Disciplina de Segurança
Teste de Penetração - Pentesters

Baseado nas apresentações de Ali Al-Shemery

Escrevendo Ferramentas de Segurança Básica em Python

Outline

- Python Básico
  - Types
  - Controls
- Python – Functions e Modules
- Python - Tips and Tricks
- Escrevendo Pentesters em Python

Porque aprender Python?

- Fácil de aprender
- Código aberto e gratuíto - multiplataforma
- Uma linguagem de programação modular de alto-nivel
- Inúmeros módulos, bibliotecas e aplicações desenvolvidas e disponibilizadas (“muitos mesmo”)
- Portável

Quais as áreas que serão tratadas ...

- Criptografia e Decriptografia (senhas e arquivos criptografados)
- Análise de Redes e Serviços (Portas)
- Testes de Penetração
- Investigações Forenses
- Análise de Tráfego de Rede
- Ataque em redes Wireless e Bluetooth
- Mineração Web em Sites (Scrapi python)
- Iludindo antivirus

Python

- Interpretador em modo interativo

```
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.
>>>
```

- Programas através de Editores de Texto
  - Vim, Nano,
    PyCharm
    Gedit, Kate,
    Ultraedit, Notepad++

```
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/"
```

Algumas funções básicas com Strings

- Substrings

```
>>> logFile="/var/log/messages"
>>> logFile[0]
    '/'
>>> logFile[1:4]
    'var'
>>> logFile[-8:]
'messages'
>>> logFile.split("/")
['', 'var', 'log', 'messages']
```

Algumas funções básicas com Strings

- Concatenação de Strings

```
>>> userName = “ali”
>>> domainName = “ashemery.com”
>>> userEmail = userName + “@” + domainName
>>> userEmail
'ali@ashemery.com‘

>>> website="http://www.ashemery.com/"
>>> param="?p=123"
>>> url = "".join([website,param])
>>> url
'http://www.ashemery.com/?p=123'
```

Python - Listas

- Listas no Python represented coleção de elementos

```
>>> 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
[]
```

Listas em Python podem misturar tipos nativos.

Python Controls – Decisões (If)

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

Nota: Python não usa terminadores de linha mas força uma identação coerente.

Python Controls – Loops (for)

```
pList = [21,22,25,80]
>>> 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 Controls – Loops (for)

```
a = [1, 2, 3]
>>> for i in [0,1,2]:
...     print(“Numero : ", a[i])
...
Numero :  1
Numero :  2
Numero :  3


>>> for i in range(0,3):
...     print("Numero : ", a[i])
```

Python – Outras funcionalidades

- Tamanho de objetos (Strings e listas)

```
>>> len(pList)
4
```

- Strings formatados

```
>>> 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 – Outras funcionalidades

- Outros exemplos de formatação

```
>>> 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 – Outras funcionalidades

- Caracteres (ASCII codes)

```
>>> x = '\x41‘
>>> print(x)
A
```

- Convertendo para Hexadecimais

```
>>> hex(255)
'0xff'
>>> hex(0)
'0x0'
>>> hex(10)
'0xa'
>>> hex(15)
'0xf'
```

Python – Entrada de Dados

- O Python pode manipular a entrada do usuário de diferentes origens:
  - Diretamente de uma entrada do usuário (e.g. teclado)
  - De arquivos
  - De GUI (não coberto neste estudo)

Python User Input – Cont.

- Diretamente do teclado

```
>>> userEmail = input("Please enter your email address: ")
Please enter your email address: ali@ashemery.com

>>> userEmail
'ali@ashemery.com'

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

Python User Input – Cont.

- De arquivos textos

```
>>> 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()
```

Outras funções de arquivo:
- write
- read
- readline

Criando suas funções

```
def fName( listOfArguments ):
    Line1
    Line2
    ….
    Line n
    return something
```

```
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
```

Trabalhando com módulos (bibliotecas)

- Módulos em Python são simplesmente qualquer arquivo contendo instruções em Python!
- Python é distribuído com muitos módulos
- Para usar um módulo:
  - import module
  - import module1, module2, moduleN
  - import module as newname
  - from module import *
  - from module import <specific>
  - import antigravity

Módulos de uso comum

- import string
- import re
- import os
- import sys
- import hashlib
- import urllib
- import socket
- import ssl
- import nmap
- import dpkt
- import pyftpdlib
- Import scapy
- import pcapy
  (para o pcapy em Windows 10 existe restrição a ser resolvida adiante)

Se um import der erro, digite …
pip install <nome_do_modulo>
... na lnha de comando

Módulos e Exemplos

https://docs.python.org/3/library/

Módulo “sys”

- Check Python path, e contagens

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

- Imprimir todos os módulos importados=

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

- Imprimindo dados do Python usado (linux, win32, mac, etc)

```
>>> print(sys.platform)
```

Módulo “sys”

- Checando dados da aplicação e parâmetros passados

```
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!“)
```

- Verificando a versão do Python

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

Módulo “os”

```
import os
```

- Nome da Plataforma / S.O. (UNIX/Linux = posix, Windows = nt):

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

- Diretório corrente

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

- Arquivos de um diretório específico

```
fList = os.listdir("/home")
for f in fList:
    print(f)
```

Módulo “os”

- Apaga arquivos

```
>>> os.remove(“file.txt")
```

- Verificando terminador de linhas (Windows = ‘\r\n’ , Linux = ‘\n’ , Mac = ‘\r’)

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

- Pega o UID do usuário corrente

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

- Verificar a existência de arquivo ou diretório

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

Módulo “os”

- Shell command

```
>>> os.system("ping 127.0.0.1")
```

- Executa um comando de S.O. e returna o resultado #linux

```
files = os.popen("ls -l /tmp")
for i in files:
    print(i)

files = os.popen(“dir /w c:\temp") #windows
for i in files:
    print(i)
```

Módulo "os"

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

Módulo "os"

```
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
```

Módulo "os"

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

Executando Programas Externos

- Executar programas externos é muito útil quando você precisa fazer automação (como em scripts) ou executar soluções de baixo nível (no S.O.) e capturar o resultado para tratamento e análise no Python.
- O Python pode executar programas de modo ...
  - Síncrono
    - Executa um comando externo e aguarda o retorno
  - Assíncrono
    - Retorna imediatamente e continua no processo principal

http://helloacm.com/execute-external-programs-the-python-ways/

Executando Programas Externos

- Use do modulo “os” o …
  - popen(), system(), startfile()

```
>>> import os
>>> print(os.popen("echo Hello, World!").read())
```

- O os.popen() trata a saída do commando (stdout, stderr) como um objeto do tipo arquivo, logo você pode capturer a saída do programa executado.

Executando Programas Externos

- O os.system() é síncrono e pode retornar o status de saída

```
>>> import os
>>> print(os.system(“notepad.exe”))
```

Executando Programas Externos

- Assim como o clique duplo sobre uma arquivo no Explorer, ainda pode ser usado o os.startfile() que “lança” o programa externo associado ao tipo do arquivo. Este método é assíncrono.

```
>>> import os
>>> os.startfile(“services.txt”)
```

  - WindowsError: [Error 2] The system cannot find the file specified:

Executando Programas Externos

- O pacote subprocess fornece um método síncronos e assíncronos, chamados de “call” e “Popen”

- Ambos os métodos usam o primeiro parâmetro como uma lista

```
import subprocess
subprocess.call([“notepad.exe”, “services.txt”])
subprocess.Popen([“notepad.exe”])
# thread continues ...
```

Executando Programas Externos

- Você ainda pode usar wait() para tornar o processo síncrono

```
import subprocess
p = subprocess.Popen(“ls”, shell=True, stdout=subprocess.PIPE,
  stderr=subprocess.STDOUT)
for line in p.stdout.readlines():
   print(line)
retval = p.wait()
print(retval)
```

Módulo “socket”

- Criando um cliente HTTP

```
import socket
import http.client
conn = http.client.HTTPConnection("localhost", 8000)
conn.request("GET","/index.html")
r1 = conn.getresponse()
print(r1.status, r1.reason)
print(r1.read(200))
```

- Criando um servidor HTTP (default port: 8000)

```
python -m http.server
```

- Criando um servidorFTP (default port: 2121)

```
python -m pyftpdlib
```

SimpleServer.pdf

Módulo "socket"

- Criando um socket TCP, e mandando e recebendo dados

```
import socket
s = socket.create_connection(('www.google.com', 80))
s.send("GET / HTTP/1.1\r\n\r\n".encode())
x = str(s.recv(10000))
print(x)
```

Nota: Para sockets UDP use SOCK_DGRAM ao invés de SOCK_STREAM

Módulo "pcapy"

- Pcapy é um módulo do Python que faz interface com a biblioteca de captura de pacotes libpcap.
- O Pcapy permite que scripts python capturem pacotes na rede.
- O Pcapy é altamente eficaz quando usado em conjunto com módulos de manipulação de pacotes, como o Impacket, que é uma coleção de classes Python para construir e detalhar pacotes de rede.
- Exemplo de captura de pacotes usando o pcapy.
  - pcapyEx1.py

Módulo “pcapy”

- Para instalação com o python 3.6.5 em Windows 10
  - To fix Python 3 on Windows 10 error, Microsoft Visual C++ 14.0 is required (para Windows 10)
  - Fix the "error Microsoft Visual C++ 14.0 is required“for Python 3.6 and 3.7 on Windows 10 :
    - It requires about 6 GB of disk space (for Visual Studio Studio 2017 Build Tools).
    - Install Microsoft Build Tools for Visual Studio 2017 <https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=BuildTools&rel=15>.
    - Select: Workloads -> Visual C++ build tools. (Cargas de Trabalho -> Ferramentas do Visual C++)
    - Install options: select only the "Windows 10 SDK”"(SDK do Windows 10)

Módulo “urllib” (Python 3)

- urllib é um módulo Python para recuperar URLs.
- Em geral usamos o “import urllib.request”
- Oferece uma interface muito simples, na forma das funções “urllib.request.Request” e “urllib.request.urlopen”
- Esse módulo é capaz de buscar URLs usando uma variedade de protocolos diferentes (http, ftp, file, etc.)
- Também oferece uma interface um pouco mais complexa para lidar com situações comuns:
  - Basic authentication
  - Cookies
  - Proxies
  - etc

Módulo “urllib” (Python 3)

- urllib do Python 3.x aceita um objeto Request para definir os cabeçalhos para uma solicitação de URL
- urllib fornece o método urlencode que é usado para a geração de strings de consulta GET
- Por causa disso, o urllib e o urllib2 são frequentemente usados juntos ou dependendo da necessidade.

Exemplo do “urllib”

```
import urllib.request
req = urllib.request.Request('http://www.google.com')
response = urllib.request.urlopen(req)
the_page = response.read()
print(the_page)
```

Criando pacotes com Scapy

Scapy Overview

- Scapy é um programa em Python que permite ao usuário enviar, “sniff”, detalhar e criar pacotes de rede.
- Esse recurso permite a construção de ferramentas que podem sondar, varrer ou atacar redes
- Ele pode substituir hping, arpspoof, arp-sk, arping, p0f e até em algumas partes o nmap, tcpdump e tshark, porém seu uso é de mais baixo nível

Scapy Overview

- Pode ser usado interativamente em um prompt do Python
- Pode ser usado incluído em scripts Python para interações mais complexas

- **Deve ser executado com privilégios de root para criar pacotes**

from scapy.all import *
Ifaces
conf
conf.iface="eth0“
Conf.iface=“Adaptador Ethernet Ethernet”

Scapy Basics

from scapy.all import *

- Criando um pacote SYN/ACK

>>> pkt = IP(dst="127.0.0.1")
>>> pkt /= TCP(dport=80, flags="SA")

- Criando um pacote “ICMP Host Unreachable”

>>> pkt = IP(dst="127.0.0.1 ")
>>> pkt /= ICMP(type=3,code=1)

Scapy Basics

- Pacote de “ICMP echo request”

```
>>> mypkt = IP(dst="127.0.0.1") /ICMP(code=0,type=8)
```

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

```
>>> mypkt = IP(dst="127.0.0.1") /
  TCP(dport=22,sport=RandShort(),seq=RandShort(),flags="F")
```

Mandando e Recebendo Pacotes (L3)

- Mandando pacotes para a camada 3 (L3 - Roteamento)

```
>>> send(packet)
```

- Mandando um pacote de na L3 e recebendo uma resposta

```
>>> resp = sr1(packet)
```

- Mandando um pacote de na L3 e recebendo todas as respostas

```
>>> ans,unans = sr(packet)
```

Mandando e Recebendo Pacotes (L2)

- Mandando pacotes para camada 2 (L2 – Enlace)

```
>>> sendp(Ether()/packet)
```

- Mandando um pacote para camada L2 e recebendo uma resposta

```
>>> resp = srp1(packet)
```

- Mandando um pacote para camada L2 e recebendo todas as respostas

```
>>> ans,unans = srp(packet)
```

Mostrando o Conteúdo de Pacotes

- Pega o sumário de cada pacote:

```
>>> mypkt.summary()
```

- Recuperando todos os elementos do pacote

```
>>> mypkt.show()
```

Descoberta de Hosts com Scapy

```
>>> ans,unans =
  srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst="127.0.0.1/32"),
  timeout=2)

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

Scanner de Porta com o Scapy

- TCP SYN Scanner

```
>>> sr1(IP(dst="127.0.0.1") /TCP(dport=90,flags="S"))

>>> a,u = sr(IP(dst="127.0.0.1") /TCP(dport=(80,100),flags="S"))

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

Scapy “Sniffing”

- O Scapy possui recursos poderosos para capturar e analisar pacotes.
- Configure a interface de rede para capturar pacotes...

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

Configure to “scapy sniffer” para “sniffar” apenas 20 pacotes

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

Scapy “Sniffing”

- “Sniffando” pacotes e parando após um determinado tempo

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

- “Sniffando” pacotes baseados em um filtro:

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

Scapy Sniffing (avançado com função lambda)

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

Exportando Pacotes

- Às vezes é muito útil salvar os pacotes capturados em um arquivo PCAP para usos futuros:

```
>>> 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])
```

Nota: Os alunos recordam de arquivos PCAP e wireshark?

Importando Pacotes

- Importando de um arquivo PCAP:

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

- … ou use um pipeline do próprio sniffer do scapy como um recurso “offline”:

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

Criando suas próprias interfaces

```
>>> def handler(packet):
        hexdump(packet.payload)

>>> sniff(count=20, prn=handler)

>>> def handler2(packet):
        sendp(packet)

>>> sniff(count=20, prn=handler2)
```

```
#!/usr/bin/env python
import sys
from scapy.all import *
def findSYN(p):
    flags = p.sprintf("%TCP.flags%")
    if flags == "S":      # Only respond to SYN Packets
        ip = p[IP]              # Received IP Packet
        tcp = p[TCP]  # Received TCP Segment
        i = IP()                # Outgoing IP Packet
        i.dst = ip.src
        i.src = ip.dst
        t = TCP()      # Outgoing TCP Segment
        t.flags = "SA"
        t.dport = tcp.sport
        t.sport = tcp.dport
        t.seq = tcp.ack
        new_ack = tcp.seq + 1
        print ("SYN/ACK sent to ",i.dst,":",t.dport)
        send(i/t)
```

```
sniff(prn=findSYN)
```

Outros

Adicionando um delay de tempo

- Delay de 5 segundos

```
>>> import time
>>> time.sleep(5)
```

- Executa “algo” a cada minute:

```
import time
while True:
  print "This prints once a minute.”
  time.sleep(60)
```

http://stackoverflow.com/questions/510348/how-can-i-make-a-time-delay-in-python

Ferramentas Python for para Pentester - Penetration Testers

Ferramentas de Rede

- Scapy: send, sniff and dissect and forge network packets. Usable interactively or as a library
- pypcap, Pcapy 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)

Cited [5]

Ferramentas de Debugger e Engenharia Reversa

- 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

Cited [5]

Ferramentas de Debugger e Engenharia Reversa

- 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

Cited [5]

Ferramentas de Fuzzing

- 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

Cited [5]

Ferramentas de Fuzzing

- 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

Cited [5]

Ferramentas para Web Recon

- 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

Ferramentas para Web Recon

- 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

Cited [5]

Ferramentas para Análise Forense

- 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

Ferramentas de Análise de Malwares

- 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

Cited [5]

Geradores de PDF

- 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

Cited [5]