# Setting up an ICMPtunnel demo on Docker

Following are my notes on setting up a basic 5-container network to demonstrate the use of an ICMPtunnel. The architecture is based on vanilla Ubuntu containers, OpenVSwitch, iproute2, and the icmptunnel utility by Dhaval Kapil. Links to source material are at the end of this file.

I've attempted to obfuscate much of the architecture generation stuff by automating it. It will generate the architecture depicted in the architecture.png file.

The idea for creating this demo arose out of discussions during the Friday night TCC Cyber Club meeting, where a desire was expressed to have more pcaps to analyze (students had just participated in the NCLs and a particularly annoying question involved recovering ICMPtunnel content that had been captured in a pcap.

## Assumptions/Disclaimers

- Unless otherwise noted, all commands in the below are executed as root.
- It is assumed that you understand the basic use of Docker (the build, run, and exec functions).
- This demo is intended for use by Tidewater Community College's Cyber Club (TC4), for use in creating additional CTF challenges,
- This demo is intended to be built on your desktop machine, which is running Docker and OpenVSwitch. This is because the build script installs a Wireshark container which you will access via http://127.0.0.1:3001. For Ubuntu users, Dpcler and OpenVSwitch can be installed via:

  ```
  apt-get install -y docker.io openvswitch-switch
  ```

  All other binaries will be installed via the scripts in this repo.
- Is is recommended that your read and underrstand each script/file before running any of the scripts. I've added commentary to each, to help explain what each command does.

## Steps for setup

1) If the files aren't already executable, run the following:

   ```
   chmod a+x build build-images client destroy destory-images proxy
   ```

2) Create the images via:

   ```
   ./build-images
   ```

   Note: the first time that you run this, it will take a couple minutes to build the 5 local images.

3) Deploy the containers by running the build script:

   ```
   ./build
   ```

4) Check that all 5 containers (wireshark, boxa, boxb, boxc, and boxd) have been deployed, by running:

   ```
   docker ps
   ```

   If not all 5 are running, scroll back through the "./build" script's output and look for errors. If that doesn't show anything untowards, try running:

   ```
   docker logs container_name
   ```

   where "container_name" is the name of the missing container.

5) Access the command line of the proxy (running the server end of the tunnel), and create the server end of the tunnel by running:

```
nohup ./icmptunnel -s 10.0.1.1 &
exit
```

Note: I've created a couple scripts (client, proxy), to access the containers, that will save keystrokes. Look at their source code to see how they work.

6) To create the "local" end of the tunnel, access the command line of the client and run:

```
nohup ./icmptunnel -c 10.2.2.2 &
```

7) Point your browser at http://127.0.0.1:3001 and resize the window as desired. Select eth1 as the interface.

8) Back in the client command line, run the following:

```
lynx http://192.168.9.2/images.jpeg
```

Follow the prompts to download the file to disk. If you receive a file called images.jpeg, that is 5662 in size, then it worked. Take a look at your wireshark display. You should notice that the file transfer was made over ICMP.

## Additional research

1) In the packet capture, you'll see some spurious echo requests with a "no response found!" message at the end. I think this might be caused by not disabling ICMP response from the containers hosting either end of the tunnel. If anyone's interested in experimenting, try configuring the container to ignore ICMP (the software will still respond to received ICMP packets). See: https://github.com/jamesbarlow/icmptunnel for an example.

2) You may also see some IPv6 traffic in the pcap. Figure out how to turn off IPv6 in client and proxy ends.

3) Experiment with tunneling other tools and protocols, pushed through the ICMP tunnel.

4) For a bit of realism (with #3), add filters to the firewall (via iptables statements) that prevent any traffic other than ICMP.

5) Develop a process and/or tool that recovers the content from the captured ICMP traffic. See example at: https://www.boiteaklou.fr/Data-exfiltration-with-PING-ICMP-NDH16.html

## Additional Reading

- Detecting IMCP tunnels with Wireshark - https://medium.com/@alpinoacademy/detecting-icmp-tunnel-with-wireshark-7c8fc6baa77a
- https://www.boiteaklou.fr/Data-exfiltration-with-PING-ICMP-NDH16.html
- https://www.giac.org/paper/gcfa/85/piping-shell-icmp-tunnel-a-forensic-study-malicious-code/104357

## Other (unrelated)

1) While researching ICMP tunnels, I stumbled across what looks like the cruft from a series of attacks on various web sites. If anyone wants to look at it, perform a Google search for "Port Knocking Script Github" (with the quotes). I found that while researching the addition of port knocking (protocol knocking?), ICMP tunnels, and ICMP tunnel traffic recovery.

2) Wait! What? There was a backdoor discovered in SOCAT in Feb 2016? See

- https://www.openwall.com/lists/oss-security/2016/02/01/4
- https://github.com/mimoo/Diffie-Hellman_Backdoor
- https://github.com/mimoo/Diffie-Hellman_Backdoor/tree/master/socat_reverse

## Sources

- https://dhavalkapil.com/icmptunnel/
- https://github.com/DhavalKapil/icmptunnel/
- https://github.com/jamesbarlow/icmptunnel
- architecture image was generated via: https://app.diagrams.net/