DNS TUNNELING WITH IODINE

I have had the privilege of traveling around the world a bit (for work, but still) and I have seen many "*public*" internet setups that require registration and/or payment. Some of these I have no problem with, but I have found a lot that try to get you on with false advertising or refund offers that go nowhere. And sometimes, for various reasons, it may not be a good idea to register. For those times, you may still be able to connect and one way of doing that is tunneling data through DNS packets. This is where "*iodine*" comes in, it is used as both server and client. Because in this type of setup you do need your own server to connect to and then outwards from there. Lets take a look..

The problem
To make sure we are testing it right, I setup my client machine to disallow SSH and HTTP connections:

```
# links www.google.ca -dump
Connection refused

# ssh 192.168.2.104
ssh: connect to host 192.168.2.104 port 22: Connection refused
```

..this is what we want to bypass.

Getting started
You need 3 things to get started:

1. A "*server*" machine. This is where you will run the iodine "*server*" component and it needs to be publicly reachable.
2. You need to setup this machine to have a DNS (*NS*) record publicly for a specific domain. This is so when the client queries, it gets to the right place
3. The *iodine* software (see here)

Getting started with the DNS records
Even if you have a hosted domain and site, there are not many providers that will allow to create a custom *A* and *NS* record. So assuming you cannot do that with what you have currently, you need to head to freedns.afraid.org. This site only does DNS records, but it does it well, it does it free (for small amounts) and does not limit you - in short, a great site. So sign-up for free and then create an "*A*" record using one of their available domains (see, said it was cool), this *A* record will have the public IP address of the server you are using. Then create a *NS* record that points to the *A* record you just created. At the end you will have something like:

```
bobpub.mooo.com - A - xx.xx.xx.xx
bobns.mooo.com - NS - bobpub.mooo.com
```

Getting the server running
Once you downloaded the software, do the "*make*" and "*make install*" two-step. This will get you the server component "*iodined*" and the client component "*iodine*". I normally run the server component in a separate "*screen*" session. For this reason I use the "*-f*" to keep it in the foreground. I also use the "*-P*" to specify a password for use. I have found using the "*-c*" option is very useful for better connectivity. Then since *iodined* will create a virtual interface you give it a IP address to use (I used 10.0.0.1). Then finally, give it the name of the NS record you created. You should see something like:

```
# /usr/local/sbin/iodined -c -f -D -P test 10.0.0.1 bobns.mooo.com
Debug level 1 enabled, will stay in foreground.
Add more -D switches to set higher debug level.
Opened dns0
Setting IP of dns0 to 10.0.0.1
Setting MTU of dns0 to 1130
```

```
Opened UDP socket
Listening to dns for domain bobns.mooo.com
```

Starting the client
Now on the client machine, lets start the iodine client and try to connect to the server:

```
# iodine -P test bobns.mooo.com
Opened dns0
Opened UDP socket
Sending DNS queries for bobns.mooo.com to 192.168.2.1
Autodetecting DNS query type (use -T to override).
Using DNS type NULL queries
Version ok, both using protocol v 0x00000502. You are user #1
Setting IP of dns0 to 10.0.0.3
Setting MTU of dns0 to 1130
Server tunnel IP is 10.0.0.1
Testing raw UDP data to the server (skip with -r)
Server is at 192.168.2.104, trying raw login: OK
Sending raw traffic directly to 192.168.2.104
Connection setup complete, transmitting data.
Detaching from terminal...
```

..on the server side you should see logs like these:

```
IN   login raw, len 16, from user 0
IN   ping raw, from user 0
IN   ping raw, from user 0
```

Using it
Now we have a connection setup, we need to somehow make use of it. Now the simplest way to do this is to use SSH to create a socks proxy for us to use. This will also allow us to test if we can now SSH since the last time we tried it was blocked in this test setup. So in one session on our client machine:

```
# ssh -D 1080 10.0.0.1
root@10.0.0.1's password:
Last login:  from 10.0.0.3
[root@localhost ~]#
```

This shows us we can SSH now. It also setups the socks proxy we can use for testing our web access. So since I am using the CLI, I setup *proxychains* to use this new proxy and then try to web browse:

```
# proxychains links www.google.ca -dump
ProxyChains-3.1 (http://proxychains.sf.net)
|DNS-request| www.google.ca
|S-chain|-<>-127.0.0.1:1080-<><>-4.2.2.2:53-<><>-OK
|DNS-response| www.google.ca is 173.194.38.152
|S-chain|-<>-127.0.0.1:1080-<><>-173.194.38.152:80-<><>-OK
   _____________________________________________________
   _____________________________________________________
   _____________________________________________________
   _____________________________________________________
   _____________________________________________________
   _____________________________________________________
   _____________________________________________________
   Search Images Maps Play YouTube News Gmail Drive More >>
   Web History | Settings | Sign in
```

Canada

Advanced
[Google Search] [I'm Feeling Lucky] searchLanguage
tools

Google.ca offered in: Franc,ais

Advertising ProgramsBusiness Solutions+GoogleAbout GoogleGoogle.com

There we go, a successful socks proxy through our DNS tunnel bypassing the blocks.

Final thoughts

The DNS tunnel will be slower then a normal connection since it uses UDP and the packet sizes are dependent on the DNS traffic allowed. Also the DNS tunnel can be used for any traffic, not just the SSH example I showed. Remember this type of thing works because folks do not setup good gateways, or take some vendors word for some equipment being a silver bullet. Regardless of why, I love the ingenuity in these types of setups - so try it out, have fun and learn.