432 Information Security
Homework 1

Ed Felten

September 15, 2008

Contact Bill Zeller with comments or questions at wzeller@cs.princeton.edu

1 Setup

Requirements:

- Python (2.5 or above)
- Note: Python libraries can be built by first typing:

  ```
  python setup.py build (to build the library)
  python setup.py install (to install the library)
  ```

  (You may need to prepend `sudo` to these commands if you're on a Linux box.)

- **Building on Linux**
  - Ensure you have Python 2.5 installed
  - Note: you may need the package "Python2.5-dev" and "build-essentials"
  - Twisted (tested with 8.1.0)
    - Note: The apt-get package "python-twisted" does not appear to work. You will have to manually install these on Linux
    - `http://twistedmatrix.com/trac/wiki/Downloads`
    - Download the source code if you're on Ubuntu (not the Ubuntu version!)
    - Build and install using above instructions
  - Zope.Interface (required by Twisted)
    - Download source code from: `http://zope.org/Products/ZopeInterface/`
    - Build and install using above instructions
  - Install PyCrypto
    - Download source code from: `http://www.amk.ca/python/code/crypto`

        - Build and install using above instructions

- **Building on Mac OS X (10.5.4 tested)**
    - PyCrypto
        - Download source code from: `http://www.amk.ca/files/python/crypto/pycrypto-2.0.1.tar.gz`
        - Build and install using above instructions
    - Zope for Mac OS X:
        - Download source code from: `http://www.zope.org/Products/ZopeInterface/3.3.0/zope.interface-3.3.0.tar.gz`
        - Build and install using above instructions
    - Twisted (SOURCE package)
        - Download source code from: `http://tmrc.mit.edu/mirror/twisted/Twisted/8.1/Twisted-8.1.0.tar.bz2`
        - Build and install using above instructions
- **Running on Princeton's CS Machines**
    - SSH into Cycles: `ssh cycles.cs.princeton.edu`
    - Download prepackaged libraries we've created: `wget http://www.cs.princeton.edu/~wzeller/432/python_libs.zip`
    - Unzip: `unzip python_libs.zip`
    - Add this directory to your path:
      `export PYTHONPATH=${PYTHONPATH}:[python_lib_directory]`
      For example, on my machine I downloaded python_libs.zip to /u/wzeller/432/test, so I ran:
      `export PYTHONPATH=${PYTHONPATH}:/u/wzeller/432/test/python_libs`
    - You will need to run the `export` command every time you log in. If you'd rather have this command run automatically, add it to your /.profile or /.bashrc files.

2 Code Structure

- PythonSSH
    - Client (client specific code)
        - MyClient.py (fully functional SSH client. You can run this directly)
    - Server (server specific code)
        - MyServer.py (fully functional SSH server. You can run this directly)
        - SSHShell.py

    - **MySSHShell.py** (You will *only* be submitting this file)
  - Fun (random code you can use when creating your shell)
  - SSHUtil.py (Utility functions that facilitate conversion between Python data types and SSH data types)

Note: You will *only* be submitting **MySSHShell.py**. You may edit other files, but we won't see or use those changes.

3 The Assignment

The goal of the first assignment is to familiarize yourself with Python and our PythonSSH code. This assignment should not be difficult if you know Python.

3.1 What To Do

1. Make sure the client and server can talk to each other.
   - Execute the command

     ```
     python MyServer.py 2222
     ```

     (you can use another port if this doesn't work).
     This should initialize the server
   - Execute the command

     ```
     python MyClient.py -p2222 username@localhost
     ```

     (python MyClient.py will give you a list of options– "`--log`" is especially useful)
   - If both of those were successful, you should be able to log in with your username and password ("username" and "password" – this can be changed in MyServer.py)
2. Complete a series of simple programming exercises to familiarize yourself with Python.
   - These exercises will be implemented as part of building your own SSH shell. They will be implemented in MySSHShell.py
   - Each function starting with `do_` magically becomes available to the shell. Python docstrings can be used to supply help text.
     - For example, typing `clear` in the shell calls `do_clear()`. Typing `help clear` prints out `Clears the screen.  Usage:  clear`, because this is the docstring of the `do_clear()` function
     - Hint: See `http://diveintopython.org/getting_to_know_python/documenting_functions.html` for more on docstrings

- A function can accept multiple arguments, or multiple arguments, automatically. All arguments are passed as strings.
    - Hint: The Python `int()` and `str()` functions can be used to convert strings to ints and ints to strings respectively.
- Write to the terminal using `self.write(s)`
- Write a new line with `self.nextLine()`

3. Functions to Implement

- *Note: "print" below means to call `self.write(s)`. Responses should be written to the shell and you should not call Python's `print` command, although this can be used for debugging.*
- `do_mult(self, x, y)`
  Print out x * y (where x and y are integers)
  Hint: Try large numbers
  Example: `$ mult 32 34234 => 1095488`
- `do_fact(self, n)`
  Print out the factorial of n
  Example: `$ fact 5 => 120`
- `do_loop(self, n)`
  Loops from 1 to n (n is positive) and prints out the result, one per line
  Example: `$ loop 5`
  ```
  1
  2
  3
  4
  5
  ```

- `do_max(self, *args)`
  Print the max of a list of integers by looping over the arguments. Usage: max n1 n2 n3 n4 ...
  If `max` is not given any arguments, this behavior is undefined.
  Example: `$ max 3 2 324 32 => 324`
    - Hint: args is an array and can be iterated over (`for i in args:` ...)
- `do_max_lambda(self, *args)`
  Print the max of a list of integers using Python's `map()` and `reduce()` and lambda functions
  If `max_lambda` is not given any arguments, this behavior is undefined.
  Example: `$ max 3 2 324 32 => 324`
    - Hint: First map the list onto integers, then reduce to find the max
    - Hint: Python has allows for the syntax `a if x else b` that returns a if x is True and returns b if x is False that might be useful in lambda functions

  - Hint: More on map/reduce: `http://docs.python.org/lib/built-in-funcs.html`
  - Hint: More on lambda functions: `http://diveintopython.org/power_of_introspection/lambda_functions.html`

- `do_sum(self, *args)`
  Print the sum of a list of integers by looping over the arguments
  If `sum` is not given any arguments, you should output 0
  Example: `$ sum 3 85 37 => 125`

- `do_sum_lambda(self, *args)`
  Print the sum of a list of integers using `map()` and `reduce()`
  If `sum_lambda` is not given any arguments, the program should output 0
  Example: `$ sum 3 85 37 => 125`

- `do_sort(self, *args)`
  Sort a list of integers and print out the result, using Python's builtin `sort()` or `sorted()` function
  If `sort` is not given any arguments, the program should output an empty new line.
  Example: `$ sort 34 21 411 5 21 12 => 5 12 21 21 34 411`
  - Hint: `sort()` sorts in place, while `sorted()` does not.
  - Hint: More on sorting in Python: `http://wiki.python.org/moin/HowTo/Sorting`

- `do_myhelp(self)`
  Simply call the superclass's help function

- `do_rss(self)`
  Retreive an RSS feed and print out the titles. Use any RSS feed you can find on the Interwebs.
  - Hint: Import the feedparser library with:
    `from PythonSSH.Fun.RSS import feedparser`
  - Hint: Titles are (most likely) in UTF-8. To print them out, call
    `item.title.encode('ascii','replace')`
  - Hint: See `http://www.feedparser.org/` for info on the feedparser API
  - Hint: The key "entries" in the object returned from `feedparser.parse()` is an array of entries, each of which have a title attribute
  - Hint: Some NYTimes RSS feeds `http://www.nytimes.com/services/xml/rss/index.html`
  - Hint: Most browsers display an RSS icon on the top right of the location bar that will show the site's RSS feed, if it exists

- **Dictionary Functions** Implement functions to manage a dictionary
  - Hint: Store the dictionary as a class variable in `__init__`

    - Hint: booleans can also be coerced to strings

- `dict_put(self, key, value)`
  Store key in dictionary
- `dict_get(self, key)`
  Print out key if it exists in dictionary
- `dict_test(self, key)`
  Print out True if key exists, False if it doesn't
- `dict_list(self)`
  List all key, value pairs in dictionary