

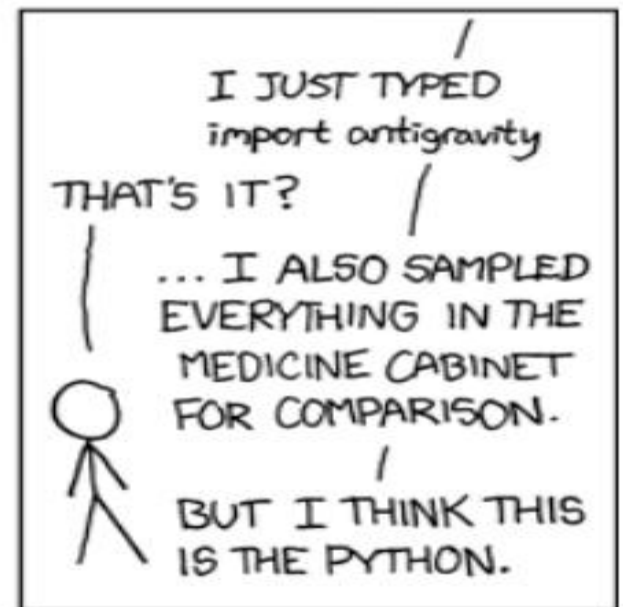
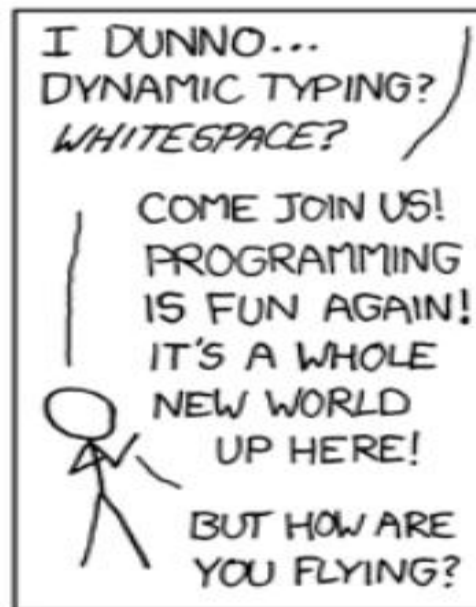
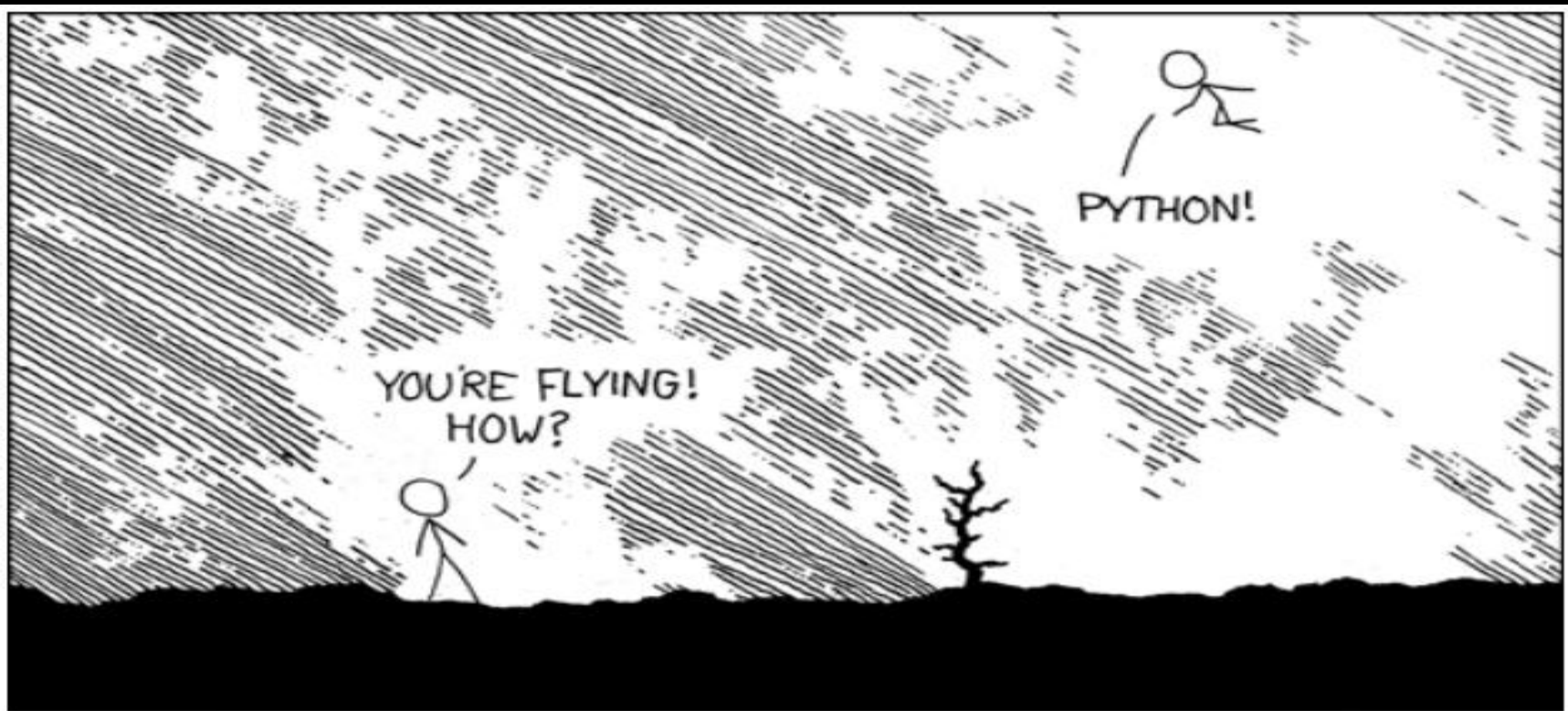
Writing Basic Security Tools using Python

Ali Al-Shemery

aka B!n@ry, @binaryz0ne

*Special thanks to Keith Dixon @Tazdrumm3r
for sharing his work...*

>>> **import** antigravity



Outline

- About Python
- Python Basics
 - Types
 - Controls
- Python Functions and Modules
- Python Tips and Tricks
- Coding for Penetration Testers

About Python

- Python is an open source programming language.
- Development started by Guido van Rossum in December 1989.
 - Conceived in the late 1980's
 - Python 2.0 was release on October 16th, 2000
 - Python 3.0 was released on December 2008
- Name came from TV series “**Monty Python's Flying Circus**”.

About Python – Cont.

- Python is cross platform
 - Linux (shipped out of the box)
 - Windows (easy to install)
 - Mac
 - Even work on your Droid!
 - etc

Why Learn Python?

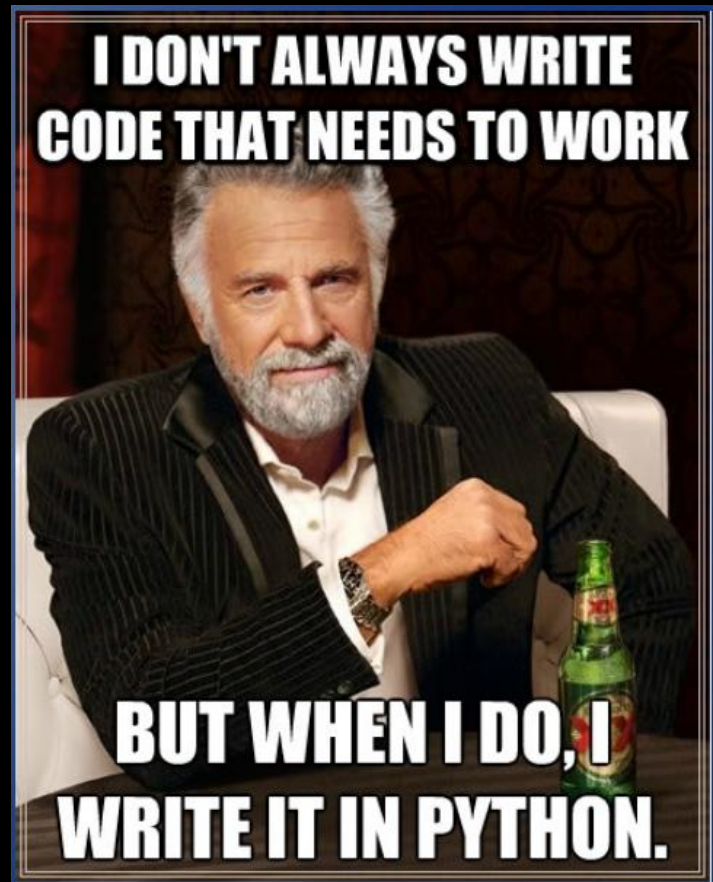
- Lot of people always ask me “**Why learn Python**”?
- The answer is simple:
 - Simple and easy to learn
 - Free and Open Source
 - Powerful high-level programming language
 - Widely used (Google, NASA, Yahoo, etc)
 - Portable
 - HUGE number of Extensive Libraries!

What is Python Good for?

- Ideal language for scripting and rapid application development in many areas on most platforms.
- All computer related subjects (IMO except system programming)
- Performing System Administration Tasks
- Encouraging and Helping Children start programming

What About Security?

- Extensive use in the information security industry
 - Exploit Development
 - Networking
 - Debugging
 - Encryption/Decryption
 - Reverse Engineering
 - Fuzzing
 - Web
 - Forensics
 - Malware analysis



Let's Start Working

- Interactive Interpreter

```
root@kali:~# python
Python 2.7.3 (default, Jan  2 2013, 13:56:14)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Text Editors
 - Vim, Nano, Geany (my favorite), Gedit, Kate, Notepad++, etc

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  # Code goes below
5
6
7
8
9
```

Python Basics

- Integers (int)

```
>>> httpPort=80
```

```
>>> Subnet=24
```

- Floating Point (float)

```
>>> 5.2/2
```

```
2.6
```

- Strings (str)

```
>>> url="http://www.linuxac.org/"
```

Playing with Strings

One of the most powerful capabilities of Python

- **String Slicing**

```
>>> logFile="/var/log/messages"
```

```
>>> logFile[0]
```

```
 '/'
```

```
>>> logFile[1:4]
```

```
 'var'
```

```
>>> logFile[-8:]
```

```
 'messages'
```

```
>>> logFile.split("/")
```

```
 ['', 'var', 'log', 'messages']
```

Playing with Strings – Cont.

- String Concatenation

```
>>> userName = "binary"
```

```
>>> domainName = "linuxac.org"
```

```
>>> userEmail = userName + "@" + domainName
```

```
>>> userEmail
```

```
'binary@linuxac.org'
```

```
>>> website="http://www.linuxac.org/"
```

```
>>> param="?p=123"
```

```
>>> url = "".join([website,param])
```

```
>>> url
```

```
'http://www.linuxac.org/?p=123'
```

Python Lists

- Python lists are very useful when you have a collection of elements

```
>>> portList = [21,22,25,80]
```

```
>>> portList[0]
```

```
21
```

```
>>> portList.append(443)
```

```
>>> portList
```

```
[21, 22, 25, 80, 443]
```

```
>>> portList.remove(22)
```

```
>>> portList
```

```
[21, 25, 80, 443]
```

```
>>> portList.insert(1,22)
```

```
>>> portList
```

```
[21, 22, 25, 80, 443]
```

```
>>> portList = []
```

```
>>> portList
```

```
[]
```

Lists in Python can be of any mixed type, even list of variables!!!

Python Controls - Decisions

- **IF, ELSE, and ELIF Statements**

```
>>> pList = [21,22,25,80]
>>> if pList[0] == 21:
...     print("FTP Service")
... elif pList[0] == 22:
...     print("SSH Service")
... else:
...     print("Unknown Service")
...
FTP
```

Important NOTE:

- Python doesn't use line terminators (ex: semicolons), but Python forces you to use indents

- Ensures writing elegant code!

Python Controls - Loops

- For and While Statements

```
>>> for port in pList:
...     print "This is port : ", port
...
This is port : 21
This is port : 22
This is port : 25
This is port : 80
```


Python Tips and Tricks

- Changing and checking data types

```
>>> httpPort=80
```

```
>>> httpPort
```

```
80
```

```
>>> type(httpPort)
```

```
<type 'int'>
```

```
>>> httpPort = str(httpPort)
```

```
>>> type(httpPort)
```

```
<type 'str'>
```

```
>>> httpPort
```

```
'80'
```

Python Tips and Tricks – Cont.

- Getting the length of an object

```
>>> len(pList)
```

```
4
```

- String formatting

```
>>> pList = [21,22,25,80]
```

```
>>> for member in pList:
```

```
...     print "This is port number %d" % member
```

```
...
```

```
This is port number 21
```

```
This is port number 22
```

```
This is port number 25
```

```
This is port number 80
```

Python Tips and Tricks – Cont.

- Another String formatting example

```
>>> ip = "192.168.1.1"
```

```
>>> mac = "AA:BB:CC:DD:EE:FF"
```

```
>>> print "The gateway has the following IP: %s and MAC: %s  
addresses" % (ip, mac)
```

The gateway has the following IP: 192.168.1.1 and MAC:
AA:BB:CC:DD:EE:FF addresses

Python Tips and Tricks – Cont.

- Working with ASCII codes

```
>>> x = '\x41'
```

```
>>> print x
```

A

- Converting to Hexadecimals

```
>>> hex(255)
```

```
'0xff'
```

```
>>> hex(0)
```

```
'0x0'
```

```
>>> hex(10)
```

```
'0xa'
```

```
>>> hex(15)
```

```
'0xf'
```

Python User Input

- Python can handle user input from different sources:
 - Directly from the user
 - From Text Files
 - From GUI (not covered in this training)

Python User Input – Cont.

- Directly from the user using `raw_input`

```
>>> userEmail = raw_input("Please enter your email  
address: ")
```

```
Please enter your email address: binary@linuxac.org
```

```
>>> userEmail  
'binary@linuxac.org'
```

```
>>> type(userEmail)  
<type 'str'>
```

Python User Input – Cont.

- From Text Files

```
>>> f = open("./services.txt", "r")
```

```
>>> for line in f:
```

```
...     print line
```

```
...
```

```
HTTP 80
```

```
SSH 22
```

```
FTP 21
```

```
HTTPS 443
```

```
SMTP 25
```

```
POP 110
```

```
>>> f.close()
```

Other common file functions:

- write
- read
- readline

Creating Functions

- Whenever you need to repeat a block of code, functions comes helpful
- Creating a Python Function (syntax)

```
def fName( listOfArguments ):
```

```
    Line1
```

```
    Line2
```

```
    ....
```

```
    Line n
```

```
    return something
```


Creating Functions – Cont.

- Basic function to check for valid port numbers

```
def checkPortNumber(port):  
    if port > 65535 or port < 0:  
        return False  
    else:  
        return True
```

- Howto use the **checkPortNumber** function:

```
print checkPortNumber(80) → True
```

```
print checkPortNumber(66000) → False
```

```
print checkPortNumber(-1) → False
```

Working with Modules

- Modules in Python are simply any file containing Python statements!
- Python is distributed with many modules
- To use a module:
 - `import module`
 - `import module1, module2, moduleN`
 - `import module as newname`
 - `from module import *`
 - `from module import <specific>`

Common Used Modules

- The most commonly used modules with security coding are:
 - string, re
 - os, sys, socket
 - hashlib
 - httplib, urllib2
 - Others? Please add ...

Modules and Examples

Module “sys”

- Check Python path, and count them

```
import sys
print "path has", len(sys.path), "members"
print "The members are:"
for member in sys.path:
    print member
```

- Print all imported modules:

```
>>> print sys.modules.keys()
```

- Print the platform type (linux, win32, mac, etc)

```
>>> print sys.platform
```

Module “sys” – Cont.

- Check application name, and list number of passed arguments

```
import sys
```

```
print “The application name is:”, sys.argv[0]
```

```
if len(sys.argv) > 1:
```

```
    print “You passed”, len(sys.argv)-1, “arguments. They are:”
```

```
    for arg in sys.argv[1:]:
```

```
        print arg
```

```
else:
```

```
    print “No arguments passed!”
```

Module “sys” – Cont.

- Check the Python working version

```
>>> sys.version
```

Module “os”

```
import os
```

- Check platform name (UNIX/Linux = posix, Windows = nt):

```
>>> os.name
```

- Print the current working directory

```
>>> os.getcwd()
```

- List files in specific directory

```
fList = os.listdir("/home")
```

```
for f in fList:
```

```
    print f
```


Module “os” – Cont.

- Remove a file (delete)

```
>>> os.remove("file.txt")
```

- Check the platform line terminator (Windows = `'\r\n'`, Linux = `'\n'`, Mac = `'\r'`)

```
>>> os.linesep
```

- Get the effective UID for current user

```
>>> os.geteuid()
```

- Check if file and check if directory

```
>>> os.path.isfile("/tmp")
```

```
>>> os.path.isdir("/tmp")
```

Module “os” – Cont.

- Run a shell command

```
>>> os.system("ping -c 2 127.0.0.1")
```

- Execute a command & return a file object

```
files = os.popen("ls -l /tmp")
```

```
for i in files:
```

```
    print i
```

Module “os” – Cont.

<code>os.system()</code>	# Executing a shell command
<code>os.stat()</code>	# Get the status of a file
<code>os.environ()</code>	# Get the users environment
<code>os.chdir()</code>	# Move focus to a different directory
<code>os.getcwd()</code>	# Returns the current working directory
<code>os.getgid()</code>	# Return the real group id of the current process
<code>os.getuid()</code>	# Return the current process’s user id
<code>os.getpid()</code>	# Returns the real process ID of the current process
<code>os.getlogin()</code>	# Return the name of the user logged
<code>os.access()</code>	# Check read permissions
<code>os.chmod()</code>	# Change the mode of path to the numeric mode
<code>os.chown()</code>	# Change the owner and group id
<code>os.umask(mask)</code>	# Set the current numeric umask
<code>os.getsize()</code>	# Get the size of a file

Module “os” – Cont.

`os.path.getmtime()` # Last time a given directory was modified
`os.path.getatime()` # Last time a given directory was accessed
`os.environ()` # Get the users environment
`os.uname()` # Return information about the current OS
`os.chroot(path)` # Change the root directory of the current process to path

`os.listdir(path)` # List of the entries in the directory given by path
`os.getloadavg()` # Show queue averaged over the last 1, 5, and 15 minutes

`os.path.exists()` # Check if a path exists
`os.walk()` # Print out all directories, sub-directories and files

Module “os” – Cont.

<code>os.mkdir(path)</code>	<code># Create a directory named path with numeric mode mode</code>
<code>os.makedirs(path)</code>	<code># Recursive directory creation function</code>
<code>os.remove(path)</code>	<code># Remove (delete) the file path</code>
<code>os.removedirs(path)</code>	<code># Remove directories recursively</code>
<code>os.rename(src, dst)</code>	<code># Rename the file or directory src to dst</code>
<code>os.rmdir(path)</code>	<code># Remove (delete) the directory path</code>

Module “socket”

```
import socket
```

- **Creating a simple TCP client**
 - Check simpleClient.py
- **Creating a simple TCP server**
 - Check simpleServer.py
- **Create a malicious FTP Client**
 - ftpClient.py

Module “socket” – Cont.

- Create TCP Socket, then send and receive data from website using the socket

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("www.linuxac.org", 80))
s.send('GET / HTTP/1.1 \r\nHost: www.linuxac.org\r\n\r\n')
data = s.recv(2048)
s.close()
print data
```

Note: For UDP Sockets use SOCK_DGRAM instead of SOCK_STREAM

Module “pcapy”

- **Pcap**y is a Python extension module that interfaces with the libpcap packet capture library.
- Pcapy enables python scripts to capture packets on the network.
- Pcapy is highly effective when used in conjunction with a packet-handling package such as Impacket, which is a collection of Python classes for constructing and dissecting network packets.
- Packet Capturing using pcapy example
 - pcapyPktCapture1.py
 - pcapyEx1.py
 - pcapyDumper.py

Module “urllib” & “urllib2”

- **urllib2** is a Python module for fetching URLs.
- Offers a very simple interface, in the form of the `urlopen` function.
- Capable of fetching URLs using a variety of different protocols (`http`, `ftp`, `file`, etc)
- Also offers a slightly more complex interface for handling common situations:
 - Basic authentication
 - Cookies
 - Proxies
 - etc

urllib vs urllib2

- Both modules do URL request related stuff, but they have different functionality.
- urllib2 can accept a Request object to set the headers for a URL request, urllib accepts only a URL.
- urllib provides the urlencode method which is used for the generation of GET query strings, urllib2 doesn't have such a function.
- Because of that urllib and urllib2 are often used together.

Example 1

```
import urllib2
request = urllib2.Request('http://www.linuxac.org')
response = urllib2.urlopen(request)
payload = response.read()
print(payload)
```

Basic URL Request

```
import urllib2
response =
    urllib2.urlopen('http://pythonforbeginners.com/')
print response.info()
html = response.read()
response.close()
```

Base64 & ROT13 Encoders

Base64

```
#!/usr/bin/python
code = raw_input("Enter the data you wish to be encoded to
Base64")
answer=code.encode('base64','strict')
print answer
```

ROT13

```
#!/usr/bin/python
code = raw_input("Enter the data you wish to be encoded to
Base64")
answer=code.encode('rot13','strict')
print answer
```

Exploit Development

```
#!/usr/bin/python
import socket
host = "target"
port = <port#>
cmd = "initial command"
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
buffer = "buffer to send"
shellcode = "shellcode"
Payload = cmd + buffer + shellcode
print "\n Any status message \n"
s.connect((host,port))
data = s.recv(1024)
s.send(payload + "\n")
s.close
```

Packet Crafting with Scapy

Scapy Overview

- Scapy is a Python program that enables the user to send, sniff and dissect and forge network packets.
- This capability allows construction of tools that can probe, scan or attack networks.
- It can replace hping, arpspoof, arp-sk, arping, p0f and even some parts of Nmap, tcpdump, and tshark.

Scapy Overview – Cont.

- Scapy was created by Philippe Biondi and runs in Python:
 - Can be used interactively at a Python prompt
 - Included within Python scripts for more complex interactions
- Must run with root privileges to craft packets,
- Don't need to be a Python Guru to use Scapy!

Scapy Basics - 1

- Supported protocols:

```
>>> ls()
```

- Details about a specific protocol:

```
>>> ls(TCP)
```

- Available commands/functions:

```
>>> lsc()
```

Scapy Basics - 2

- Crafting a SYN/ACK Packet

```
>>> pkt = IP(dst="192.168.122.101")
```

```
>>> pkt /= TCP(dport=80, flags="SA")
```

- Crafting ICMP Host Unreachable Packet

```
>>> pkt = IP(dst="192.168.122.101")
```

```
>>> pkt /= ICMP(type=3,code=1)
```

Scapy Basics - 3

Single Line:

- ICMP echo request Packet

```
>>> mypkt = IP(dst="192.168.122.101")  
/ICMP(code=0,type=8)
```

- TCP FIN, Port 22, Random Source Port, and Random Seq#

```
>>> mypkt = IP(dst="192.168.122.101")  
/TCP(dport=22,sport=RandShort(),seq=RandShort(  
) ,flags="F")
```

Sending and Receiving Packets

– @L3

- Send packet at layer 3
>>> send(packet)
- Send packet at L3 and receive one response
>>> resp = sr1(packet)
- Send packet at L3 and receive all responses
>>> ans,unans = sr(packet)

Sending and Receiving Packets

– @L2

- Send packet at layer 2
>>> sendp(Ether()/packet)
- Send packet at L2 and receive one response
>>> resp = srp1(packet)
- Send packet at L2 and receive all responses
>>> ans,unans = srp(packet)

Displaying Packets

- Get a summary of each packet:

```
>>> pkts.summary()
```

- Get the whole packet list:

```
>>> pkts.show()
```

Scapy Host Discovery

```
>>> ans,unans =  
    srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst="192.168.1  
    22.0/24"),timeout=2)
```

```
>>> ans.summary(lambda(s,r): r.sprintf("Ether:  
    %Ether.src% \t\t Host: %ARP.psrc%"))
```


Scapy Port Scanning

- TCP SYN Scanner

```
>>> sr1(IP(dst="192.168.122.101")  
/TCP(dport=90,flags="S"))
```

```
>>> a,u = sr(IP(dst="192.168.122.101")  
/TCP(dport=(80,100),flags="S"))
```

```
>>> a.summary(lambda(s,r): r.sprintf("Port:  
%TCP.sport% \t\t Flags: %TCP.flags%"))
```

Scapy Sniffing - 1

- Scapy has powerful capabilities to capture and analyze packets.
- Configure the network interface to sniff packets from:

```
>>> conf.iface="eth0"
```

Configure the scapy sniffer to sniff only 20 packets

```
>>> pkts=sniff(count=20)
```

Scapy Sniffing - 2

- Sniff packets and stop after a defined time:

```
>>> pkts=sniff(count=100,timeout=60)
```

- Sniff only packets based on a filter:

```
>>> pkts = sniff(count=100,filter="tcp port 80")
```

Scapy Sniffing - 3

```
>>> pkts = sniff(count=10,prn=lambda  
x:x.sprintf("SrcIP={IP:%IP.src% ->  
DestIP=%IP.dst%} |  
Payload={Raw:%Raw.load% \n}"))
```

- What is that doing ???

Exporting Packets

- Sometimes it is very useful to save the captured packets in a PCAP file for future work:

```
>>> wrpcap("file1.cap", pkts)
```

- Dumping packets in HEX format:

```
>>> hexdump(pkts)
```

- Dump a single packet in HEX format:

```
>>> hexdump(pkts[2])
```

- Convert a packet to hex string:

```
>>> str(pkts[2])
```

- Exporting to Base64 encoded packets:

```
>>> export_object(pkts)
```

Importing Packets

- To import from a PCAP file:

```
>>> pkts = rdpcap("file1.cap")
```

- Or use the scapy sniffer but with the offline argument:

```
>>> pkts2 = sniff(offline="file1.cap")
```

Create your own tools

```
>>> def handler(packet):  
        hexdump(packet.payload)  
>>> sniff(count=20, prn=handler)  
  
>>> def handler2(packet):  
        sendp(packet)  
>>> sniff(count=20, prn=handler2)
```

Python Tools for Penetration Testers

Network Tools

- [Scapy](#): send, sniff and dissect and forge network packets. Usable interactively or as a library
- [pypcap](#), [Pcap](#) and [pylibpcap](#): several different Python bindings for libpcap
- [libdnet](#): low-level networking routines, including interface lookup and Ethernet frame transmission
- [dpkt](#): fast, simple packet creation/parsing, with definitions for the basic TCP/IP protocols
- [Impacket](#): craft and decode network packets. Includes support for higher-level protocols such as NMB and SMB
- [pynids](#): libnids wrapper offering sniffing, IP defragmentation, TCP stream reassembly and port scan detection
- [Dirtbags py-pcap](#): read pcap files without libpcap
- [flowgrep](#): grep through packet payloads using regular expressions
- [Knock Subdomain Scan](#), enumerate subdomains on a target domain through a wordlist
- [Mallory](#), extensible TCP/UDP man-in-the-middle proxy, supports modifying non-standard protocols on the fly
- [Pytbull](#): flexible IDS/IPS testing framework (shipped with more than 300 tests)

Debugging and Reverse Engineering Tools

- Paimei: reverse engineering framework, includes PyDBG, PIDA, pGRAPH
- Immunity Debugger: scriptable GUI and command line debugger
- mona.py: PyCommand for Immunity Debugger that replaces and improves on pvefindaddr
- IDAPython: IDA Pro plugin that integrates the Python programming language, allowing scripts to run in IDA Pro
- PyEMU: fully scriptable IA-32 emulator, useful for malware analysis
- pefile: read and work with Portable Executable (aka PE) files
- pydasm: Python interface to the libdasm x86 disassembling library
- PyDbgEng: Python wrapper for the Microsoft Windows Debugging Engine
- uhooker: intercept calls to API calls inside DLLs, and also arbitrary addresses within the executable file in memory
- diStorm: disassembler library for AMD64, licensed under the BSD license
- python-ptrace: debugger using ptrace (Linux, BSD and Darwin system call to trace processes) written in Python
- vdb / vtrace: vtrace is a cross-platform process debugging API implemented in python, and vdb is a debugger which uses it
- Androguard: reverse engineering and analysis of Android applications

Fuzzing Tools

- [Sulley](#): fuzzer development and fuzz testing framework consisting of multiple extensible components
- [Peach Fuzzing Platform](#): extensible fuzzing framework for generation and mutation based fuzzing (v2 was written in Python)
- [antiparser](#): fuzz testing and fault injection API
- [TAOF](#), (The Art of Fuzzing) including ProxyFuzz, a man-in-the-middle non-deterministic network fuzzer
- [untidy](#): general purpose XML fuzzer
- [Powerfuzzer](#): highly automated and fully customizable web fuzzer (HTTP protocol based application fuzzer)
- [SMUDGE](#)
- [Mistress](#): probe file formats on the fly and protocols with malformed data, based on pre-defined patterns
- [Fuzzbox](#): multi-codec media fuzzer
- [Forensic Fuzzing Tools](#): generate fuzzed files, fuzzed file systems, and file systems containing fuzzed files in order to test the robustness of forensics tools and examination systems
- [Windows IPC Fuzzing Tools](#): tools used to fuzz applications that use Windows Interprocess Communication mechanisms
- [WSBang](#): perform automated security testing of SOAP based web services
- [Construct](#): library for parsing and building of data structures (binary or textual). Define your data structures in a declarative manner
- [fuzzer.py \(feliam\)](#): simple fuzzer by Felipe Andres Manzano
- [Fusil](#): Python library used to write fuzzing programs

Web Tools

- Requests: elegant and simple HTTP library, built for human beings
- HTTPIe: human-friendly cURL-like command line HTTP client
- ProxMon: processes proxy logs and reports discovered issues
- WSMap: find web service endpoints and discovery files
- Twill: browse the Web from a command-line interface. Supports automated Web testing
- Ghost.py: webkit web client written in Python
- Windmill: web testing tool designed to let you painlessly automate and debug your web application
- FunkLoad: functional and load web tester
- spynner: Programmatic web browsing module for Python with Javascript/AJAX support
- python-spidermonkey: bridge to the Mozilla SpiderMonkey JavaScript engine; allows for the evaluation and calling of Javascript scripts and functions
- mitmproxy: SSL-capable, intercepting HTTP proxy. Console interface allows traffic flows to be inspected and edited on the fly
- pathod / pathoc: pathological daemon/client for tormenting HTTP clients and servers

Forensic Tools

- Volatility: extract digital artifacts from volatile memory (RAM) samples
- LibForensics: library for developing digital forensics applications
- TrIDLib, identify file types from their binary signatures. Now includes Python binding
- aft: Android forensic toolkit

Malware Analysis Tools

- [pyew](#): command line hexadecimal editor and disassembler, mainly to analyze malware
- [Exefilter](#): filter file formats in e-mails, web pages or files. Detects many common file formats and can remove active content
- [pyClamAV](#): add virus detection capabilities to your Python software
- [jsunpack-n](#), generic JavaScript unpacker: emulates browser functionality to detect exploits that target browser and browser plug-in vulnerabilities
- [yara-python](#): identify and classify malware samples
- [phoneyc](#): pure Python honeyclient implementation

PDF Tools

- [Didier Stevens' PDF tools](#): analyse, identify and create PDF files (includes [PDFiD](#), [pdf-parser](#) and [make-pdf](#) and mPDF)
- [Opaf](#): Open PDF Analysis Framework. Converts PDF to an XML tree that can be analyzed and modified.
- [Origapy](#): Python wrapper for the Origami Ruby module which sanitizes PDF files
- [pyPDF](#): pure Python PDF toolkit: extract info, spilt, merge, crop, encrypt, decrypt...
- [PDFMiner](#): extract text from PDF files
- [python-poppler-qt4](#): Python binding for the Poppler PDF library, including Qt4 support

Lab Time!

DIY 😊

This lab is a Do It Yourself (DIY) Lab that must be done at home:

- [1] Create a TCP ACK Port Scanner
- [2] Create a TCP Replay Tool
- [3] Create a UDP Ping Tool
- [4] Create a Sniffer that filters based on user input
- [5] Create a tool for HTTP Basic Authentication over
- [6] Create a basic Honeypot that logs all activity to a text file

SUMMARY

- Discussed Why Learn Python
- Discussed What is Python Good for?
- Explained Python Basics:
 - Integers, Floating point, etc
 - Strings,
 - Lists,
 - Controls,
 - etc
- Some Quick Python Tips and Tricks
- Python User Input
- Howto Create Functions using Python
- Working with Modules, and the Python Common Used Modules

SUMMARY – Cont.

- Howto use the Python SYS and OS Modules
- Using Python to work with Networks: Sockets, pcap, etc
- Using Python to work with the Web (urllib, urllib2)
- Using Python to create simple Encoders
- Howto use Python for Exploit Development
- Craft your own packets using Scapy
- Python tools for penetration testers
- DIY Labs

Works Cited

- [1] Python Comic, <http://xkcd.com/353/>,
- [2] Keith Dixon, @Tazdrumm3r, <http://tazdrumm3r.wordpress.com/>
- [3] Live Packet Capture in Python with pcap, <http://snipplr.com/view/3579/live-packet-capture-in-python-with-pcap/>
- [4] How to use urllib2 in Python, <http://www.pythonforbeginners.com/python-on-the-web/how-to-use-urllib2-in-python/>
- [5] Python tools for penetration testers, <http://www.dirk-loss.de/python-tools.htm>

References

- [1] Coding for Penetration Testers Book,
- [2] Violent Python Book,
- [3] Scapy Documentation, <http://www.secdev.org/projects/scapy/doc/>
- [4] Python, <http://www.python.org/>
- [5] Python Infosec tools, <http://www.dirk-loss.de/python-tools.htm>
- [6] Grow Your Own Forensic Tools: A Taxonomy of Python Libraries Helpful for Forensic Analysis, http://www.sans.org/reading_room/whitepapers/incident/grow-forensic-tools-taxonomy-python-libraries-helpful-forensic-analysis_33453
- [7] Python Docs, <http://docs.python.org/>
- [8] Python Tutorial, <http://www.tutorialspoint.com/python/index.htm>
- [9] pcap, <http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=tool&name=Pcap>
- [10] Basic Authentication Authentication with Python, <http://www.voidspace.org.uk/python/articles/authentication.shtml>
- [11] Justin Searle, Python Basics for Web App Pentesters, InGuardians Inc