A Few Advanced Network Topics

Introduction

- Python provides a variety of libraries for simplifying network programming
  - urllib
  - xmlrpc
  - SocketServer
  - Many others
- However, there are some tricky issues

Overview

- Topics covered
  - Advanced urllib (HTTP)
  - SSL/crypto
- A stepping stone for further work

urllib Module

- Provide client access to web services

```
>>> from urllib import urlopen
>>> u = urlopen("http://download.finance.yahoo.com/d/
quotes.csv?s=IBM&f=sl1")
>>> data = u.read()
>>>
```

- Supports HTTP, HTTPS, and FTP

```
u = urllib.urlopen("http://www.foo.com")
u = urllib.urlopen("https://www.foo.com/private")
u = urllib.urlopen("ftp://ftp.foo.com/README")
```

HTML Forms

- urllib can submit form data

Your name:
Your email:
Subscribe

- Example HTML source for the form

```
<FORM ACTION="/subscribe" METHOD="POST">
Your name: <INPUT type="text" name="name" size="30"><br>
Your email: <INPUT type="text" name="email" size="30"><br>
<INPUT type="submit" name="submit-button" value="Subscribe">
```

HTML Forms

- Within the form, you will find an action and named parameters for the form fields

```
<FORM ACTION="/subscribe" METHOD="POST">
Your name: <INPUT type="text" name="name" size="30"><br>
Your email: <INPUT type="text" name="email" size="30"><br>
<INPUT type="submit" name="submit-button" value="Subscribe">
```

- Action (a URL)

```
http://somedomain.com/subscribe
```

- Parameters:

```
name
email
```

Parameter Encoding

- urllib.urlencode()
- Takes a dictionary of fields and creates a URL-encoded string of parameters

```
fields = {
   'name' : 'Dave',
   'email' : 'dave@dabeaz.com'
}

parms = urllib.urlencode(fields)
```

- Sample result

```
>>> parms
'name=Dave&email=dave%40dabeaz.com'
>>>
```

Sending Parameters

- Case 1 : GET Requests

```
<FORM ACTION="/subscribe" METHOD="GET">
Your name: <INPUT type="text" name="name" size="30"><br>
Your email: <INPUT type="text" name="email" size="30"><br>
<INPUT type="submit" name="submit-button" value="Subscribe">
```

- Example code:

```
fields = { ... }
parms = urllib.urlencode(fields)
u = urllib.urlopen("http://somedomain.com/subscribe?"+parms)
```

You create a long URL by concatenating the request with the parameters

```
http://somedomain.com/subscribe?name=Dave&email=dave%40dabeaz.com
```

Sending Parameters

- Case 2 : POST Requests

```
<FORM ACTION="/subscribe" METHOD="POST">
Your name: <INPUT type="text" name="name" size="30"><br>
Your email: <INPUT type="text" name="email" size="30"><br>
<INPUT type="submit" name="submit-button" value="Subscribe">
```

- Example code:

```
fields = { ... }
parms = urllib.urlencode(fields)
u = urllib.urlopen("http://somedomain.com/subscribe", parms)
```

Parameters get uploaded separately
as part of the request body

```
POST /subscribe HTTP/1.0
...
name=Dave&email=dave%40dabeaz.com
```

Web Services

- The technique used for forms also applies more generally to various web services
    - Example : Maps, stock quotes, etc.
- Example : REST-based APIs
- Can issue GET/POST requests with encoded request parameters

urllib Limitations

- Does not support cookies
- Does not support authentication
- Does not report HTTP errors gracefully
- Only supports GET/POST requests

urllib2 Module

- urllib2 - The sequel to urllib
- Builds upon and expands urllib
- Can interact with servers that require cookies, passwords, and other details
- Is the preferred library for modern code

urllib2 Example

- urllib2 provides urlopen() as before

```
>>> import urllib2
>>> u = urllib2.urlopen("http://www.python.org/index.html")
>>> data = u.read()
>>>
```

- However, the module expands functionality in two primary areas
  - Requests
  - Openers

urllib2 Requests

- Requests are now objects

```
>>> r = urllib2.Request("http://www.python.org")
>>> u = urllib2.urlopen(r)
>>> data = u.read()
```

- Requests can have additional attributes added
- User data (for POST requests)
- Customized HTTP headers

Requests with Data

- Create a POST request with user data

```
data = {
   'name' : 'dave',
   'email' : 'dave@dabeaz.com'
}

r = urllib2.Request("http://somedomain.com/subscribe",
                    urllib.urlencode(data))
u = urllib2.urlopen(r)
response = u.read()
```

- Note : You still use urllib.urlencode() from the original urllib library

Request Headers

- Adding/Modifying client HTTP headers

```
headers = {
   'User-Agent' : 'Mozilla/4.0 (compatible; MSIE 7.0;
Windows NT 5.1; .NET CLR 2.0.50727)'
}

r = urllib2.Request("http://somedomain.com/",
                    headers=headers)
u = urllib2.urlopen(r)
response = u.read()
```

- This can be used if you need to emulate a specific client (e.g., Internet Explorer, etc.)

urllib2 Openers

- The function urlopen() is an "opener"
- It knows how to open a connection, interact with the server, and return a response.
- It only has a few basic features---it does not know how to deal with cookies and passwords
- However, you can make your own opener objects with these features enabled

urllib2 build_opener()

- build_opener() makes an custom opener

```
# Make a URL opener with cookie support
opener = urllib2.build_opener(
            urllib2.HTTPCookieProcessor()
)
u = opener.open("http://www.python.org/index.html")
```

- Can add a set of new features from this list

```
CacheFTPHandler
HTTPBasicAuthHandler
HTTPCookieProcessor
HTTPDigestAuthHandler
ProxyHandler
ProxyBasicAuthHandler
ProxyDigestAuthHandler
```

Example : Login Cookies

```
fields = {
    'txtUsername' : 'dave',
    'txtPassword' : '12345',
    'submit_login' : 'Log In'
}
opener = urllib2.build_opener(
            urllib2.HTTPCookieProcessor()
         )
request = urllib2.Request(
      "http://somedomain.com/login.asp",
      urllib.urlencode(fields))

# Login
u = opener.open(request)
resp = u.read()

# Get a page, but use cookies returned by initial login
u = opener.open("http://somedomain.com/private.asp")
resp = u.read()
```

Commentary

- Expanding urllib2 with new "openers" is one technique used for getting Python to interact with more advanced HTTP-based services
- Examples
  - Adding support for uploads
  - Presenting SSL client certificates

SSL

- Python provides low-level support for SSL (Secure Sockets Layer)
- Built on OpenSSL library
- Provided by ssl module
- http://docs.python.org/library/ssl

SSL Client Connection

```
import socket
import ssl

KEYFILE = "clientkey.pem"   # Client's private key
CERTFILE = "clientcert.crt" # Client's certificate
CA_CERTS = "ca.crt"         # CA certificate

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((hostname,port))
ssl_s = ssl.wrap_socket(s,
                        keyfile=KEYFILE,
                        certfile=CERTFILE,
                        cert_reqs=ssl.CERT_REQUIRED,
                        ca_certs=CA_CERTS)

# Use ssl_s as a socket
```

SSL Server Connection

```
import socket,ssl

KEYFILE = "servkey.pem"    # Server's private key
CERTFILE = "servcert.crt"  # Server's certificate
CA_CERTS = "ca.crt"        # CA certificate

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((hostname,port))
s.listen(1)
while True:
   client,addr = s.accept()
   try:
       client_ssl = ssl.wrap_socket(client,
                                    keyfile=KEYFILE,
                                    certfile=CERTFILE,
                                    server_side=True,
                                    cert_reqs=ssl.CERT_REQUIRED,
                                    ca_certs=CA_CERTS)
       # Use client_ssl as a socket
   except Exception as e:
       print "Failed: %s" % e
```

SSL Setup

- Biggest challenge for SSL is the preliminary setup of keys, certificates, and certificate authorities
- Not Python related, but will illustrate

Public/Private Keys

- Connection endpoints in SSL are identified by a public/private key-pair (RSA)
- Can generate with ssh-keygen

```
bash % ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/
beazley/.ssh/id_rsa): mykey_rsa
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in mykey_rsa.
Your public key has been saved in mykey_rsa.pub.
bash %
```

Example Key-File (PEM)


```
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEA0g/qMm2ttFHxhrMErChqPrHt/fbXaYtMZy+QLqK98AG87F6D
nOJnezAzKAHLG1jmRk5geOjiPMsiurudYQYIjS9gm/Q9aTrwsBmoh7v2juUwdYV6
y3ljHRrJW5nzH3stiRJTzWSHtFumog0pJ1T7NxgImHf3XQ5YQM2lMgCL8ZVSg48K
uANJ/gq0sZtZxVvb0bsYKOh8eTTKjrCH8o7jJeNadAe2XDe4LAjbwcpBpVrP+3aP
PvdBVJODSkLeQTZ+SPagISsBvVF/FibMB/S9uZ2gws25VWdLID7lKHz5dTVD4Zuf
0oT8L1FlbKcS2ntYpLxgMGm01OG7cLnc1+G2HwIBIwKCAQAkArpvC3zdFVyav5pm
p9ey+e5Iyzre5K4DD3fNevv9QiBjCONcuRkNzb+gdVYEsCd5xEsNacBiMXOq+5dS
dhAYNAHu2WmdAsos0TLVcK3snFFzO3Qi4Zv2XF0IY4jDkXWFNleCS79+ARVAWgcO
DpF3KLEEMdKiPPkvrjmTixCtEfkZPs+6GQW7hDCAUJC24XmTNu53mgP/RVpmtZec
HWefhDCqpCnwRFUDklgzMBJTf1UomA4v7Jev0QXjIj1qs0I+6UB+5mxnFNUA796o
CD0cnrmcWa1XTpfouNwCPI00cQIULgfhWf8swnLo63E3l9rPx5hHhVhzIq1tQaFE
XyTLAoGBAPCRj4KjsEgWt9fBlnMp+76CWYje5vTPN2i+S9HeNEuPvXUoSjE3u9kR
xfw4XkE0vh/avyWjNWdQ6NYlFt3bwgMIZWE2oVWQ6ZcG0dYm0QFY2n5M9YcnqlOJ
jYniUJnyJyPD2gadwLCWsNZLbj7JI+nkrC1QK2ABDSi9WHuuWltPAoGBAN+JZncB
a7XMJRdY2Zpwki10oSA6/xsDk30j7BcFrxBzosloVTsq7USFbTJe05WJiz9elcGE
Mw/XZ2Ai5ETMXs4m60aw9DPTGQ2J3bhiSQKPJT0Qd78Fx9bJJa05IH0meulStRXn
jKW+MkYn22pNDEeuro/ptHuJikm2DTRwBwQxAoGAFJ7DKHRuMhCTakPLES97+mLx
u0ZOT395xyZBA1wwXj+FRI5swmPc5rhhbWPq0mObRI8XssTYsRWQTN6bj1v6r82F
CFUjxYFzG5LeyTaG8XylA4LweUyK8TetC9GR4U9FLvOHt2ybfNm3YtND9sDIkGQO
wg4vmoO/TKKD7VgWX5kCgYEA0sNgnBdITFLD4tAdoD5AroPoYDegEifxdf1MUDiP
HiPioKQzGoePQJsPL336sZBQFy1LXq/YX2Sx7O2yp0RZY0lEO1Zil0Ngw58+w8pi
GFsUe2dMVQVzRtrpAmkQAPhlQmPskP7jsjb8M4SqTkilLaSzNztvp62RA6umDN6n
h5sCgYEAx5gFJK5Kk2HP5Npz9h6zEPhEtrOxl+fm/0du0qyZ/vs0AHzDla77z9e1
BFky5r6EQH+pTi4PeQWal/bffVSDNBBb8ilBeFIKVxNaMiCnlQvIbmWJMy3Q/vZS
vXN6S4PRwhJmUUPe7yD+nXCN0wdsms9kgewID/Czx5O3pOogWz0=
-----END RSA PRIVATE KEY-----
```

Certificates

- To authenticate, keys must be signed by a trusted certificate authority (e.g., verisign, equifax, etc.)
- You first make a certificate request

```
bash % openssl req -new -key mykey_rsa -out mycert.csr
```

- Submit to a certificate authority (with $$)
- Will eventually receive a signed certificate

Example Certificate (PEM)


```
-----BEGIN CERTIFICATE-----
MIICtTCCAh4CAQAwDQYJKoZIhvcNAQEEBQAwRzELMAkGA1UEBhMCVVMxETAPBgNV
BAgTCElsbGlub2lzMRAwDgYDVQQHEwdDaGljYWdvMRMwEQYDVQQKEwpEYWJlYXog
TExDMB4XDTExMDUyMjE0MjYyMloXDTEyMDUyMTE0MjYyMlowfTELMAkGA1UEBhMC
VVMxETAPBgNVBAgTCElsbGlub2lzMRAwDgYDVQQHEwdDaGljYWdvMQ0wCwYDVQQK
EwRBQ01FMQ0wCwYDVQQLEwRBQ01FMQ0wCwYDVQQDEwRBQ01FMRwwGgYJKoZIhvcN
AQkBFg1hY21lQGFjbWUuY29tMIIBIDANBgkqhkiG9w0BAQEFAAOCAQ0AMIIBCAKC
AQEA0g/qMm2ttFHxhrMErChqPrHt/fbXaYtMZy+QLqK98AG87F6DnOJnezAzKAHL
G1jmRk5geOjiPMsiurudYQYIjS9gm/Q9aTrwsBmoh7v2juUwdYV6y3ljHRrJW5nz
H3stiRJTzWSHtFumog0pJ1T7NxgImHf3XQ5YQM2lMgCL8ZVSg48KuANJ/gq0sZtZ
xVvb0bsYKOh8eTTKjrCH8o7jJeNadAe2XDe4LAjbwcpBpVrP+3aPPvdBVJODSkLe
QTZ+SPagISsBvVF/FibMB/S9uZ2gws25VWdLID7lKHz5dTVD4Zuf0oT8L1FlbKcS
2ntYpLxgMGm01OG7cLnc1+G2HwIBIzANBgkqhkiG9w0BAQQFAAOBgQCRUIcIEbRA
+cXCajmPZP2rr50q9zdG7YjoeEZF3B9X3RSLDrGfi681rkrm4WiNye7uMdVedPYP
SigRMUr76z1SIwgGOs5ucQnM14EQHzBsVgHr+OPqRo++qIHptnTMgxZqVB6FRrRC
6zm0g/GLpnIaC043dJUaX1y0LdPydh2XAg==
-----END CERTIFICATE-----
```

Keys and Certificates

- Both the private-key and certificate files are supplied when setting up the socket

```
KEYFILE = "mykey_rsa.pem"
CERTFILE = "mycert.crt"
ssl_s = ssl.wrap_socket(s,
                        keyfile=KEYFILE,
                        certfile=CERTFILE,
                        ...)
```

- Note: private key is not transmitted
- Note: may not be necessary if the other endpoint doesn't request a certificate

Certificate Authorities

- To authenticate certificates, you need the certificate of the certificate authority
- Can be downloaded from various CAs
- Usually included in web browsers
- You must provide it if you want validation

Example CA Cert File

```
GTE CyberTrust Global Root
==========================
-----BEGIN CERTIFICATE-----
MIICWjCCAcMCAgGlMA0GCSqGSIb3DQEBBAUAMHUxCzAJBgNVBAYTAlVTMRgwFgYDVQQKEw9HV
Q29ycG9yYXRpb24xJzAlBgNVBAsTHkdURSBDeWJlclRydXN0IFNvbHV0aW9ucywgSW5jLjEjM
...
NMQkpw0PlZPvy5TYnh+dXIVtx6quTx8itc2VrbqnzPmrC3p/
-----END CERTIFICATE-----

Digital Signature Trust Co. Global CA 1
=======================================
-----BEGIN CERTIFICATE-----
MIIDKTCCApKgAwIBAgIENnAVljANBgkqhkiG9w0BAQUFADBGMQswCQYDVQQGEwJVUzEkMCIGA
ChMbRGlnaXRhbCBTaWduYXR1cmUgVHJ1c3QgQ28uMREwDwYDVQQLEwhEU1RDQSBFMTAeFw05C
...
RbyhkwS7hp86W0N6w4pl
-----END CERTIFICATE-----

... continues ...
```

Certificate Validation

- Example of certificate validation

```
CA_CERT = "ca_certs.pem"
...
ssl_s = ssl.wrap_socket(s,
                        keyfile=KEYFILE,
                        certfile=CERTFILE,
                        cert_reqs=ssl.CERT_REQUIRED,
                        ca_certs=CA_CERT,
                        ...
)
```

- Generates an error if the other other end of the connection provides a certificate that can't be validated by entries in the ca_certs file

SSL Commentary

- SSL support in Python is extremely low-level
- Operates at the level of sockets
- Only partially supported elsewhere (e.g., urllib, servers, etc.)
- To incorporate it elsewhere, you subclass

Example : SSL-XMLRPC

```
class SSLSimpleXMLRPCServer(SimpleXMLRPCServer):
    def get_request(self):
        client, addr = SimpleXMLRPCServer.get_request(self)
            # Wrap the client in an SSL layer
            client_ssl = ssl.wrap_socket(client,
                                         keyfile=KEYFILE,
                                         certfile=CERTFILE,
                                         server_side=True,
                                         cert_reqs=ssl.CERT_REQUIRED,
                                         ca_certs=CA_CERTS)
            return client_ssl, addr

serv = SSLSimpleXMLRPCServer(("",15000))
serv.register_function(foo)
serv.serve_forever()
```

Example : SSL-urllib

```
import urllib2, httplib
class HTTPSClientAuthHandler(urllib2.HTTPSHandler):
    def __init__(self, key, cert):
        urllib2.HTTPSHandler.__init__(self)
        self.key = key
        self.cert = cert

    def https_open(self, req):
        return self.do_open(self.getConnection, req)

    def getConnection(self, host, timeout=300):
        return httplib.HTTPSConnection(host,
                                       key_file=self.key,
                                       cert_file=self.cert)

opener = urllib2.build_opener(
                      HTTPSClientAuthHandler(KEYFILE, CERTFILE))
response = opener.open("https://somehost.com/index.html")
print response.read()
```

Other Crypto Features

- No real cryptographic support built-in
- There are some libraries for hashing however
- hashlib, hmac

hashlib module

- Support for MD5, SHA hashing
- Example:

```
>>> import hashlib
>>> digest = hashlib.sha256()
>>> digest.update("some data")
>>> digest.update("more data")
>>> digest.hexdigest()
'dc21329ae1173d5d13046561fa7c7c60ed598a6d666e29c7c61bbbd4b8c7
>>>
```

More Information

- There are third-party add-ons that do more
- pycrypto (Cryptography)
- pyopenssl (more OpenSSL support)

Example Code

- See "PythonClass/Solutions/ssl"