Proxying DNS with python (Nov 13 2012)

So a while ago I needed to proxy my python requests over a SOCKS4/5 proxy, so I started digging into some modules to do that.

I came across the nice socks module, which would allow me to basically monkey-patch the socket used by any other module whenever I needed to. Quite handy for sure!

This example shows how to proxy a urllib2.urlopen() call through a proxy listening on port 4444 locally (in my case ssh -D 4444).

```
import socks
import urllib2

# Set default proxy
socks.setdefaultproxy(socks.PROXY_TYPE_SOCKS5, 'localhost', 4444)
# Wrap our module
socks.wrapmodule(urllib2)
# Make our request
print json.loads(urllib2.urlopen('http://ifconfig.me/all.json').read())
```

Fine and dandy, right? Well sure, as long as we don't need to proxy our DNS requests. An example would be when you're trying to communcate on the TOR network, or you want to **actually** send all your traffic through a proxy. If you don't, then all your DNS requests get sent to your default DNS server. This can obviously be bad, since DNS doesn't use any encryption, so an attacker sitting in the middle of your connection will still be able to make an educated guess as to what you're doing, and even subvert your requests! Not cool.

So I posted a question over at StackOverflow, and kept trying to figure out how to get it working. No matter what I did, nothing seemed to work.

Until I started thinking about how modules are imported.

What if when a module is initialized it sets up all it's socket stuff? That's what I would do, as that's a very setup-y thing to do!

Tinkering around, I finally found something that worked:

```
import socks
import socket

# Can be socks4/5
socks.setdefaultproxy(socks.PROXY_TYPE_SOCKS4,'127.0.0.1', 9050)
socket.socket = socks.socksocket

# Magic!
def getaddrinfo(*args):
    return [(socket.AF_INET, socket.SOCK_STREAM, 6, '', (args[0], args[1]))]
socket.getaddrinfo = getaddrinfo

import urllib
# Do some stuff with proxied urllib
```

This proxies all DNS requests through your proxy server, and works cross-platform on Win, Linux, and probably OSX (not tested).

Here's a nice wrapper function to set/reset proxies for you, pretty useless but you get the idea.

```
import socks
import socket
orig_sock     = socket.socket

# This is the way we monkey patch! yay!
def getaddrinfo(*args):
    return [(socket.AF_INET, socket.SOCK_STREAM, 6, '', (args[0], args[1]))]
socket.getaddrinfo = getaddrinfo

import urllib

# Set our current proxy
def setProxy(type=socks.PROXY_TYPE_SOCKS5, host="127.0.0.1", port="9050"):
    socks.setdefaultproxy(type, host, port)
    socket.socket = socks.socksocket

# Reset proxy to our original socket (no proxy)
def unsetProxy():
    socket.socket = orig_sock

# Check tor
def checkTor():
    if "Sorry" in urllib.urlopen('https://check.torproject.org/').read():
        return False
    else:
        return True

# Set our proxy to a local SOCKS5 listening on 4444
setProxy(type=socks.PROXY_TYPE_SOCKS5, host='127.0.0.1', port=4444)
print "Using tor? [" + str(checkTor()) + "]"
print urllib.urlopen('http://ifconfig.me/all.json').read()

# Set our proxy to tor running locally
setProxy(type=socks.PROXY_TYPE_SOCKS5, host='127.0.0.1', port=9050)
print "Using tor? [" + str(checkTor()) + "]"
print urllib.urlopen('http://ifconfig.me/all.json').read()

# Reset proxy settings
unsetProxy()
print "Using tor? [" + str(checkTor()) + "]"
print urllib.urlopen('http://ifconfig.me/all.json').read()
```

Hope someone finds it useful! I was pulling my hair out for a while about it, and monkey-patching the socket module is definitely not for the weak of will.