ScriptPentestOsint

Ce script python a pour but d'exécuter une succession de commandes systèmes linux sur une machine virtuelle dans le but de récupérer des données propres d'une entreprise à partir de son nom de domaine.

Ici l'ingénieur entre en début de programme le nom de domaine de l'entreprise cliente (ex : google.com), le script va ainsi enregistrer dans une bdd MongoDB toutes les informations recuieillies grâce à une méthode OSINT passive (donc l'approche de collecte d'informations reposant sur l'analyse et l'interprétation des données accessibles au public, sans engager d'activités intrusives ou illégales), ces données vont ainsi être classées par tables représentées par des fonctions et dont le but est d'être traitées une par une à la suite :

- partie dns : Nom FQDN : dig google.com | awk '/QUESTION SECTION:/{getline; print $1}'

  Type : dig google.com | awk '/ANSWER SECTION:/{getline; print "Type :", $3, $4}'

  Adresse ip : dig google.com | awk '/ANSWER SECTION:/{getline; print "Adresse IP :", $5}'

  As: theHarvester -d axians.com -l 500 -b all -f test.json

  Entreprise : whois google.com | grep "Registrant Organization"

- partie host1 :

  utilisation du modules brute_hosts sur recon-ng dans le but de trouver des hosts et leurs ip à partir de Wordlists disponible dans les secLists

- partie host2 :

  utilisation du modules hackertarget sur recon-ng dans le but de trouver des hosts et leurs ip, cela permet ainsi que les deux modules ne trouvent pas forcément la même chose et donc de collecter davantages de données.

  DANS CES DEUX FONCTIONS, la récolte des données se permet via le module reporting/json qui formate celles-ci dans le but qu'elles soient traitées plus facilement par le script.

- partie Mail :

utilisation de l'outil TheHarvester qui va chercher des adresses email associées au nom de domaine demandé toujours en les formatant dans un fichier json

- partie ports :

  La fonction cmdPorts du script est a modifié car celle-ci utilise nmap (méthode OSINT active...).

- partie Personnes :

  La fonction cmdPeople cherche des informations sur des personnes publiquement reliées à leur entreprise(nom, prénom,mail; ces trois infos ne sont pas forcément tout le temps présentes dans la récolte) à l'aide du module bing_linkedin_cache de recon-ng et les formatent dans un fichier json pour les implenter dans la bdd

- partie Site :

  Location : curl -s -I google.com | grep -i "^location:" | awk '{print $2}'

  Content-type : curl -s -I google.com | grep -i "^content-type:" | awk '{print $2}'

  Cache control : curl -s -I google.com | grep -i "^cache-control:" | awk '{$1=""; print $0}'

  Server : curl -s -I google.com | grep -i "^server:" | awk '{$1=""; print $0}'

  Xframe : curl -s -I google.com | grep -i "^x-frame-options:" | awk '{$1=""; print $0}'

  l'exécution de ces commandes peut très souvent ne pas fonctionnées totalement pour les 3 dernieres mais les commandes sont bonnes donc n'hésitez pas à les faire à part et à remplacer les données à la main dans votre BDD

  NOTE de synthèse :

  ```
  Il est plus intéressant d'exécuter ces fonctions une par une dans le but de gagner du temps car cela peut prendre + d'une demi-heure si on lance tout en même temps.
  ```

  Script :

  ```
  import os

  import subprocess

  import json
  ```

```
import pexpect

import xml.etree.ElementTree as ET

import random

import string

from datetime import datetime

from mongoengine import connect, Document, StringField, IntField, ReferenceField,
ListField




connect('OsintDB', host='192.168.73.130', port=27017)



def generer_id(longueur):

    caracteres = string.ascii_letters + string.digits

    id_generer = ''.join(random.choice(caracteres) for _ in range(longueur))

    return id_generer



longueur_id = 0

while longueur_id < 20 or longueur_id > 30:

    longueur_id = int(input("Entrez un nombre de caractères entre 20 et 30 pour
l'ID : "))



iddd = generer_id(longueur_id)

print("IDenregistré :", iddd)



class Scan(Document):

    datetime = IntField()



class User(Document):

    nom = StringField()
```

```
    idDatas = StringField(required=True)


class NomDomaine(Document):
    fqdn = StringField()
    scans = ListField(ReferenceField(Scan))


class Client(Document):
    name = StringField()
    noms_domaine = ListField(ReferenceField(NomDomaine))


class PartieDNS(Document):
    name = StringField(required=True)
    DNStype = StringField(required=True)
    adressIP = StringField(required=True)
    DNSentreprise = StringField(required=True)
    idDatas = StringField(required=True)
    Scan = ReferenceField(Scan)



class PartieHosts(Document):
    host = StringField(required=True)
    adressiphosts = StringField(required=None)
    module = StringField(required=True)
    idDatas = StringField(required=True)


class PartieHosts2(Document):
    host = StringField(required=True)
    adressiphosts = StringField(required=True)
```

```
    module = StringField(required=True)
    idDatas = StringField(required=True)

class PartieSquatting(Document):
    nomFQDN = StringField(required=True)
    adressipsquatting = StringField(required=True)
    serveurDNS = StringField(required=True)
    Squattingtype= StringField(required=True)
    idDatas = StringField(required=True)

class PartiePorts(Document):
    nomFQDNport = StringField(required=True)
    port = StringField(required=True)
    state = StringField(required=True)
    services = StringField(required=True)
    source = StringField(required=True)
    idDatas = StringField(required=True)

class PartieSite(Document):
    location = StringField(required=None)
    content = StringField(required=None)
    cache_control = StringField(required=None)
    server = StringField(required=None)
    xframe = StringField(required=None)

class PartieEmail(Document):
    addressmail = StringField(required=True)
```

```
    source = StringField(required=True)
    compromission= StringField(required=True)
    occurrence = StringField(required=True)
    idDatas = StringField(required=True)

class PartieFicherpublics(Document):
    url = StringField(required=True)
    typefp = StringField(required=True)
    confidentialite=StringField(required=True)
    idDatas = StringField(required=True)

class PartieName(Document):
    firstname = StringField(required=None)
    middlename = StringField(required=None)
    lastname = StringField(required=None)
    email = StringField(required=None)
    module = StringField(required=None)
    idDatas = StringField(required=True)

class PartieGeneratedmails(Document):
    generatedmails = StringField(required=True)
    idDatas = StringField(required=True)

# executiion cmd reconng
def execute_recon_ng(commands):
    try:
        child = pexpect.spawn('recon-ng')
        child.expect('recon-ng')
```

```
        results = []

        for command in commands:
            child.sendline(command)
            index = child.expect(['recon-ng', pexpect.TIMEOUT], timeout=3000)
            if index == 0:
                results.append(child.before.decode('utf-8'))
            elif index == 1:
                print("Timeout exceeded for command:", command)

        child.sendline('show hosts')
        index = child.expect(['recon-ng', pexpect.TIMEOUT], timeout=3000)
        if index == 0:
            results.append(child.before.decode('utf-8'))
        elif index == 1:
            print("Timeout exceeded for 'show hosts' command")

        child.sendline('exit')
        child.close()

        return results

    except Exception as e:
        print(f"An error occurred: {e}")
        return []

# FONCTION PARTIE DNS EXECUTION CMD
```

```
def cmdDNS(nom_domaine):

    try:

        #  nom FQDN

        resultat_fqdn = subprocess.run(['dig', nom_domaine], capture_output=True,
text=True, check=True)

        fqdn = subprocess.run(['awk', '/QUESTION SECTION:/{getline; print $1}'],
input=resultat_fqdn.stdout, capture_output=True, text=True, check=True)

        fqdn_str = fqdn.stdout.strip()



        # type

        resultat_type = subprocess.run(['dig', nom_domaine], capture_output=True,
text=True, check=True)

        type_info = subprocess.run(['awk', '/ANSWER SECTION:/{getline; print "Type
:", $3, $4}'], input=resultat_type.stdout, capture_output=True, text=True,
check=True)

        type_info_str = type_info.stdout.strip()



        # adresse IP

        resultat_ip = subprocess.run(['dig', nom_domaine], capture_output=True,
text=True, check=True)

        ip = subprocess.run(['awk', '/ANSWER SECTION:/{getline; print $5}'],
input=resultat_ip.stdout, capture_output=True, text=True, check=True)

        ip_str = ip.stdout.strip()



        # whois

        entreprise_info = "pas récupérée"  # par défaut

        whois_result = subprocess.run(['whois', nom_domaine], capture_output=True,
text=True, check=True)



        try:

            entreprise = subprocess.run(['grep', 'Registrant Organization'],
input=whois_result.stdout, capture_output=True, text=True, check=True)
```

```
            # une entreprise a été récupérée ?
            if entreprise.stdout.strip() != "":
                entreprise_info = entreprise.stdout.strip()
        except subprocess.CalledProcessError as e:
            print(f"Erreur lors de la récup infos sur l'entreprise : {e}")

        return fqdn_str, type_info_str, ip_str, entreprise_info

    except subprocess.CalledProcessError as e:
        print(f"Erreur commande dig : {e}")
        return "", "", "", "pas récupérée"

# FONCTION PARTIE Site EXECUTION CMD
def cmdSite(nom_domaine):
    try:
        curl_command = f"curl -s -I {nom_domaine}"

        # Location
        try:
            location_result = subprocess.run(f"{curl_command} | grep -i
'Location:' | awk '{{print $2}}'", shell=True, capture_output=True, text=True,
check=True)
            location_str = location_result.stdout.strip()
        except subprocess.CalledProcessError as e:
            print("Location header not found à faire à la main")
            location_str = "N/A"

        # Content-type
        try:
```

```
            content_type_result = subprocess.run(f"{curl_command} | grep -i 'Content-type:' | awk '{{print $2}}'", shell=True, capture_output=True, text=True, check=True)

            content_type_str = content_type_result.stdout.strip()

        except subprocess.CalledProcessError as e:

            print("Content-Type header not found à faire à la main")

            content_type_str = "N/A"



        # Cache control

        try:

            cache_control_result = subprocess.run(f"{curl_command} | grep -i '^cache-control:' | awk '{{\$1=\"\"; print \$0}}'", shell=True, capture_output=True, text=True, check=True)

            cache_control_str = cache_control_result.stdout.strip()

        except subprocess.CalledProcessError as e:

            print("Cache-Control header not found à faire à la main")

            cache_control_str = "N/A"



        # Server

        try:

            server_result = subprocess.run(f"{curl_command} | grep -i 'Server:' | awk '{{\$1=\"\"; print \$0}}'", shell=True, capture_output=True, text=True, check=True)

            server_str = server_result.stdout.strip()

        except subprocess.CalledProcessError as e:

            print("Server header not found à faire à la main")

            server_str = "N/A"



        # X-Frame-Options

        try:
```

```
            x_frame_options_result = subprocess.run(f"{curl_command} | grep -i 'X-frame-options:' | awk '{{\$1=\"\"; print \$0}}'", shell=True, capture_output=True, text=True, check=True)

            x_frame_options_str = x_frame_options_result.stdout.strip()

        except subprocess.CalledProcessError as e:

            print("X-Frame-Options header not found à faire à la main")

            x_frame_options_str = "N/A"



        return location_str, content_type_str, cache_control_str, server_str, x_frame_options_str



    except subprocess.CalledProcessError as e:

        print(f"Erreur lors de l'exécution de la cmd curl : {e}")

        return "", "", "", "", ""



# FONCTION PARTIE HOSTS EXECUTION CMD

def cmdHosts(nom_domaine):

    try:

        print("Starting host part")



        # exécution des commandes dans Recon-ng

        commands_recon_ng = [

            'modules load recon/domains-hosts/brute_hosts',

            'db delete hosts 1-10000',

            'options set SOURCE ' + nom_domaine,

            'options set WORDLIST /usr/share/wordlists/seclists/Discovery/DNS/subdomains-top1million-5000.txt',

            'info',

            'input',

            'run',
```

```
            'modules load reporting/json',
            'run',
            'exit'
        ]

        # résultats de show hosts
        show_hosts_results = execute_recon_ng(commands_recon_ng)

        results_json = '/home/kali/.recon-ng/workspaces/default/results.json'
        with open(results_json, 'r') as results_file:
            results_json_content = results_file.read()

        results_json_parse = json.loads(results_json_content)
        for result in results_json_parse['hosts']:
            partie_host = PartieHosts(
                host=result['host'],
                adressiphosts=result['ip_address'],
                module=result['module'],
                idDatas=iddd
            )

            partie_host.save()

        return show_hosts_results, results_json_content

    except Exception as e:
        print(f"Une erreur s'est produite : {e}")
        return "", "pas récupérée",""
```

```
# FONCTION PARTIE HOSTS2 EXECUTION CMD

def cmdHosts2(nom_domaine):

    try:

        print("Starting host2 part")


        # exécution des commandes dans Recon-ng

        commands_recon_ng = [

            'modules load hackertarget',

            'db delete hosts 1-10000',

            'options set SOURCE ' + nom_domaine,

            'info',

            'input',

            'run',

            'modules load reporting/json',

            'run',

            'exit'

        ]


        # résultats de show hosts

        show_hosts_results2 = execute_recon_ng(commands_recon_ng)


        results_json = '/home/kali/.recon-ng/workspaces/default/results.json'

        with open(results_json, 'r') as results_file:

            results_json_content2 = results_file.read()


        results_json_parse = json.loads(results_json_content2)

        for result in results_json_parse['hosts']:
```

```
            partie_host2 = PartieHosts2(
                host=result['host'],
                adressiphosts=result['ip_address'],
                module=result['module'],
                idDatas=iddd
            )
            partie_host2.save()

        return show_hosts_results2, results_json_content2

    except Exception as e:
        print(f"Une erreur s'est produite : {e}")
        return "", ""

# FONCTION PARTIE MAIL EXECUTION CMD
def cmdMail(nom_domaine):
    try:
        # cmd theharvester
        command = f'theHarvester -d {nom_domaine} -l 500 -b all -f test.json'
        subprocess.run(command, shell=True, check=True)

        print("TheHarvester fonctionne")

        result_file_path = 'test.json'
        with open(result_file_path, 'r') as result_file:
            test_json = result_file.read()
            result_json_parse = json.loads(test_json)
```

```
            for email in result_json_parse["hosts"]:
                partie_email = PartieEmail(
                    addressmail=email,
                    source="TheHarvester",
                    compromission="",
                    occurrence="1",
                    idDatas=iddd
                )
                partie_email.save()
                print(f"Email enregistré: {email}")
    except subprocess.CalledProcessError as e:
        print(f"erreur TheHarvester : {e}")



# FONCTION PARTIE PORTS EXECUTION CMD
def cmdPorts(nom_domaine):
    try:
        resultat_nmap = subprocess.run(['nmap', nom_domaine], capture_output=True,
text=True, check=True)
        nmap_output = resultat_nmap.stdout


        for line in nmap_output.split('\n'):
            if '/' in line:
                parts = line.split('/')
                if len(parts) == 3:
                    port, state, services = parts


                    partie_port = PartiePorts(
                        nomFQDNport=nom_domaine,
```

```
                        port=port.strip(),
                        state=state.strip(),
                        services=services.strip(),
                        source= "nmap.org",
                        idDatas=iddd
                    )
                    partie_port.save()

    except subprocess.CalledProcessError as e:
        print(f"Erreur lors de l'exécution de nmap : {e}")

# FONCTION PARTIE PERSONNES EXECUTION CMD
def cmdPeople(nom_domaine):
    try:
        print("Starting people part")

        # exécution des commandes dans Recon-ng
        commands_recon_ng = [
            'modules load recon/companies-contacts/bing_linkedin_cache',
            'db delete hosts 1-1000',
            'db delete contacts 1-1000',
            'options set SOURCE ' + nom_domaine,
            'input',
            'run',
            'modules load reporting/json',
            'run',
            'exit'
        ]
```

```
        # résultats de show hosts
        show_people_results = execute_recon_ng(commands_recon_ng)

        results_json = '/home/kali/.recon-ng/workspaces/default/results.json'
        with open(results_json, 'r') as results_file:
            results_json_content3 = results_file.read()

        results_json_parse = json.loads(results_json_content3)
        for result in results_json_parse['contacts']:
            partie_people = PartieName(
                firstname=result['first_name'],
                middlename=result['middle_name'],
                lastname=result['last_name'],
                email=result['email'],
                module=result['module'],
                idDatas=iddd
            )
            partie_people.save()

        return show_people_results, results_json_content3

    except Exception as e:
        print(f"Une erreur s'est produite : {e}")
        return "", "","", "", ""

# PROGRAMME EXECUTION DES FONCTIONS
def executioncmd():
```

```
try:
    '''nom_client = input("Entrez le nom du client : ")
    if Client.objects(name = nom_client).count() == 1:
        client = Client.objects(name = nom_client)[0]
    else:
        client = Client(name = nom_client)
        client.save()

    for nom in client.noms_domaine:
        print(nom.fqdn)'''
    user = input("login user : ")
    nom_domaine = input("Entrez le nom de domaine : ")
    if not nom_domaine:
        print("Nom de domaine invalide.")
        return'''

    if NomDomaine.objects(fqdn = nom_domaine).count() == 1:
        print(NomDomaine.objects(fqdn = nom_domaine)[0].id)
        nom_domaine = NomDomaine.objects(fqdn = nom_domaine)[0]
    else:
        nom_domaine = NomDomaine(fqdn = nom_domaine)
        nom_domaine.save()

    client.noms_domaine.append(nom_domaine)
    client.save()

    scan = Scan(datetime = 122)
    scan.save()
```

```
        nom_domaine.scans.append(scan)

        nom_domaine.save()'''



        partie_user = User(

            nom = user,

            idDatas=iddd

        )

        partie_user.save()







        fqdn, type_info, ip, entreprise_info = cmdDNS(nom_domaine)

        #location, content_type, cache_control, server, x_frame_options =
cmdSite(nom_domaine)

        show_hosts_results, results_json = cmdHosts(nom_domaine)

        #show_hosts_results2, results_json2 = cmdHosts2(nom_domaine)

        #show_people_results, results_json3 = cmdPeople(nom_domaine)

        #cmdMail(nom_domaine)

        #cmdPorts(nom_domaine)



        partie_dns = PartieDNS(

            name=nom_domaine,

            DNStype=type_info,

            adressIP=ip,

            DNSentreprise=entreprise_info,

            idDatas=iddd

        )

        partie_dns.save()
```

```
'''partie_site =PartieSite(
    location = location,
    content = content_type,
    cache_control = cache_control,
    server = server,
    xframe = x_frame_options

)
partie_site.save()'''

print(f"Les informations pour {nom_domaine} ont été enregistrées.")

partie_squatting = PartieSquatting(
    nomFQDN = "",
    adressipsquatting = "",
    serveurDNS = "",
    Squattingtype= "",
    idDatas=iddd        )
partie_squatting.save()



partie_fp = PartieFicherpublics(
    url = "",
    typefp = "",
    confidentialite="",
    idDatas=iddd
)
partie_fp.save()
```

```
    except Exception as e:
        print(f"Une erreur s'est produite : {e}")


# fonction main
if __name__ == "__main__":
    executioncmd()
```