Aplicação de Segurança Informática

Fonte, *Luís, - 6043*

***Abstract*—Este trabalho representa a avaliação da disciplina Linguagens de Programação Dinâmicas, lecionada no âmbito do Mestrado em Engenharia de Segurança Informática no Instituto Politécnico de Beja. É apresentada uma aplicação que permite verificar quais as portas abertas numa máquina na rede local, verificar quais as ligações estabelecidas localmente e efetuar o tratamento de ficheiros de log de firewall. A aplicação permite exportar os dados em CSV e criar gráficos, bem como mapas, no seguimento do tratamento dos logs de Firewall, tornando possivel identificar o local geográfico. Sendo um requisito para a fase de recurso, foram adicionadas algumas funcionalidades extra (três ao todo), nomeadamente DNS Lookup, WHOIS e Reverse Lookup.**

***Index Terms*—Python, segurança, portscan, conexões, firewall, md5, base de dados, sqlite3, dns lookup, whois, reverse lookup.**

I. Introdução

O trabalho abrange alguns pontos de verificação da rede informática com vista à identificação e deteção de eventuais vulnerabilidades ao nível de segurança da própria rede. Para o desenvolvimento foi utilizado o sistema operativo Ubuntu 14.04 LTS, juntamente com o editor de texto default do ambiente de desenvolvimento integrado IDLE [1], que está presente no package de instalação da linguagem Python [2], que é a linguagem utilizada nesta aplicação.

II. Sistema de Controlo de Versões

O sistema de controlo de versões utilizado foi o Git [3], recorrendo ao serviço GitHub [4] como provedor de serviço e repositório. O nome do projeto criado é lpd14156043 [5].

III. Estrutura de Ficheiros

A aplicação é constituída pelo seguinte conjunto de ficheiros:

- connections.py
- db_userLogin.py
- generate_graph.py
- generate_maps.py
- main_exe.py
- portscan.py
- process_firewall_log.py

A aplicação é iniciada pelo ficheiro main_exe.py, apresentando um menu com 7 opções principais. Ainda antes da apresentação do menu é pedida a autenticação do utilizador no sistema. No momento inicial apenas existe o utilizador "fonte", que tem como password "mesi2015".

```
if __name__ == "__main__":

    ok = True
        while ok:
            user = raw_input("Username: ")
            passw = raw_input("Password: ")
            if db_userLogin.login(user, passw):
                print "\nSucess."
                break
            else:
                print "\nError, try again.\n"
            os.system("clear")
```

É chamada a função login no ficheiro db_userLogin.py:

```
    def login(username, password):
        """
        Do login.

        @False - returns false if login fails
        @True - returns true if login succeded
        """
        u = hashlib.md5(str.encode(username)).hexdigest()
        p = hashlib.md5(str.encode(password)).hexdigest()
        con = sqlite3.connect("users.db")
        cursor = con.cursor()
        cursor.execute("SELECT * FROM users WHERE user=?
and pass=?", (u,p))
        user = cursor.fetchone()
        if user == None:
            return False
        else:
            return True
        pass
```

É realizado um encode do username e password, calculando o hash *md5* para uma maior segurança na autenticação.

A criação e gestão da base de dados é feita com sqlite3.

Após autenticação no sistema, é apresentado então o Menu:

1. Portscan
2. Connections
3. Firewall Log Processing
4. Add user to DB
5. DNS Lookup
6. WhoIS
7. Reverse Lookup

IV. Funcionalidades da Aplicação

A. *Portscan*

Para a verificação de portas abertas numa máquina, foi utilizada a biblioteca para Python *python-nmap*[6]. Esta funcionalidade requer a introdução do endereço IP por parte do utilizador, e os portos que pretende conhecer sobre o seu estado.

```
import nmap

def portscan():
    """
    Does the portscan to a target IP and and ethernet ports
defined by user
    """
    addresses = str(raw_input('IP to scan: '))
    ports = str(raw_input('Ports to scan: '))

    scanner = nmap.PortScanner()
    try:
        scanner = nmap.PortScanner()
        scanner.scan(addresses, ports)

        for targetHost in scanner.all_hosts():
            if scanner[targetHost].state() == 'up':
                print targetHost + ' is up \n'
                for targetPort in scanner[targetHost]['tcp']:
                    print 'Port ' + str(targetPort) + '/tcp ' +
scanner[targetHost]['tcp'][int(targetPort)]['name'] + ' is ' +
scanner[targetHost]['tcp'][int(targetPort)]['state']
                    print ''
                    pass
                pass
            pass
    except Exception, e:
        print '[-] Something bad happened: ' + str(e)
        pass
```

B. *Connections*

Esta funcionalidade permite obter quais as conexões existentes na nossa máquina local, bem como o protocolo em utilização, o porto, e o estado.

```
def getConnections():
    """
    Returns array with connections
    Calls a method that "asks" to system for connections
established
    Adds the top row with titles to the connection data

    @connections_array - returns array with logs
    """
    con = processConnections()
    connections_array = []
    for x in con:
        c = []
        treatcon = x[0].split(" ")
        treatcon = filter(lambda x: x != "", treatcon)
        c.append(treatcon[0])#PROTOCOL
        a = treatcon[3].split(":")
        c.append(a[0])#LOCALUSER
        c.append(a[1])#PORT
        a = treatcon[4].split(":")
        c.append(a[0])#CONNECTION
        c.append(a[1])#PROTO
        c.append(treatcon[5].strip("\n"))
        connections_array.append(c)
        pass
    return connections_array
```

Após recolhida a informação com a função *getConnections*, a informação é processada, sob a forma de texto, e tratada para ser apresentada de forma mais "simpática" aos olhos do utilizador.
A aplicação permite ainda guardar a informação obtida num ficheiro CSV, através da biblioteca *csv*, e gerar gráficos de acordo com a informação desejada.

```
    def makeGraf(in_data, column, title, label):
        """
        Builds graph, from array with information.

        var in_data - information array
        var column - column
        var title - graph title
        var lable - axis legend
        """
        data = [x[column] for x in in_data]
        data_distint = list(set(data))
        counts = []
        for ip in data_distint:
            count = 0
            for x in data:
                if x == ip:
                    count+=1
            counts.append(count)

        plotGraf(title, data_distint, 0, 0, counts, label)
    pass
```

C. Firewall Log Processing

Esta funcionalidade processa ficheiros de log das fireal (ufw) com o objetivo de obter informação sobre tentativas de acesso vindas de fora, ou seja, tentativas de intrusão. É feita ainda a georreferenciação dos endereços ip. Esta funcionalidade tem 4 sub funcionalidades:

- Show logs – Apresenta no terminal as informações georreferenciadas sobre os endereços IP.
- Generate Map – Gera um mapa "firewall_log_map.htm" utilizando a biblioteca *pymaps* com a referenciação dos endereços IP no próprio mapa.
- Generate CSV – Guarda todos os registos obtidos, num ficheiro CSV.
- Generate Graph – Gera um gráfico com opções definidas pelo utilizador.

```
def do_FirewallLogProcessing():
    """
    Imports file with code to do the firewall loga processing
    """
    import process_firewall_log
    process_firewall_log.showLogs()
    pass
```

Extração dos logs do ficheiro…

```
def getLogs():
    """
    Reads log file from firewall

    @logsArray - Returns array with logs from file
    """
    logFile_open = open(LOG_FILE, "r") #log file from uncomplicated firewall
    logsProcess = [] #Array to store logs separated by spaces
    for line in logFile_open:
        logsProcess.append(line.split(" ")) #append with split done
    logsArray = [] #array to store file logs
    for log in logsProcess:
        date_log = log[0]+log[1]+log[2]
        for x in log:
            if x.startswith("IN="):
                in_log = x.strip("IN=")
            elif x.startswith("OUT="):
                out_log = x.strip("OUT=")
            elif x.startswith("SRC="):
                src_log = x.strip("SRC=")
            elif x.startswith("DST="):
                dst_log = x.strip("DST=")
            elif x.startswith("PROTO="):
                proto_log = x[6:]
            elif x.startswith("SPT="):
                port_log = x[4:]


logsArray.append([date_log,in_log,out_log,src_log,dst_log,proto_log,port_log])#append all data formated
    return logsArray
    pass
```

Determinar a localização geográfica dos endereços IP através do *pygeoip* [7] com o ficheiro "GeoIP.dat"…

```
def getLogsLocation():
    """
    Adds geoloction to each log row

    @logs_final - Returns logs array complete and with geo location
    """
    import pygeoip
    GI = pygeoip.GeoIP("GeoIP.dat", pygeoip.MEMORY_CACHE) #connections with geoip file
    logs = getLogs() #calls method that extrat logs from firewall file
    logs_final = [] #array to store the logs
    for log in logs:
        try:
            info = GI.record_by_addr(log[3])

logs_final.append([log[0],log[1],log[2],log[3],log[4],log[5],log[6],info["continent"],info["country_name"],info["city"],info["longitude"],info["latitude"]])
        except:
            pass
    return logs_final
    pass
```

D. Add user to DB

Esta funcionalidade permite adicionar mais utilizadores à base de dados existente, além do utilizador administrador, como já referido.

```
def user_add(username, password):
    """
    Add user to db.
    """
    u = hashlib.md5(username).hexdigest()
    p = hashlib.md5(password).hexdigest()
    con = sqlite3.connect("users.db")
    cursor = con.cursor()
    cursor.execute("INSERT INTO users VALUES (null, ?, ?)", (u,p))
    con.commit()
    pass
```

E. DNS Lookup

As informações DNS (Doman Name System) são muito úteis na área da segurança informática pois facilmente dão informações sobre determinado IP ou nome de domínio de forma a correlacionar com um eventual log detetado pela firewall com a sua proveniência. Esta funcionalidade requer a introdução do endereço IP por parte do utilizador.

```
def do_nslookup():
    """
    Do DNS Lookup with IP address given by user

    var address_to_check - stores ip given by user
    var ais - socket function to search dns info
    var ip_list - list with ips found
    """
    import socket #https://docs.python.org/2/library/socket.html
    ip_list = []
    address_to_check = raw_input("Address to check: ")
    ais = socket.getaddrinfo(address_to_check,0,0,0,0)
    for result in ais:
        ip_list.append(result[-1][0])
    ip_list = list(set(ip_list))
    print ip_list
    pass
```

F. WhoIS

WhoIS é um serviço que normalmente corre na porta 43 e que permite consultar informações de contacto e de entidades na Internet, tal como em que nome e morada está registado um determinado endereço IP. Mais uma vez, em conjunto com os IP registados pelos log da firewall, facilmente se pode obter informação sobre o eventual autor duma tentativa de ataque. Esta funcionalidade requer a introdução dum endereço IP a pesquisar, por parte do utilizador.

```
def do_whois():
    """
    Do Whois search aboout ip address registry

    var ip_to_search - stores ip given by user
    var obj - ip whois function with ip to search
    """
    import ipwhois # from https://pypi.python.org/pypi/ipwhois
    from ipwhois import IPWhois
    import pprint #https://docs.python.org/2/library/pprint.html
    ip_to_search = str(raw_input("IP address to search for: "))
    print "\n"
    obj = IPWhois(ip_to_search)
    try:
        pprint.pprint(obj.lookup_rws())
    except:
        print "\nError."
        raw_input("\nClick Enter to continue...")
        pass
    raw_input("\nClick Enter to continue...")
    pass
```

G. Reverse Lookup

Do mesmo género do DNS Lookup, mas de forma contrária no output. Esta funcionalidade permite determinar um endereço IP através do nome de domínio. Foi utilizada a biblioteca *socket*.

```
def do_reverseLookup():
    """
    Gets hostname by IP address

    var address - ip address to searchS
    """
    import socket

    try:
        print "-Reverse Lookup-\n"
        address = raw_input("Enter the Domain name: ")
        print "\n"
        print socket.gethostbyaddr(address)
        print "\n"
        raw_input("\nClick Enter to continue...")
        pass
    except:
        print "\nError."
        raw_input("\nClick Enter to continue...")
        pass
    pass
```

V. Main Menu

```
------------------------------------------------------------
---------- Network Security Application -----------
------------------------------------------------------------
---------------- Author: Luis Fonte -------------------
---------------- LPD MESI IPBEJA ----------------
------------------------------------------------------------

--MENU--
 1 - Portscan
 2 - Connections
 3 - Firewall Log Processing
 4 - Add user to DB
 5 - DNS Lookup
 6 - WhoIS
 7 - Reverse Lookup
...
 0 - Exit Program
```

References

[1] Python Documentation – IDLE – Disponível em: https://docs.python.org/2/library/idle.html
[2] Python Software Foundation – Disponível em: https://www.python.org/
[3] Git (Open Source distributed version control system) – Disponível em: *https://git-scm.com*
[4] Git Hub (Online Project Hosting) – Disponível em: *https://github.com*
[5] Git Hub Project Repository (Luís Fonte) – Disponível em: *https://github.com/LFonte/lpd14156043*
[6] *Python-nmap (nmap python lybrary) – Disponível em: https://pypi.python.org/pypi/python-nmap*
[7] *Pygeoip (python lybrary for geographic localization of IP addresses) – Disponível em: https://pypi.python.org/pypi/pygeoip*