TTM4536 - Ethical Hacking - Information Security, Specialization Course

- Plan for 12 Sep 2017
  1. Analysis of the CTF solution from last week
  2. Completing exercises with bhnet.py,
  3. Completing exercises proxy.py and
  4. Completing exercises SSH with Paramiko from Chapter 2

Analysis of the CTF solution from last week

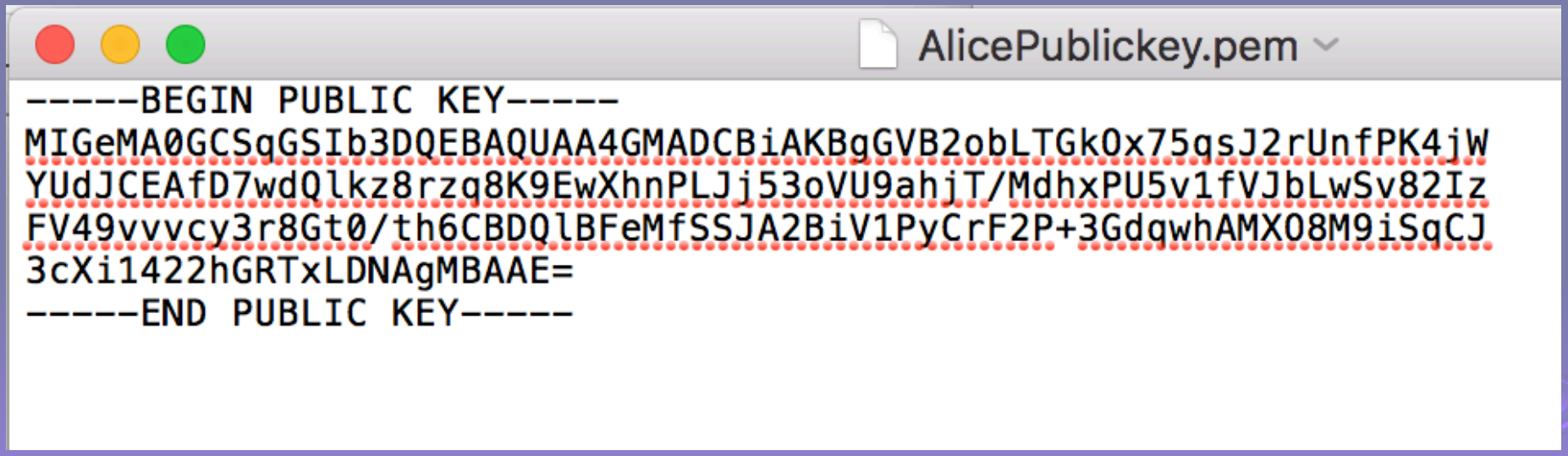
- The public keys of Alice and Bob are given in pem files.
- Bob sent the encrypted message Ciphertext.txt to Alice.
- Can you figure out the secret flag hidden in the Bob's message?
- Hint: Both Alice and Bob are from Taiwan where people prefer certain numbers more than others.

Analysis of the CTF solution from last week

- The public keys of Alice and Bob are given in pem files.

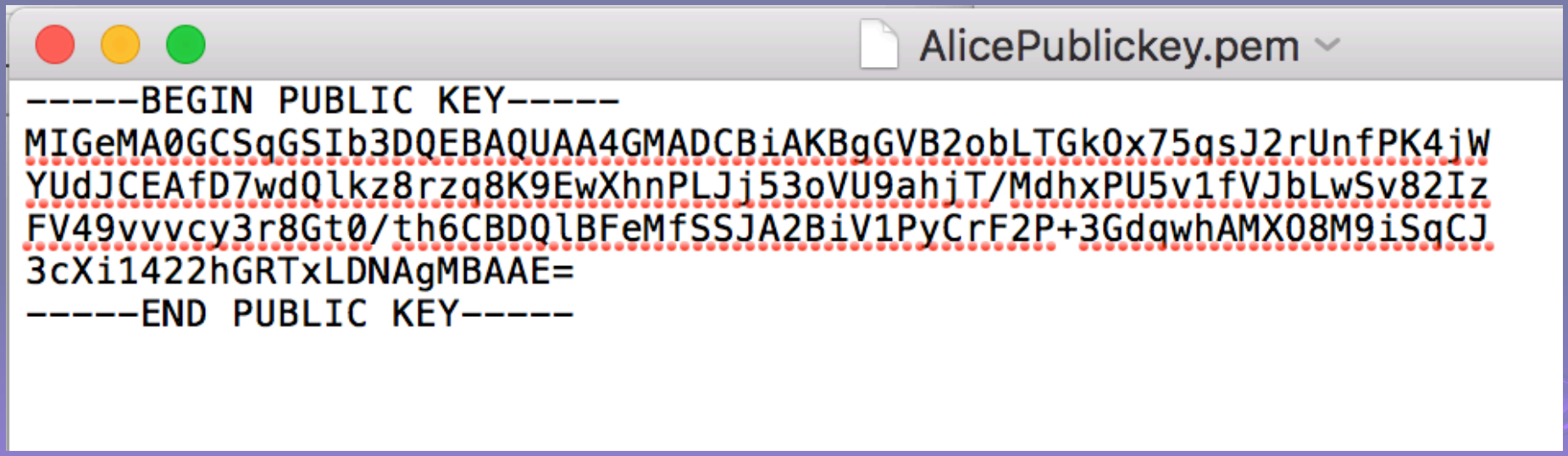
Analysis of the CTF solution from last week

- The public keys of Alice and Bob are given in pem files.



Analysis of the CTF solution from last week

- The public keys of Alice and Bob are given in pem files.



- Search the net and learn about the format of pem files.
- Search the net and learn how to work with pem files in python

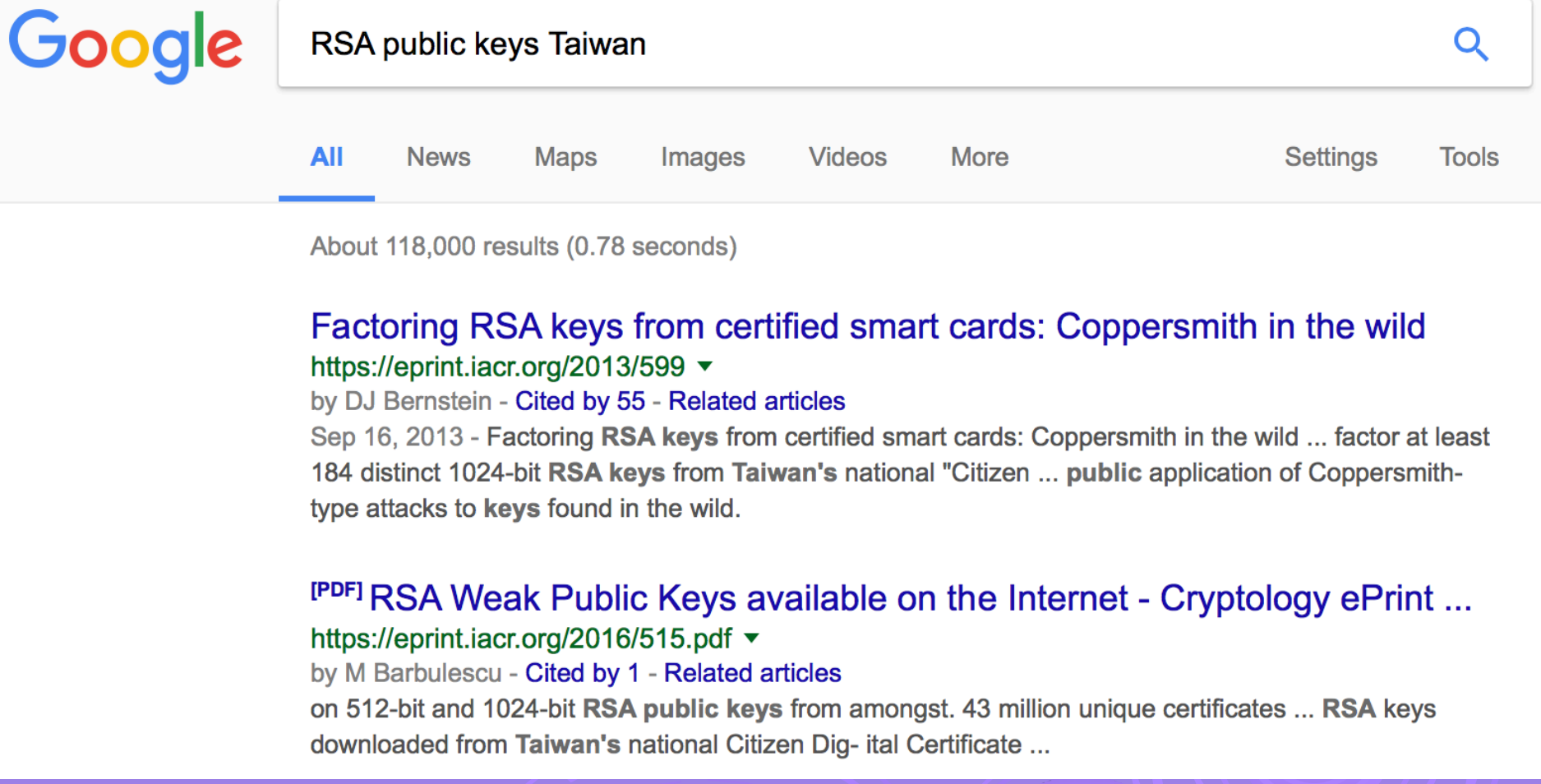
The meaning of the hint

- Hint: Both Alice and Bob are from Taiwan where people prefer certain numbers more than others.
- ??!??!?!

The meaning of the hint

- Hint: Both Alice and Bob are from Taiwan where people prefer certain numbers more than others.
- ??!??!?!
- Search the net for example with:
  - RSA public keys Taiwan

The meaning of the hint



Factoring RSA keys?

Key generation [edit]

The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers p and q.
   - For security purposes, the integers p and q should be chosen at random, and should be similar in magnitude but 'differ in length by a few digits'[2] to make factoring harder. Prime integers can be efficiently found using a primality test.
2. Compute $n = pq$.
   - n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
3. Compute $\lambda(n) = \text{lcm}(\lambda(p), \lambda(q)) = \text{lcm}(p-1, q-1)$, where λ is Carmichael's totient function. This value is kept private.
4. Choose an integer e such that $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; i.e., e and $\lambda(n)$ are coprime.
5. Determine d as $d \equiv e^{-1} \pmod{\lambda(n)}$; i.e., d is the modular multiplicative inverse of e (modulo $\lambda(n)$).
   - This is more clearly stated as: solve for d given $d \cdot e \equiv 1 \pmod{\lambda(n)}$.
   - e having a short bit-length and small Hamming weight results in more efficient encryption – most commonly $e = 2^{16} + 1 = 65{,}537$. However, much smaller values of e (such as 3) have been shown to be less secure in some settings.[14]
   - e is released as the public key exponent.
   - d is kept as the private key exponent.

Factoring RSA keys?

- Alice's public key:
  - N_a = p_a q_a
- Bob's public key:
  - N_b = p_b q_b
- Public exponent is usually:
  - e = 2^16 + 1 = 65537
- Prime numbers p_a, q_a, p_b, q_b have to be generated uniformly at random
- BUT …?

Factoring RSA keys?

- BUT …?
- If Alice and Bob have some common prime numbers p_a = p_b, then it is easy to factor N_a and N_b by frst computing
  - p_a = p_b = GCD(N_a, N_b)

Python script for complete solving of the CTF challenge

- Start python from the command line
- >>> from Crypto.PublicKey import RSA
- >>> f = open('AlicePublickey.pem','r')
- >>> AlicePublicKey = RSA.importKey(f.read())
- >>> f.close()
- >>> f = open('BobPublickey.pem','r')
- >>> BobPublicKey = RSA.importKey(f.read())
- >>> f.close()
- >>> print AlicePublicKey.n
- >>> print BobPublicKey.n

- COMPUTE GCD on AlicePublicKey.n and BobPublicKey.n
- …

Completing exercises with bhnet.py, proxy.py and SSH with Paramiko from Chapter 2

- You should have so far found the sources for the textbook Black Hat Python. If not, search for "Black Hat Python sources", go to the website of No Starch Press and "**Download the resources and code samples from the book**"
- Uncompress the folder BHP-Code
- Use the files from the folder Chapter 2

Completing exercises with bhnet.py, proxy.py and SSH with Paramiko from Chapter 2

- In PyCharm open a new project Ch02
- Add an empty python file bhnet.py (delete the author command in the beginning of the file)
- Open the file bhnet.py from Chapter 2 and copy-paste its content to the file bhnet.py of the Ch02 project

Completing exercises with bhnet.py, proxy.py and SSH with Paramiko from Chapter 2

- For our installed Kali machines you should change the first line from:

```
#!/opt/local/bin/python2.7
```

- to

```
#!/usr/bin/python2.7
```

- Save the file
- Open two terminals and go to the folder

```
/PycharmProjects/Ch02/
```

Completing exercises with bhnet.py, proxy.py and SSH with Paramiko from Chapter 2

- Make the bhnet.py executable with:

```
# chmod 755 bhnet.py
```

- Start bhnet.py first as a server to listen the port 9999

```
# ./bhnet.py -l -p 9999 -c
```

- Then in the other terminal start bhnet.py as a client

```
# ./bhnet.py -t localhost -p 9999
```

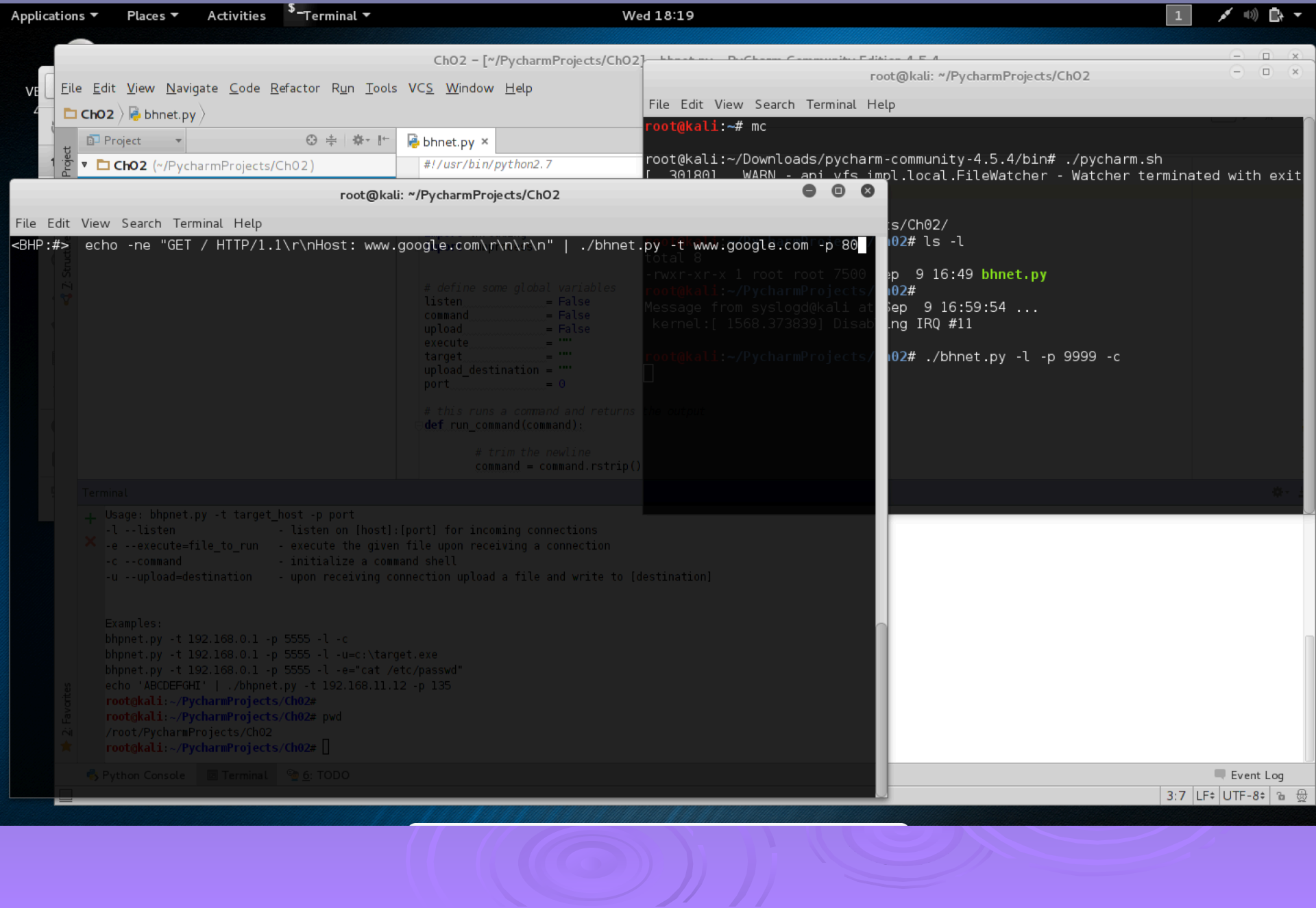
- Press Ctrl D in order to send EOF and the execution will enter an inside terminal with a prompt `<BHP:#>`

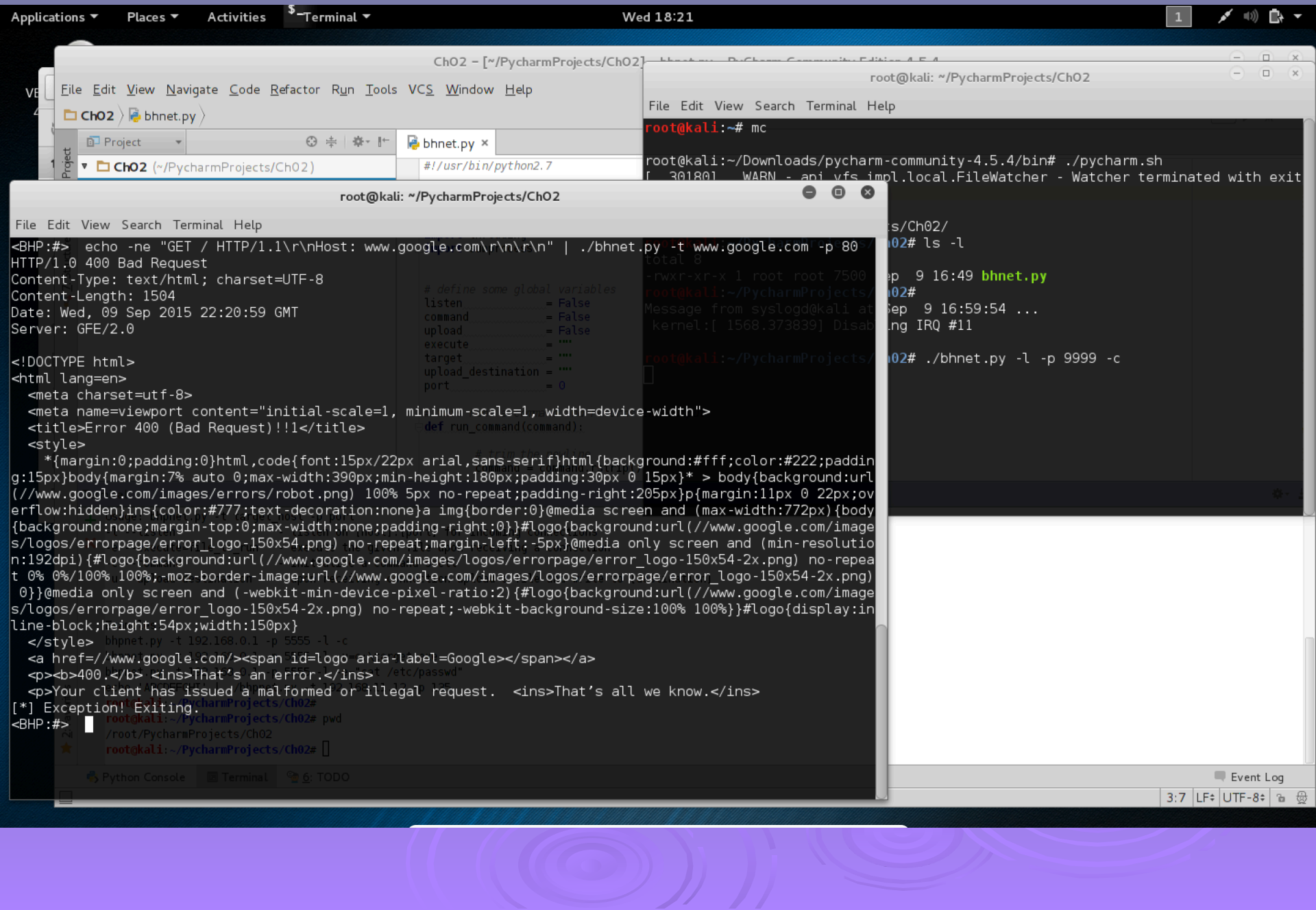
Completing exercises with bhnet.py, proxy.py and SSH with Paramiko from Chapter 2

- Type at <BHP:#> prompt

```
# echo -ne "GET / HTTP/1.1\r\nHost: www.google.com\r\n\r\n" | ./bhnet.py -t www.google.com -p 80
```

- Note that the response from google now is different than it is reported in the textbook


Applications
Places
Activities
Terminal
Wed 18:19
Ch02 – [~/PycharmProjects/Ch02]
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Ch02 bhnet.py
Project
bhnet.py
#!/usr/bin/python2.7
root@kali: ~/PycharmProjects/Ch02
File Edit View Search Terminal Help
root@kali:~# mc
root@kali:~/Downloads/pycharm-community-4.5.4/bin# ./pycharm.sh
root@kali: ~/PycharmProjects/Ch02
File Edit View Search Terminal Help
<BHP:#> echo -ne "GET / HTTP/1.1\r\nHost: www.google.com\r\n\r\n" | ./bhnet.py -t www.google.com -p 80
# define some global variables
listen = False
command = False
upload = False
execute = ""
target = ""
upload_destination = ""
port = 0
# this runs a command and returns the output
def run_command(command):
# trim the newline
command = command.rstrip()
Terminal
Usage: bhpnet.py -t target_host -p port
-l --listen - listen on [host]:[port] for incoming connections
-e --execute=file_to_run - execute the given file upon receiving a connection
-c --command - initialize a command shell
-u --upload=destination - upon receiving connection upload a file and write to [destination]
Examples:
bhpnet.py -t 192.168.0.1 -p 5555 -l -c
bhpnet.py -t 192.168.0.1 -p 5555 -l -u=c:\target.exe
bhpnet.py -t 192.168.0.1 -p 5555 -l -e="cat /etc/passwd"
echo 'ABCDEFGHI' | ./bhpnet.py -t 192.168.11.12 -p 135
root@kali:~/PycharmProjects/Ch02#
root@kali:~/PycharmProjects/Ch02# pwd
/root/PycharmProjects/Ch02
root@kali:~/PycharmProjects/Ch02#
Python Console
Terminal
6: TODO
Event Log
3:7
LF
UTF-8

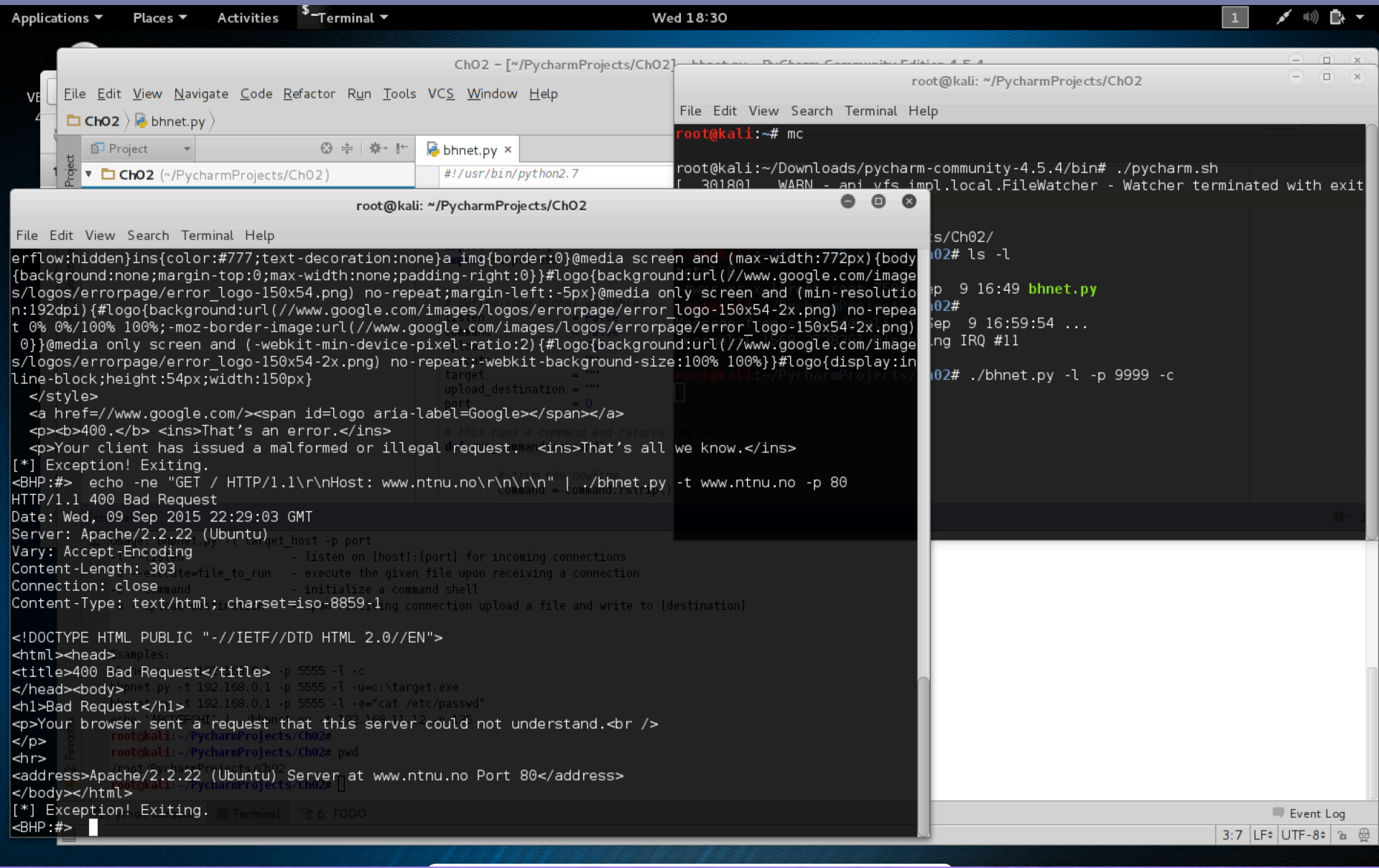

Applications
Places
Activities
Terminal
Wed 18:21
Ch02 – [~/PycharmProjects/Ch02]
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Ch02 bhnet.py
Project
bhnet.py
Ch02 (~/PycharmProjects/Ch02)
#!/usr/bin/python2.7
root@kali: ~/PycharmProjects/Ch02
File Edit View Search Terminal Help
root@kali:~# mc
root@kali:~/Downloads/pycharm-community-4.5.4/bin# ./pycharm.sh
root@kali: ~/PycharmProjects/Ch02
File Edit View Search Terminal Help
<BHP:#> echo -ne "GET / HTTP/1.1\r\nHost: www.google.com\r\n\r\n" | ./bhnet.py -t www.google.com -p 80
HTTP/1.0 400 Bad Request
Content-Type: text/html; charset=UTF-8
Content-Length: 1504
Date: Wed, 09 Sep 2015 22:20:59 GMT
Server: GFE/2.0
<!DOCTYPE html>
<html lang=en>
<meta charset=utf-8>
<meta name=viewport content="initial-scale=1, minimum-scale=1, width=device-width">
<title>Error 400 (Bad Request)!!1</title>
<style>
{margin:0;padding:0}html,code{font:15px/22px arial,sans-serif}html{background:#fff;color:#222;padding:15px}body{margin:7% auto 0;max-width:390px;min-height:180px;padding:30px 0 15px} > body{background:url(//www.google.com/images/errors/robot.png) 100% 5px no-repeat;padding-right:205px}p{margin:11px 0 22px;overflow:hidden}ins{color:#777;text-decoration:none}a img{border:0}@media screen and (max-width:772px){body{background:none;margin-top:0;max-width:none;padding-right:0}}#logo{background:url(//www.google.com/images/logos/errorpage/error_logo-150x54.png) no-repeat;margin-left:-5px}@media only screen and (min-resolution:192dpi){#logo{background:url(//www.google.com/images/logos/errorpage/error_logo-150x54-2x.png) no-repeat 0% 0%/100% 100%;-moz-border-image:url(//www.google.com/images/logos/errorpage/error_logo-150x54-2x.png) 0}}@media only screen and (-webkit-min-device-pixel-ratio:2){#logo{background:url(//www.google.com/images/logos/errorpage/error_logo-150x54-2x.png) no-repeat;-webkit-background-size:100% 100%}}#logo{display:inline-block;height:54px;width:150px}
</style>

<p>400. <ins>That’s an error.</ins>
<p>Your client has issued a malformed or illegal request. <ins>That’s all we know.</ins>
[*] Exception! Exiting.
<BHP:#>
root@kali:~/PycharmProjects/Ch02# ls -l
root@kali:~/PycharmProjects/Ch02# ./bhnet.py -l -p 9999 -c
root@kali:~/PycharmProjects/Ch02# pwd
/root/PycharmProjects/Ch02
Python Console
Terminal
6: TODO
Event Log
3:7
LF
UTF-8

Completing exercises with bhnet.py, proxy.py and SSH with Paramiko from Chapter 2

- See the response from www.ntnu.no server.
- Type at <BHP:#> prompt

```
# echo -ne "GET / HTTP/1.1\r\nHost: www.ntnu.no\r\n\r\n" | ./bhnet.py -t www.ntnu.no -p 80
```


Applications
Places
Activities
Terminal
Wed 18:30
Ch02 – [~/PycharmProjects/Ch02]
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Ch02 bhnet.py
Project
bhnet.py
Ch02 (~/PycharmProjects/Ch02)
#!/usr/bin/python2.7
root@kali: ~/PycharmProjects/Ch02
File Edit View Search Terminal Help
root@kali:~# mc
root@kali:~/Downloads/pycharm-community-4.5.4/bin# ./pycharm.sh
root@kali: ~/PycharmProjects/Ch02
File Edit View Search Terminal Help
erflow:hidden}ins{color:#777;text-decoration:none}a img{border:0}@media screen and (max-width:772px){body
{background:none;margin-top:0;max-width:none;padding-right:0}}#logo{background:url(//www.google.com/image
s/logos/errorpage/error_logo-150x54.png) no-repeat;margin-left:-5px}@media only screen and (min-resolutio
n:192dpi){#logo{background:url(//www.google.com/images/logos/errorpage/error_logo-150x54-2x.png) no-repea
t 0% 0%/100% 100%;-moz-border-image:url(//www.google.com/images/logos/errorpage/error_logo-150x54-2x.png)
0}}@media only screen and (-webkit-min-device-pixel-ratio:2){#logo{background:url(//www.google.com/image
s/logos/errorpage/error_logo-150x54-2x.png) no-repeat;-webkit-background-size:100% 100%}}#logo{display:in
line-block;height:54px;width:150px}
</style>

400. <ins>That's an error.</ins>
Your client has issued a malformed or illegal request. <ins>That's all we know.</ins>
[*] Exception! Exiting.
<BHP:#> echo -ne "GET / HTTP/1.1\r\nHost: www.ntnu.no\r\n\r\n" | ./bhnet.py -t www.ntnu.no -p 80
HTTP/1.1 400 Bad Request
Date: Wed, 09 Sep 2015 22:29:03 GMT
Server: Apache/2.2.22 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 303
Connection: close
Content-Type: text/html; charset=iso-8859-1
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
Your browser sent a request that this server could not understand.

<hr>
<address>Apache/2.2.22 (Ubuntu) Server at www.ntnu.no Port 80</address>
</body></html>
[*] Exception! Exiting.
<BHP:#>
root@kali:~/Downloads/pycharm-community-4.5.4/bin# ./pycharm.sh
[30180] WARN - api.vfs.impl.local.FileWatcher - Watcher terminated with exit
s/Ch02/
02# ls -l
ep 9 16:49 bhnet.py
02#
Sep 9 16:59:54 ...
ng IRQ #11
02# ./bhnet.py -l -p 9999 -c
Event Log
3:7 LF UTF-8

Completing exercises with bhnet.py, proxy.py and SSH with Paramiko from Chapter 2

- Actually `<BHP:#>` terminal is not good terminal and is not parsing the strings correctly
- Type at the command prompt of an ordinary terminal the same command

```
# echo -ne "GET / HTTP/1.1\r\nHost: www.ntnu.no\r\n\r\n" | ./bhnet.py -t www.ntnu.no -p 80
```

- You should receive the proper response

Spend some time to understand bhnet.py

- Study the used modules: sys, socket, getopt, threading, subprocess
- Study all the defined functions

http://www.python-course.eu/sys_module.php

SYS-MODUL

INFORMATION ON THE PYTHON INTERPRETER

Like all the other modules, the sys module has to be imported with the import statement, i.e.

```
import sys
```

If there are questions about the import statement, we recommend the introductory chapter of our basic course concerning this topic Modular Programming and Modules

The sys module provides information about constants, functions and methods of the Python interpreter. dir(system) gives a summary of the available constants, functions and methods. Another possibility is the help() function. Using help(sys) provides valuable detail information.

The module sys informs e.g. about the maximal recursion depth (`sys.getrecursionlimit()`) and provides the possibility to change (sys.setrecursionlimit())
The current version number of Python can be accessed as well:

```
>>> import sys
>>> sys.version
'2.6.5 (r265:79063, Apr 16 2010, 13:57:41) \n[GCC 4.4.3]'
>>> sys.version_info
(2, 6, 5, 'final', 0)
>>>
```

Sockets programming in Python

In languages with sockets, the socket is universally the same -- it's a conduit between the two applications that can communicate with one another.

Preliminaries

Whether you're writing a sockets application in Python, Perl, Ruby, Scheme, or any other useful language (and by *useful* I mean languages that have a sockets interface), the socket is universally the same. It's a conduit between the two applications that can communicate with one another (either locally on a single machine or between two machines in separate locations).

The difference with sockets programming in a language like Python is in the helper classes and methods that can simplify sockets programming. In this section I'll demonstrate the Python `socket` API. You can execute the Python interpreter with a script or, if you execute Python by itself, you can interact with it one line at a time. In this way, you can see the result of each method invoked.

The following example illustrates interacting with the Python interpreter. Here, I use the `socket` class method `gethostbyname` to resolve a fully qualified domain name (www.ibm.com) to a string quad-dotted IP address ('129.42.19.99'):

Listing 3. Using the socket API from the interpreter command line

```
[camus]$ <strong>python</strong>
Python 2.4 (#1, Feb 20 2005, 11:25:45)
[GCC 3.2.2 20030222 (Red Hat Linux 3.2.2-5)] on linux2
Type "help", "copyright", "credits" or "license" for more
   information.
>>> <strong>import socket</strong>
>>> <strong>socket.gethostbyname('www.ibm.com')</strong>
'129.42.19.99'
>>>
```

After the `socket` module is imported, I invoke the `gethostbyname` class method to resolve the domain name to an IP address.

Now, I'll discuss the basic `socket` methods and communicating through sockets. Feel free to follow along with your Python interpreter.

https://pymotw.com/2/getopt/

getopt – Command line option parsing

Purpose:	Command line option parsing
Available In:	1.4

The getopt module is the *old-school* command line option parser that supports the conventions established by the Unix function `getopt()`. It parses an argument sequence, such as `sys.argv` and returns a sequence of (option, argument) pairs and a sequence of non-option arguments.

Supported option syntax includes:

```
-a
-bval
-b val
--noarg
--witharg=val
--witharg val
```

Function Arguments

The getopt function takes three arguments:

- The first argument is the sequence of arguments to be parsed. This usually comes from `sys.argv[1:]` (ignoring the program name in `sys.arg[0]`).
- The second argument is the option definition string for single character options. If one of the options requires an argument, its letter is followed by a colon.
- The third argument, if used, should be a sequence of the long-style option names. Long style options can be more than a single character, such as `--noarg` or `--witharg`. The option names in the sequence should not include the `--` prefix. If any long option requires an argument, its name should have a suffix of `=`.

Short and long form options can be combined in a single call.

Short Form Options

If a program wants to take 2 options, `-a`, and `-b` with the b option requiring an argument, the value should be `"ab:"`.

```
import getopt

print getopt.getopt(['-a', '-bval', '-c', 'val'], 'ab:c:')
```

```
$ python getopt_short.py

([('-a', ''), ('-b', 'val'), ('-c', 'val')], [])
```

Long Form Options

If a program wants to take 2 options, `--noarg` and `--witharg` the sequence should be `[ 'noarg', 'witharg=' ]`.

```
import getopt

print getopt.getopt([ '--noarg', '--witharg', 'val', '--witharg2=another' ],
                    '',
                    [ 'noarg', 'witharg=', 'witharg2=' ])
```

```
$ python getopt_long.py

([('--noarg', ''), ('--witharg', 'val'), ('--witharg2', 'another')], [])
```

https://pymotw.com/2/threading/

threading – Manage concurrent threads

Purpose:	Builds on the **thread** module to more easily manage several threads of execution.
Available In:	1.5.2 and later

The **threading** module builds on the low-level features of **thread** to make working with threads even easier and more *pythonic*. Using threads allows a program to run multiple operations concurrently in the same process space.

Thread Objects

The simplest way to use a **Thread** is to instantiate it with a target function and call **start()** to let it begin working.

```
import threading

def worker():
    """thread worker function"""
    print 'Worker'
    return

threads = []
for i in range(5):
    t = threading.Thread(target=worker)
    threads.append(t)
    t.start()
```

The output is five lines with `"Worker"` on each:

```
$ python threading_simple.py

Worker
Worker
Worker
Worker
Worker
```

It useful to be able to spawn a thread and pass it arguments to tell it what work to do. This example passes a number, which the thread then prints.

```
import threading

def worker(num):
    """thread worker function"""
    print 'Worker: %s' % num
    return

threads = []
for i in range(5):
    t = threading.Thread(target=worker, args=(i,))
    threads.append(t)
    t.start()
```

The integer argument is now included in the message printed by each thread:

```
    $ python -u threading_simpleargs.py

Worker: 0
Worker: 1
Worker: 2
Worker: 3
Worker: 4
```

https://pymotw.com/2/subprocess/

If you find this information useful, consider picking up a copy of my book, *The Python Standard Library By Example*.

subprocess – Work with additional processes

Purpose:	Spawn and communicate with additional processes.
Available In:	2.4 and later

The `subprocess` module provides a consistent interface to creating and working with additional processes. It offers a higher-level interface than some of the other available modules, and is intended to replace functions such as `os.system()`, `os.spawn*()`, `os.popen*()`, `popen2.*()` and `commands.*()`. To make it easier to compare `subprocess` with those other modules, many of the examples here re-create the ones used for `os` and `popen`.

The `subprocess` module defines one class, `Popen` and a few wrapper functions that use that class. The constructor for `Popen` takes arguments to set up the new process so the parent can communicate with it via pipes. It provides all of the functionality of the other modules and functions it replaces, and more. The API is consistent for all uses, and many of the extra steps of overhead needed (such as closing extra file descriptors and ensuring the pipes are closed) are "built in" instead of being handled by the application code separately.

Note: The API is roughly the same, but the underlying implementation is slightly different between Unix and Windows. All of the examples shown here were tested on Mac OS X. Behavior on a non-Unix OS will vary.

Running External Command

To run an external command without interacting with it, such as one would do with *os.system()*, Use the `call()` function.

```
import subprocess

# Simple command
subprocess.call(['ls', '-1'], shell=True)
```

The command line arguments are passed as a list of strings, which avoids the need for escaping quotes or other special characters that might be interpreted by the shell.

```
$ python subprocess_os_system.py

__init__.py
index.rst
interaction.py
repeater.py
signal_child.py
signal_parent.py
subprocess_check_call.py
subprocess_check_output.py
subprocess_check_output_error.py
subprocess_check_output_error_trap_output.py
subprocess_os_system.py
subprocess_pipes.py
subprocess_popen2.py
subprocess_popen3.py
subprocess_popen4.py
subprocess_popen_read.py
subprocess_popen_write.py
subprocess_shell_variables.py
subprocess_signal_parent_shell.py
subprocess_signal_setsid.py
```

Spend some time to understand bhnet.py

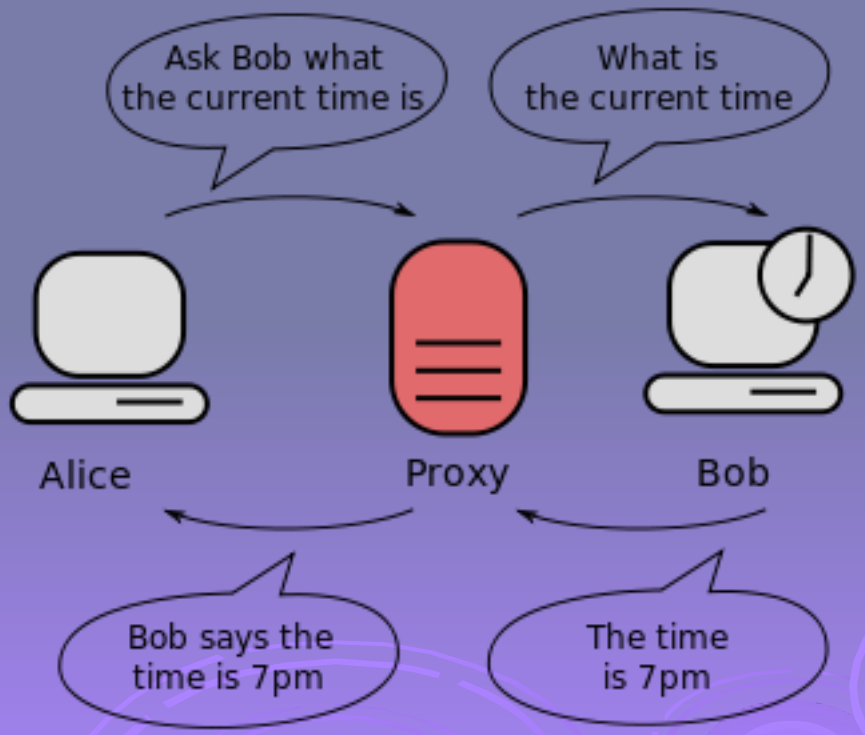
- At the end of testing – kill both the server and client terminal

Study the program proxy.py

- What is Proxy server?
- From Wikipedia:
- In computer networks, a proxy server is a server (a computer system or an application) that acts as an intermediary for requests from clients seeking resources from other servers.
- A client connects to the proxy server, requesting some service, such as a file, connection, web page, or other resource available from a different server and
- the proxy server evaluates the request as a way to simplify and control its complexity.
- Proxies were invented to add structure and encapsulation to distributed systems.
- Today, most proxies are web proxies, facilitating access to content on the World Wide Web and providing anonymity.

Study the program proxy.py

- What is Proxy server?
- From Wikipedia:



Study the program proxy.py

- Import proxy.py into the project Ch02 as you did with bhnet.py
- Put also as the first line the following
- `#!/usr/bin/python2.7`
- Open two terminals and go to the folder

  `/PycharmProjects/Ch02/`
- Make the proxy.py executable with:

  `# chmod 755 proxy.py`

Study the program proxy.py

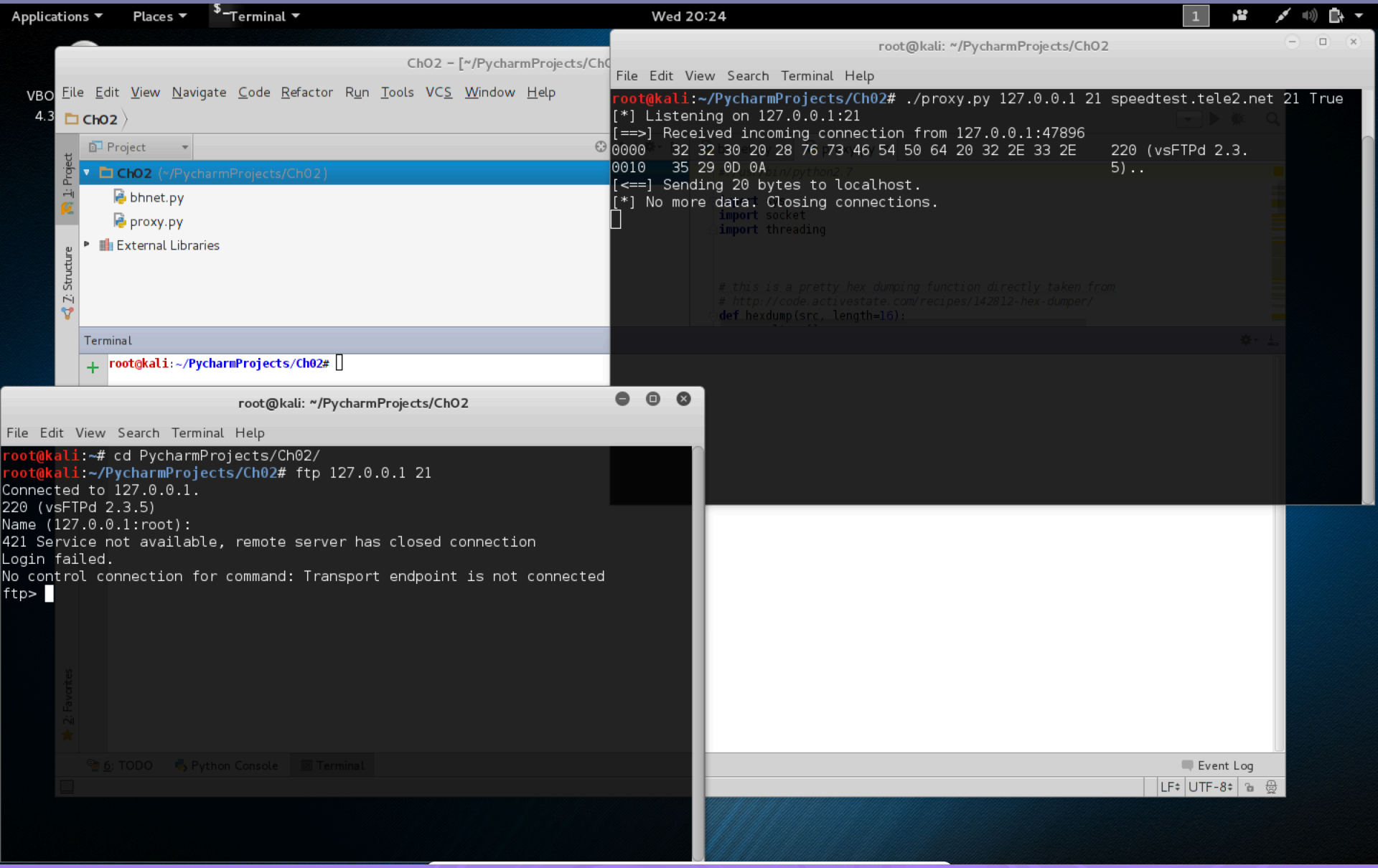
- In one terminal start the proxy server with
- `#./proxy.py 127.0.0.1 21 speedtest.tele2.net 21 True`
- The proxy server will inform that it is listening to the port 127.0.0.1:21

- In the second terminal start a ftp client that would like to connect to the address 127.0.0.1 and the port 21 with the command

```
# ftp 127.0.0.1 21
```

Study the program proxy.py

- The proxy server will start its role and the request to connect to 127.0.0.1:21 will be forwarded to the other server speedtest.tele2.net:21 and the received information from that server will be displayed


Applications
Places
Terminal
Wed 20:24
Ch02 – [~/PycharmProjects/Ch0
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Ch02
Project
Ch02 (~/PycharmProjects/Ch02)
bhnet.py
proxy.py
External Libraries
1: Project
7: Structure
Terminal
root@kali:~/PycharmProjects/Ch02#
root@kali: ~/PycharmProjects/Ch02
File Edit View Search Terminal Help
root@kali:~/PycharmProjects/Ch02# ./proxy.py 127.0.0.1 21 speedtest.tele2.net 21 True
[*] Listening on 127.0.0.1:21
[==>] Received incoming connection from 127.0.0.1:47896
0000 32 32 30 20 28 76 73 46 54 50 64 20 32 2E 33 2E 220 (vsFTPd 2.3.
0010 35 29 0D 0A 5)..
[<==] Sending 20 bytes to localhost.
[*] No more data. Closing connections.
import socket
import threading
# this is a pretty hex dumping function directly taken from
# http://code.activestate.com/recipes/142812-hex-dumper/
def hexdump(src, length=16):
root@kali: ~/PycharmProjects/Ch02
File Edit View Search Terminal Help
root@kali:~# cd PycharmProjects/Ch02/
root@kali:~/PycharmProjects/Ch02# ftp 127.0.0.1 21
Connected to 127.0.0.1.
220 (vsFTPd 2.3.5)
Name (127.0.0.1:root):
421 Service not available, remote server has closed connection
Login failed.
No control connection for command: Transport endpoint is not connected
ftp>
2: Favorites
6: TODO
Python Console
Terminal
Event Log
LF
UTF-8

Study the program proxy.py

- In the proxy.py change the waiting time from 2 seconds
- `connection.settimeout(2)`
- to
- 10 seconds
- `connection.settimeout(10)`
- Try to connect once more and put the user name guest

Study the program proxy.py

- Try to connect to the following anonymous ftp server

  ftp.csx.cam.ac.uk

  with username: anonymous

  password: something@some.com

Study the program proxy.py

- Study all defined functions in proxy.py

SSH with Paramiko

- Paramiko is installed in Python 2.7 version
- The sources in the book are not present in the folder Chapter 2 so copy-paste them from the textbook to PyCharm
- Try to follow the instructions in the book that apply for the Linux machine (not the Windows machine)

http://www.paramiko.org

Paramiko

A Python implementation of SSHv2.

Watch 222

build passing

Navigation

Quick search

Go

Enter search terms or a module, class or function name.

Welcome to Paramiko!

Paramiko is a Python (2.6+, 3.3+) implementation of the SSHv2 protocol [1], providing both client and server functionality. While it leverages a Python C extension for low level cryptography (Cryptography), Paramiko itself is a pure Python interface around SSH networking concepts.

This website covers project information for Paramiko such as the changelog, contribution guidelines, development roadmap, news/blog, and so forth. Detailed usage and API documentation can be found at our code documentation site, docs.paramiko.org.

Please see the sidebar to the left to begin.

Footnotes

[1] SSH is defined in RFC 4251, RFC 4252, RFC 4253 and RFC 4254. The primary working implementation of the protocol is the OpenSSH project. Paramiko implements a large portion of the SSH feature set, but there are occasional gaps.

Create a new file called *bh_sshcmd.py* and enter the following:

```
import threading
import paramiko
import subprocess

❶ def ssh_command(ip, user, passwd, command):
    client = paramiko.SSHClient()
❷   #client.load_host_keys('/home/justin/.ssh/known_hosts')
❸   client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    client.connect(ip, username=user, password=passwd)
    ssh_session = client.get_transport().open_session()
    if ssh_session.active:
❹       ssh_session.exec_command(command)
        print ssh_session.recv(1024)
    return

ssh_command('192.168.100.131', 'justin', 'lovesthepython','id')
```

This is a fairly straightforward program. We create a function called `ssh_command` ❶, which makes a connection to an SSH server and runs a single command. Notice that Paramiko supports authentication with keys ❷ instead of (or in addition to) password authentication. Using SSH key authentication is strongly recommended on a real engagement, but for ease of use in this example, we'll stick with the traditional username and password authentication.

Create a new file called *bh_sshcmd.py* and enter the following:

```
import threading
import paramiko
import subprocess

❶ def ssh_command(ip, user, passwd, command):
    client = paramiko.SSHClient()
    #client.load_host_keys('/home/justin/.ssh/known_hosts')
❷   client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
❸   client.connect(ip, username=user, password=passwd)
    ssh_session = client.get_transport().open_session()
    if ssh_session.active:
❹       ssh_session.exec_command(command)
        print ssh_session.recv(1024)
    return

ssh_command('192.168.100.131', 'justin', 'lovesthepython','id')
```

Put here the proper IP address of stud.ntnu.no, your proper username, proper password and some unix command (ls, pwd, …)

After successful connection end remote execution of the command, fill immediately the password field with '000000000000' and write the script once again.

This is a fairly straightforward program. We create a function called `ssh_command` ❶, which makes a connection to an SSH server and runs a single command. Notice that Paramiko supports authentication with keys ❷ instead of (or in addition to) password authentication. Using SSH key authentication is strongly recommended on a real engagement, but for ease of use in this example, we'll stick with the traditional username and password authentication.