Aplicação de Segurança Informática

Pedro Moreira - 10015 e João Carlos Mendes, 12825

Abstract—Este trabalho insere-se numa avaliação da disciplina Linguagens de Programação Dinâmicas, leccionada no âmbito do Mestrado em Engenharia de Segurança Informática no Instituto Politécnico de Beja. Pretende-se uma aplicação que permita verificar quais as portas abertas numa máquina na rede local, verificar quais as ligações estabelecidas localmente e efectuar o tratamento de ficheiros de log de firewall. A aplicação deve permitir exportar os dados em CSV e PDF. É possivel identificar o local geográfico da origem dos ips obtidos nas várias funcionalidades.

Index Terms—Python, SQlite3, portscan, conscan, logscan, encription

1 Introdução

O Trabalho efectuado abrange alguns pontos de verificação da rede informática com vista à identificação e detecção de vulnerabilidades na segurança da rede.
Desde logo a identificação de portos abertos nos dispositivos ligados na rede local permite actuar proactivamente na verificação da necessidade da existência de determinados serviços activos.
A análise dos logs produzidos pela firewall ufw poderá identificar possíveis intrusões e a visualização da localização geográfica da origem dos acessos externos pode dar a ideia da distribuição geográfica de potenciais ataques maliciosos, principalmente se os acessos forem persistentes.

2 Estrutura de Ficheiros

A aplicação elaborada é constituída por um conjunto de sete ficheiros:

- scanner.py
- classes.py
- NmapScan.py
- LogScan.py
- ConScan.py
- Export.py
- GeoLiteCity.dat

A aplicação arranca quando é executado o programa $scanner.py$. No ficheiro classes.py residem as classes que através do $alchemy$[1] criam ou apenas fazem a ligação às tabelas na base de dados. Também se encontra aí a classe que efectua a ligação cifragem e decifragem da BD. Os ficheiros $NmapScan.py$, $LogScan.py$, $ConScan.py$ e $Export.py$ contêm classes com o mesmo nome, através dos quais se podem executar as operações da aplicação. Para que a geoferenciação dos endereços ip funcione é necessária a utilização do ficheiro $GeoLiteCity.py$

3 Autenticação e confidencialidade

Não existe nenhum impedimento para qualquer utilizador do programa, no entanto os dados gerados por cada um apenas a ele pertencem e só podem ser acedidos através da disponibilização do username e password.
Sempre que o programa é executado, é necessário indicar o nome de utilizador. Posteriomente o sistema solicita a respectiva password através do módulo getpass [3] que possibilita a inserção da mesma sem que esta seja apresentada no ecran.
São gerados os hash $md5$ do username e da password que após de aplicada a operação XOR: entre eles dá origem ao nome da base de dados para o utilizador com a password actuais.
Quando o programa termina, este cifra a base de dados com $AES - CBC - 256$, utilizando como chave o hash $md5$ da password completada com o byte $\0$[1] até ter a dimensão da chave necessária, neste caso usamos 256 bytes.
Ao iniciar o programa pode ter dois comportamentos. Ou cria uma nova base de dados no caso desta não existir, ou decifra a correspondente ao login efectuado. único para cada login

Listing 1. Gerar o nome da base de dados

```
u = hashlib.md5()
u.update(username)
self.username = u.hexdigest()
p = hashlib.md5()
p.update(password)
self.password = p.hexdigest()
db = ''.join(chr(ord(a) ^ ord(b))
   for a,b in zip(self.username,
   self.password))
m = hashlib.md5()
m.update(db)
self.hash = m.hexdigest()
self.db_name = '%s.db' % self.hash
```

1. Este aspecto precisa ser melhorado

Listing 2. Cifragem da base de dados

```
def encrypt(self, message, key, key_size=256):
  message = self.pad(message)
  iv = Random.new().read(AES.block_size)
  cipher = AES.new(key, AES.MODE_CBC, iv)
  return iv + cipher.encrypt(message)

def pad(self, s):
  return s + b"\0" *
    (AES.block_size - len(s)
      % AES.block_size)
```

4 Base de dados

O sistema de gestão de base de dados utilizado é a biblioteca SQLite3 [2]. A ligação entre este e o Python é feito através do toolkit *Alchemy*.
Foram criadas classes que herdam uma *declarativebase* [4]. Desta forma as classes ficam directamente mapeadas às tabelas correspondentes na base de dados.
As classes criadas foram:

- IP(id, ip, country, country_name, lon, lat)
- LogScanDB(id, time, fk ipsrc_id)
- ConScanDB(id, local_port, remote_port, time, fk remote_id_id)
- NmapScanDB(id, port, time, protocol, fk ip_id)

Através do toolkit *Alchemy* é possível a inserção de registos na base de dados apenas criando um objecto da classe pretendida. Para tal é necessário criar uma sessão ligada à base de dados do login actual. Após a criação dos objectos necessários para a inserção dos registos na base de dados, estes podem ser guardados efectuando commit.

Listing 3. Exemplo de inserção de registos em sqlalchemy

```
engine = create_engine('sqlite:///%s'
  % self.db_name)
self.base.metadata.bind = engine
DBSession = sessionmaker(bind=engine)
self.session = DBSession()
one_ip = IP(ip='127.0.0.1',
  country='PT',
  country_name='Portugal',
  lat=None, lon=None)
another_ip = IP('192.168.0.1', 'PT',
  'Portugal', None, None)
self.session.add(one_ip)
self.session.add(another_ip)
self.session.commit()
```

5 Portscan

Para verificar se uma ou mais portas estão abertas numa máquina a aplicação utiliza a biblioteca para python *python − nmap*[5]. Esta biblioteca permite facilmente manipular resultados do conhecido nmap[**?**] e é uma boa ferramenta para administradores de sistemas, dando a):possibilidade de serem criadas tarefas e relatórios automatizados.
A cada endereço de ip indicado no scan é também feita uma tentativa de geolocalização do mesmo com auxílio do *GeoIP City Database* [7]. Neste trabalho apenas é aproveitado o código do país, nome do país, latitude e longitude.
Cada IP envolvido é gravado ou associado a um registo na base de dados e cada porta encontrada aberta é também guardada sendo associada ao respectivo endereço de ip.
Na Listing 4 é apresentado o código principal desta funcionalidade, sendo omitidas as impressões feitas ao longo da sua execução.

Listing 4. PortScan

```
nm = nmap.PortScanner()
  nm.scan(self.ip, self.port)
  now = datetime.now()
  for host in nm.all_hosts():
    try:
      gi = GeoIP.open('GeoLiteCity.dat',
        GeoIP.GEOIP_STANDARD)
      geo = gi.record_by_addr(host)
      country_=geo['country_code']
      country_name_=geo['country_name']
      lon_=geo['longitude']
      lat_=geo['latitude']
    except:
      country_ = None
      country_name_ = None
      lon_ = None
      lat_ = None
    finally:
      new_ip = IP(ip=host,
      country=country_,
      country_name=country_name_,
      lon=lon_,
      lat=lat_)
    ip_address = self.session.query(
      IP).filter_by(  ip=host).first()

    if ip_address == None:
      self.session.add(new_ip)
      ip_address = new_ip

    for proto in nm[host].all_protocols():
      if proto in ['tcp', 'udp']:
      lport = nm[host][proto].keys()
      lport.sort()
      for p in lport:
        nmp = NmapScanDB(port=p,
        time=now,
        protocol=proto,
        ip=ip_address)
  self.session.add(nmp)
```

```
    self.session.commit()
```

Listing 5. Georeferenciação de um IP

```
gi = GeoIP.open('GeoLiteCity.dat',
  GeoIP.GEOIP_STANDARD)
geo = gi.record_by_addr(address)
country_ = geo['country_code']
country_name_ = geo['country_name']
lon_ = geo['longitude']
lat_ = geo['latitude']
new_ip = IP(ip=address, country=country_,
  country_name=country_name_,
  lon=lon_, lat=lat_)
```

6 CONSCAN

Esta função do programa centra a sua utilização nas bibliotecas *psutil* [8] e *socket* [9]. A aplicação começa por obter os processos que estão a ser executados localmente, filtrando-os por conexões do tipo *inet*. De seguida tenta obter a georeferencia do ip encontrado e armazena na base de dados as informações do ip, bem como porta local e remota, hora da ligação, o status da mesma, e qual o nome e id do processo responsável pela conexão. A execução desta funcionalidade apenas é conseguida caso o utilizador tenha permissões de root.

Listing 6. Verificação de conexões locais

```
AF_INET6 = getattr(socket, 'AF_INET6',
  object())
proto_map = {(AF_INET, SOCK_STREAM)
: 'TCP',
  (AF_INET6, SOCK_STREAM) : 'TCP6',
  (AF_INET, SOCK_DGRAM)   : 'UDP',
  (AF_INET6, SOCK_DGRAM)  : 'UDP6'}
for p in psutil.process_iter():
  try:
    program = p.name
    con = p.get_connections(kind='inet')
    for c in con:
            if c.raddr:
        # obter a georeferenciacao
        new_ip = IP(
          ip=c.raddr[0],
          country=country_,
          country_name=country_name_,
          lon=lon_, lat=lat_)
        ip = self.session.query(
          IP).filter_by(
          ip=c.raddr[0]).first()
        if ip == None:
          self.session.add(new_ip)
          ip = new_ip
        con = ConScanDB(
          local_port=c.laddr[1],
          remote_port=c.raddr[1],
          time=datetime.now(),
          remote_ip=ip)
          self.session.add(con)
  except:
    pass
self.session.commit()
```

7 LOG SCAN

Outra das funcionalidades da ferramenta é a capacidade de armazenar na base de dados os eventos registados no log de firewall, neste caso está apenas preparada para logs da firewall ufw [10] O programa percorre todas as linhas do log tentando, como noutros casos, obter a georeferenciação do ip, e armazena os dados.

Listing 7. Armazenamento de eventos de log

```
for line in logfile.readlines():
  if not re.search("SRC=192", line)
    and not re.search("SRC=0", line)
    and not re.search("SRC=172", line):
  lista = line.split("SRC=")
  ip_src = lista[1].split(' ')[0]
  lineMonth = str(line)
  dataT = lineMonth[:15]
  device = line.split("IN=")
  dInterface = device[1].split(' ')[0]
  if (str(dInterface) ==""):
    device = line.split("OUT=")
    dInterface = device[1].split(' ')[0]
    event = line.split("IN= ")
    eventSrc = event[1].split('=')[0]
  else:
    eventSrc = "IN"
    proto=line.split("PROTO=")
    protoInf=proto[1].split(' ')[0]
  try:
    if len(ip_src)<=15:
      spt=line.split("SPT=")
      sptPort=spt[1].split(' ')[0]
      dpt=line.split("DPT=")
      dptPort=dpt[1].split(' ')[0]
      ttlinf=line.split("TTL=")
      ttlData=str(ttlinf[1].split(' ')[0])
    else:
      continue
    # ... verificar geoip e se ip existe ...
    log = LogScanDB(
      time = datetime.strptime(
        dataT,
        "%b %d %H:%M:%S"),
      ipsrc = ip_address,
      event_src = eventSrc,
      device = dInterface,
      protocol = protoInf,
      ttl = ttlData,
      src_port = sptPort,
      dst_port = dptPort)
    self.session.add(log)
  except Exception as e:
```

```
        print e
```

8 EXPORTAR DADOS

É possível exportar o conteúdo da base de dados para análise. Visto que a base de dados se encontra cifrada, podendo existir a necessidade de a obter na integra foi adicionada a possibilidade de se obter uma cópia da mesma já decifrada. O método utilizado é semelhante ao da cifra já explicado anteriormente.
Além de poder exportar toda a base de dados também há a possibilidade de exportar os dados em CSV ou PDF.

8.1 CSV

O ficheiro CSV é gerado com base no módulo de python *CSV* [11]. O programa faz a pesquisa na base de dados por todos os registos e adiciona os resultados, agrupados por tipo. A listagem 8 mostra um exemplo do query à base de dados com o alchemy que também é usado para o tipo PDF.

Listing 8. Exportar CSV

```
s = csv.writer(
  open(filename, 'wb'),
  delimiter=';',
  quotechar='\x22',
  quoting=csv.QUOTE_MINIMAL)
#Query    base de dados com alchemy
conscans =
  self.session.query(ConScanDB).all()
s.writerow(["LOCAL CONNECTIONS"])
s.writerow(["[Time]"]
  + ["[Local Port]"]
  + ["[Remote IP]"]
  + ["[Remote Port]"])
 for line in conscans:
  s.writerow([line.time]
    + [line.local_port]
            + [line.remote_ip.ip]
    + [line.remote_port])
```

8.2 PDF

A criação dos pdfs utiliza a biblioteca Reportlab [12]. O documento é criado com tabelas que alojam os dados armazenados. Os endereços de IP apresentados podem ser clicados pois têm um link para o google maps centrado a sua geo localização.

Listing 9. Exportar PDF

```
doc = SimpleDocTemplate(filename,
  pagesize=landscape(A4))
elements = []
styles=getSampleStyleSheet()
styleN = styles["Normal"]
# Leitura de todos os conscans
scans =
  self.session.query(ConScanDB).all()
data = [["LOCAL CONNECTIONS"]]
elements.append(self.drawTable(data, 1))
data = [["Time","Local Port",
  "Remote IP", "Remote Port"]]
for line in scans:
  data.append([line.time,
  line.local_port,
  Paragraph("<a href='
    https://maps.google.com/
    maps?q=loc:%s,%s'>%s</a>"
    % (line.ipsrc.lat, line.ipsrc.lon,
    line.ipsrc.ip),
  styles["Normal"]),
  line.remote_port])
elements.append(self.drawTable(data))
#  ....
doc.build(elements)
print "File saved to %s" % filename

def drawTable(self, data, blank = 0):
  result = Table(data, repeatRows=1)
  result.hAlign = 'LEFT'
  if blank == 1:
    tblStyle = TableStyle(
      [('TEXTCOLOR',
        (0,0), (-1,-1),
        colors.black),
        ('VALIGN',(0,0),
        (-1,-1),'TOP')])
  else:
    tblStyle = TableStyle([
      ('TEXTCOLOR',(0,0),
        (-1,-1),colors.black),
      ('VALIGN',(0,0),
        (-1,-1),'TOP'),
      ('LINEBELOW',(0,0),
        (-1,-1),1,colors.black),
      ('INNERGRID',(0,0),
        (-1,-1),1,colors.black),
        ('BOX',(0,0),(-1,-1),
        1,colors.black)])
    tblStyle.add('BACKGROUND',(0,0),
      (-1,-1),colors.lightblue)
    tblStyle.add('BACKGROUND',(0,1),
      (-1,-1),colors.white)
  result.setStyle(tblStyle)
  return result
```

9 UTILIZAÇÃO

A utilização do programa basea-se em linha de comandos através da passagem de alguns argumentos. O tratamento desses argumentos é feito com auxílio do módulo de python Argparse [13].

Listing 10. Tratamento dos argumentos linha de comandos

```
parser = argparse.ArgumentParser()
```

```
parser.add_argument("-u", "--username",
  required=True)
parser.add_argument("-portscan",
  nargs=2,
  metavar=('ip', 'ports'),
  required=False,
          help="Perform a portscan")
parser.add_argument("-conscan",
  action='store_true',
  required=False,
  help="Scan for local connections")
parser.add_argument("-logscan",
  nargs=1,
  metavar=('file'),
  required=False,
  help="Store log cons into database")
parser.add_argument("-export",
  nargs=2,
  metavar=('filename', 'filetype'),
  required=False,
  help="export database [db, csv, pdf]")
parser.add_argument("-delete",
  action="store_true",
  required=False,
  help="Delete database")
args = parser.parse_args()
```

O programa é executado através do ficheiro.$/scanner.py - u < username >$ e os argumentos disponíveis para a execução do programa são os apresentados abaixo, em todos eles é solicitada a password para a execução:

- $-conscan$
  Verificação de conexões estabelecidas na máquina local
- $-portscan < ip >< ports >$
  Efectuar um portscan a uma máquina
- $-logscan < logfile >$
  Tratamento dos dados de um ficheiro de log de firewall
- $-export < filename >< filetype >$
  Exportar os dados
- $-delete$
  Eliminar (limpar) a base de dados do utilizador activo

10 Conclusão

O trabalho foi realizado totalmente na linguagem de programação Python, utilizando-se bibliotecas standard e outras que pela sua natureza se mostraram muito eficazes neste projecto. As principais bibliotecas utilizadas foram a python-nmap que fornece um interface com o programa nmap, a python-psutil que disponibiliza funções sobre os processos.
Optamos por registar numa base de dados SQLite3 todas as informações recolhidas numa fase de inventariação de dados e posteriormente a disponibilização da informação registada para vários tipos de saída, tais como seja em formato db (SQLite), PDF ou CSV. Para maior segurança a base de dados é cifrada a partir dos dados do utilizador e respectiva senha para evitar acessos directos aos dados e assim existir maior segurança. Para interface com a base de dados optamos por utilizar a biblioteca python-sqlalchemy que permite interagir de forma simples com a maioria das base de dados entre as quais a SQLite utilizada neste projecto.
Para a disponibilização da informação em formato PDF utilizou-se a biblioteca python-reportlab a qual permite a criação de relatórios em formato PDF com muitas funcionalidades e potencialidades interessantes. Também foi utilizado a biblioteca python-geoip que permite a localização geográfica de um IP público. A forma prática de como se fez a utilização desta biblioteca foi através das referências da latitude e longitude obtidas para cada IP, ser incluída num hyperlink em cada IP listado no formato PDF, para permitir ao utilizador aceder à localização geográfica via Google Maps.
Como melhoria ao projecto salientamos a possibilidade de através da biblioteca python-reportlabs serem criados gráficos e tabelas com dados estatísticos nos relatórios em formato PDF.

References

[1] Michael Bayer. SQLAlchemy. Disponível no link http://www.sqlalchemy.org. Consultado em Março de 2014
[2] SQLite, Disponível no link https://www.sqlite.org. Consultado em Março de 2014
[3] Python. getpass — Portable password input. Disponível no link: http://docs.python.org/2/library/getpass.html. Consultado em Março de 2014
[4] Declarative. SQLAlchemy 0.9 Documentation. Disponível no link: http://docs.sqlalchemy.org/en/rel_0_9/orm/extensions/ declarative.html. Consultado em Março de 2014
[5] Python. python-nmap 0.3.3. Disponível no link: https://pypi.python.org/pypi/python-nmap/0.3.3. Consultado em Março de 2014
[6] Nmap - Free Security Scanner For Network Exploration & Security Audits. Disponível no link: http://nmap.org/. Consultado em Março de 2014
[7] Maxmind Developer Site. GeoIP City Database. Disponível no link: http://dev.maxmind.com/geoip/legacy/install/city/. Consultado em Março de 2014
[8] Python. psutil 2.0.0. Disponível no link: https://pypi.python.org/pypi/psutil/. Consultado em Março de 2014
[9] Python. socket — Low-level networking interface. Disponível no link: http://docs.python.org/2/library/socket.html. Consultado em Março de 2014
[10] Ubuntu. UFW - Uncomplicated Firewall. Disponível no link: https://help.ubuntu.com/community/UFW. Consultado em Março de 2014
[11] Python. csv — CSV File Reading and Writing. Disponível no link: http://docs.python.org/2/library/csv.html. Consultado em Março de 2014
[12] ReportLab: Open Source Python Libraries for PDF creation. Disponível no link: http://www.reportlab.com/software/opensource/. Consultado em Março de 2014
[13] Python. argparse — Parser for command-line options, arguments and sub-commands. Disponível no link: http://docs.python.org/3.4/library/argparse.html. Consultado em Março de 2014