -X 10.0.0.1
SSH -Y 10.0.0.1 # less secure alternative - but faster
```

~/.ssh/config:

```
ForwardX11 yes
ForwardX11Trusted yes # less secure alternative - but faster
```

SSH Agents

SSH agents can be used to hold your private SSH keys in memory. The agent will then authenticate you to any hosts that trust your SSH key.

This has the following advantages:

- You don't have to keep entering your passphrase (if you chose to encrypt your private key)
- But you still get to store your private SSH key in an encrypted format on disk.

Using an SSH agent is probably more secure than storing your key in cleartext, but agents can be hijacked.

Using an SSH Agent

First start your agent:

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

Then add your keys to it – you'll need to enter your passphrase for any encrypted keys:

```
ssh-add ~/dir/mykey
```

Hijacking SSH Agents

If you see SSH agents running on a pentest (process called "ssh-agent"), you might be able to use it to authenticate you to other hosts – or other accounts on that host. Check out ~/.ssh/known_hosts for some ideas of where you might be able to connect to.

You can use any agents running under the account you compromised. If you're root you can use any SSH agent.

SSH agents listen on a unix socket. You need to figure where this is for each agent (e.g. /tmp/ssh-tqiEl28473/agent.28473). You can then use the agent like this:

```
export  SSH_AUTH_SOCK=/tmp/ssh-tqiEl28473/agent.28473
ssh-add -l # lists the keys loaded into the agent
ssh user@host # will authenticate you if server trusts key in agent
```

This command illustrates how you could inspect the environment of every ssh-agent process on a Linux system. It should yield a list of unix sockets for SSH agents.

```
ps auxeww | grep ssh-agent | grep SSH_AUTH_SOCK | sed 's/.*SSH_AUTH_SOCK=//' | cut -f 1 -d ' '
```

Agent Forwarding

If you enable SSH agent forwarding then you'll be able to carry on using the SSH agent on your SSH client during your session on the SSH server. This is potentially insecure because so will anyone else who is root on the SSH server you're connected to. Avoid using this feature with any keys you care about.

Tags: pentest, ssh

Posted in Cheat Sheets