

Secure HUNT (shunt.py) and reverse ssh tunnel maker

Version 0.1.0 — copyright Kalvis M Jansons (2006)

The main purpose of shunt.py is, when given a list of hosts, to find a route through those hosts, using ssh, to one more more targets, and then set up a reverse ssh tunnel back to the starting point.

The reason that you might want to do this, and the reason that I wrote shunt.py, was so that I could script a computer behind a firewall to find a connection to the outside world and make a reverse ssh tunnel back to its ssh demon port so I could then log in from the outside world. The reason I needed the hunting route aspect was that many, particularly Apple networks, are almost randomly connected together (just because they can be), and I wanted the computer behind the firewall to be sure to get a connection out.

There are a few comments in the source code that will also help in local customization, but here we give a few random examples of its use.

Before we start, make sure that shunt.py is in your path and it is executable, or in the examples below replace "shunt.py" with "python shunt.py"

Example 1

```
shunt.py a b - e
ssh -tAq -R2332:localhost:22 e
ssh -tAq -R2400:localhost:22 a ssh -tAq -R2697:localhost:2400 e ssh -tAq -R2547:localhost:22 b
ssh -tAq -R3039:localhost:2547 e ssh -tAq -R2786:localhost:22 a ssh -tAq -R3191:localhost:2786
b ssh -tAq -R2305:localhost:3 191 e
ssh -tAq -R2970:localhost:22 b ssh -tAq -R3094:localhost:2970 a ssh -tAq -R2530:localhost:3
094 e
```

Here, shunt.py is trying to connect to host e and it is allowed to route through any combination of a and b. It uses random high ports for the ssh tunnel that it makes back to the sshd port of the starting host, and prints out the command lines as it goes. In this case, it failed as there is no host e to connect to, so we can see all of the routes it would try. Note however the ports it was trying to set up on e to tunnel back to the starting host were also random, and normally we would like to set this value. This can be done if we replace e with, say, 2222::e as in the next example.

Example 2

```
shunt.py a b - 2222::e
ssh -tAq -R2222:localhost:22 e
ssh -tAq -R2786:localhost:22 a ssh -tAq -R2222:localhost:2786 e ssh -tAq -R2430:localhost:22 b
ssh -tAq -R2222:localhost:2430 e ssh -tAq -R2270:localhost:22 a ssh -tAq -R2229:localhost:2270
b ssh -tAq -R2222:localhost:2 229 e
ssh -tAq -R2793:localhost:22 b ssh -tAq -R2536:localhost:2793 a ssh -tAq -R2222:localhost:2
536 e
```

Now the tunnels are from port 2222 on e back to 22 on the starting host.

Example 3

Other hosts can have these ports set too:

```
shunt.py a 2211::b - 2222::e
ssh -tAq -R2222:localhost:22 e
ssh -tAq -R2211:localhost:22 b ssh -tAq -R2222:localhost:2211 e ssh -tAq -R2807:localhost:22 a
ssh -tAq -R2222:localhost:2807 e ssh -tAq -R2211:localhost:22 b ssh -tAq -R2538:localhost:2211
a ssh -tAq -R2222:localhost:2 538 e
ssh -tAq -R2611:localhost:22 a ssh -tAq -R2211:localhost:2611 b ssh -tAq -R2222:localhost:2
211 e
```

Notice now the port 2211 is used on b, but on a we still use random ports.

Example 4

The route fragments do not need to be hosts, but could be something like:

```
"9999::ikf ssh -tAq -R9999:localhost:9999 talisman"
```

where port 9999 is used on host ikf and that host is to be followed by host talisman, and we have been careful to connect the ports for this section so the reverse route made by shunt.py will still work.

Example 5

There can be more than one target and these can also be jobs:

```
shunt.py a b c d - 2222::e "2223::f my-program" "g another-program"
```

In this example, shunt will use a b c d for routes and first connect to e and then f and run my-program and then run another-program on g, using the special ports where given.

Note also that we use a linear hold-off, starting with 1 second with increments of 1 second, so we do not look like we are doing a port scan or cracking the target system.

You can also, obviously, use all the standard ssh options at each stage.

Comments

This is a very early release and a lot will be added later, but it is already useful. If you think of some good additions, either send me an updated program and if I like it I will make it part of the next version, or, if you are not a programmer, just tell me what you would like it to do.

Once you have a reverse tunnel, from port 2222 on host e back to the firewalled machine say, you can connect to it using

```
ssh -p2222 -o"HostKeyAlias something" localhost
```

where the "something" could be the hostname of the firewalled machine or just random, but stops the ssh program worrying about the unexpected host key it will find. These options can also be put in your .ssh/config file (see the man pages) to save you typing them each time.

We can use shunt.py also as a python module, which is handy when trying to improve it. In python, just run either of the following:

```
import shunt
```

or

```
from shunt import *
```

Contact

The contact email address is shunt@kalvis.com