1. Python ilə sadə bir IP skaner necə yazılır?

import socket

import ipaddress

def scan\_ip(ip, port):

try:

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

s.settimeout(1)

s.connect((ip, port))

return True

except:

return False

ip\_range = ipaddress.ip\_network('192.168.1.0/24')

for ip in ip\_range.hosts():

for port in [22, 80, 443]: # Nümunə olaraq 22, 80, ve 443 nömrəli portlar yoxlanılır

if scan\_ip(str(ip), port):

print(f"Açıq port tapıldı: {ip} port {port}")

1. Bir şəbəkə qurğusunun konfiqurasiya faylını Python ilə necə təhlil edə bilərsiniz?

Bir şəbəkə qurğusunun konfiqurasiya faylını təhlil etmək üçün Python-dan istifadə edərkən bir neçə üsul var. Konfiqurasiya faylının formatı (məsələn, INI, XML, JSON, və ya xüsusi bir format) əsasında yanaşma dəyişə bilər. Aşağıdakı metodlardan istifadə oluna bilər:

1. Standart Kitabxana Modullarını İstifadə Etmək: Python, INI, XML, JSON kimi yaygın fayl formatlarını oxumaq və yazmaq üçün daxili modullara malikdir (`configparser`, `xml.etree.ElementTree`, `json` və s.). Bu modullar standart formatdakı faylları asanlıqla işləməyə imkan verir.

2. Regex İstifadə Etmək: Əgər konfiqurasiya faylı xüsusi bir formatdadırsa və ya müəyyən məlumatları çıxarmaq istəyirsinizsə, `re` modulu ilə düzenli ifadələr istifadə edə bilərsiniz. Bu, mətn içərisindən spesifik məlumatları tapmaq və çıxarmaq üçün güclü bir yoldur.

3.Yazılmış Kitabxanalardan İstifadə Etmək: Bəzi şəbəkə qurğuları üçün (məsələn, Cisco, Juniper) Python kitabxanaları mövcuddur. Bu kitabxanalar, həmin qurğuların konfiqurasiya fayllarını oxumaq və təhlil etmək üçün xüsusi olaraq hazırlanmışdır (məsələn, `pyats`, `napalm`).

4. Faylı Oxuyub Təhlil Etmək: `open()` funksiyasını istifadə edərək faylı açıb, satır satır oxuyaraq lazımi məlumatları çıxara bilərsiniz. Bu metod, fayl strukturu sadə və mütəmadi olarsa işləyir.

5. YAML və digər Formatlar: Əgər konfiqurasiya faylı YAML kimi bir formatda yazılmışsa, uyğun kitabxanalardan (məsələn, `pyyaml`) istifadə etmək mümkündür.

6. Pandas Kitabxanası: Əgər konfiqurasiya məlumatları cədvəl formatında (məsələn, CSV) təqdim edilirsə, `pandas` kitabxanası bu məlumatları asanlıqla oxuyub işləyə bilər.

Fərz edək ki, bizim şəbəkə qurğumuzun konfiqurasiya faylı sadə mətn formatında və aşağıdakı kimi xüsusiyyətlərə malikdir:

- Hər bir konfiqurasiya sətirində `ayar\_adi=deyer` formatı istifadə edilir.

- Fayl içərisində şərh sətrləri `#` işarəsi ilə başlayır.

Bu tip bir konfiqurasiya faylını Python ilə təhlil etmək üçün, biz faylı oxuyacaq, hər bir ayarı ayrı-ayrılıqda təhlil edəcək və nəticələri bir lüğət (dictionary) daxilində saxlayacağıq. İndi bu prosesi bir Python skripti ilə necə həyata keçirə biləcəyimizi göstərək:

```python

def konfigurasiya\_faylini\_oxu(fayl\_yolu):

konfigurasiya = {}

with open(fayl\_yolu, 'r') as fayl:

for satir in fayl:

satir = satir.strip()

# Şərh və boş sətirləri yoxlama

if satir.startswith("#") or not satir:

continue

# Ayar adını və dəyərini ayırma

if '=' in satir:

ayar\_adi, deyer = satir.split('=', 1)

konfigurasiya[ayar\_adi.strip()] = deyer.strip()

return konfigurasiya

# Konfiqurasiya faylını oxuyun və nəticələri göstərin

konfigurasiya = konfigurasiya\_faylini\_oxu("yol/konfigurasiya\_fayli.txt")

print(konfigurasiya)

```

Bu skript, göstərilən yoldakı konfiqurasiya faylını oxuyur və hər bir ayarı bir lüğətə (key-value pairs) çevirir. Bu lüğət, ayar adlarını açar (key) olaraq və müvafiq dəyərləri dəyər (value) olaraq saxlayır. Skript, şərh və boş sətirləri nəzərə almadan yalnızca konfiqurasiyanı işləyir.

1. Python ilə bir şifrələmə alqoritmini necə həyata keçirə bilərsiniz?

Python ilə şifrələmə alqoritmi həyata keçirmək üçün, ən sadə yanaşma bir çevrilmə (substitution) şifrəsindən istifadə etməkdir. Bu nümunədə, biz "Caesar Cipher" adlanan klassik bir şifrələmə metodu istifadə edəcəyik. Caesar Cipher, hər hərfi əlifbanın içində müəyyən bir sayda mövqelər ilə irəli və ya geri sürüşdürməklə işləyir. Məsələn, 3 mövqe ilə sürüşdürmə ilə 'A' hərfi 'D' olur, 'B' isə 'E', və s.:

1. `encrypt` funksiyası verilmiş mətni şifrələyəcək.

2. `decrypt` funksiyası şifrələnmiş mətni orijinal halına qaytaracaq.

Bu nümunə üçün hər hərfi üç mövqe irəli sürüşdürəcəyik.

Yuxarıda verilmiş Python kodu ilə Caesar Cipher şifrələmə və deşifrələmə funksiyalarını həyata keçirdik. Bu nümunədə, hər hərfi üç mövqe irəli sürüşdürdük.

- Orijinal mətn: `"Hello, World!"`

- Şifrələnmiş mətn: `"Khoor, Zruog!"`

- Deşifrə edilmiş mətn: `"Hello, World!"`

Bu kod, hərfləri sürüşdürərək şifrələyir və şifrələnmiş mətni yenidən orijinal halına qaytarır. Diqqət edilməsi lazım olan məqam, bu kodun yalnız hərfləri (latın əlifbasını) şifrələməsi və digər simvolları (məsələn, nöqtə, vergül) dəyişdirməməsidir. Bu, Caesar Cipher'ın ən sadə nümunəsidir və daha mürəkkəb şifrələmə tələbləri üçün daha güclü kriptografiya metodlarından istifadə etmək lazımdır.

def caesar\_cipher\_encrypt(text, shift):

encrypted\_text = ""

for char in text:

if char.isalpha(): # Yalnız hərfləri şifrələ

shift\_amount = shift % 26

char\_code = ord(char) + shift\_amount

if char.isupper():

char\_code = 65 + (char\_code - 65) % 26

else:

char\_code = 97 + (char\_code - 97) % 26

encrypted\_text += chr(char\_code)

else:

encrypted\_text += char

return encrypted\_text

def caesar\_cipher\_decrypt(encrypted\_text, shift):

return caesar\_cipher\_encrypt(encrypted\_text, -shift)

# Nümunə şifrələmə və deşifrələmə

original\_text = "Hello, World!"

shift\_amount = 3

encrypted = caesar\_cipher\_encrypt(original\_text, shift\_amount)

decrypted = caesar\_cipher\_decrypt(encrypted, shift\_amount)

original\_text, encrypted, decrypted

1. Bir şəbəkə qurğusunun avtomatik konfiqurasiyası üçün Python skripti necə yazılır?

Bir şəbəkə qurğusunun avtomatik konfiqurasiyası üçün Python skripti yazmaq, şəbəkə avtomatlaşdırmasının əsas hissəsidir. Bu cür skriptlər, şəbəkə qurğularını konfiqurasiya etmək, statuslarını yoxlamaq və məlumat toplamaq üçün istifadə edilir. Şəbəkə avtomatlaşdırması üçün ən populyar kitabxanalardan biri `netmiko`dur. `netmiko`, Cisco, Juniper, Arista və bir çox digər şəbəkə cihazlarını dəstəkləyən multi-vendor kitabxanadır.

Aşağıda, bir Cisco router'ını konfiqurasiya etmək üçün sadə bir `netmiko` skripti nümunəsi verilmişdir. Bu nümunədə, biz router'a bağlanıb, bir neçə konfiqurasiya əmri göndərəcəyik və sonra dəyişiklikləri tətbiq edəcəyik.

Əvvəlcə, `netmiko` kitabxanasını quraşdırmaq lazımdır. Bunu `pip install netmiko` komandası ilə edə bilərsiniz.

Nümunə Python Skripti:

```python

from netmiko import ConnectHandler

# Şəbəkə cihazının konfiqurasiya detalları

device = {

'device\_type': 'cisco\_ios', # Cihaz növü

'host': '192.168.1.1', # Cihazın IP ünvanı

'username': 'admin', # İstifadəçi adı

'password': 'yourpassword', # Şifrə

'port': 22, # SSH portu (əksər hallarda 22)

}

# Bağlantı yaradılır

net\_connect = ConnectHandler(\*\*device)

# Göndəriləcək konfiqurasiya əmrləri

commands = [

'conf t',

'interface GigabitEthernet0/1',

'description Python ile konfiqurasiya',

'ip address 10.0.0.1 255.255.255.0',

'no shutdown',

'end'

]

# Əmrlərin göndərilməsi

output = net\_connect.send\_config\_set(commands)

# Əlaqənin sonlandırılması

net\_connect.disconnect()

# Çıxışın çap edilməsi

print(output)

```

Bu skript, müəyyən bir router'a SSH vasitəsilə bağlanır, müəyyən interfeysi konfiqurasiya edir (məsələn, IP ünvan təyin edir, təsvir əlavə edir, və interfeysi aktivləşdirir) və sonra dəyişiklikləri tətbiq edir.

1. Python ilə log fayllarını təhlil etmək üçün nümunə skript necə olur?

Python ilə log fayllarını təhlil etmək üçün skript yazmaq, adətən, log fayllarını oxumaq, məlumatları axtarış kriteriyalarına əsasən süzmək və nəticələri təhlil etmək üçün lazımdır. Aşağıda sadə bir nümunə verəcəyəm ki, bu skript bir log faylından məlumatları oxuyub, hər bir xətt üzərində müəyyən bir axtarış kriteriyasına əsasən filtrasiya edəcək və uyğun gələn sətirləri göstərəcək.

Əsas fikir, `open` funksiyasını istifadə edərək log faylını oxumaq və hər bir xətt üzərində müəyyən bir şərti (məsələn, xüsusi bir söz və ya fraze varsa) yoxlamaqdır.

Məsələn, bir web server log faylında 404 xəta kodlarını axtaran bir skript yazaq:

# Log faylının yolu

log\_file\_path = 'path/to/your/logfile.log'

# Axtarılacaq fraze

search\_term = '404'

# Log faylını oxuyub axtarış aparılacaq funksiya

def analyze\_log(file\_path, term):

with open(file\_path, 'r') as file:

for line in file:

if term in line:

print(line.strip())

# Log faylını təhlil etmək

analyze\_log(log\_file\_path, search\_term)

```

Bu skript, göstərilən log faylındakı hər bir xətti oxuyur və hər xətt üçün `404` sözünü axtarır. Bu söz tapıldıqda, uyğun xətt ekrana çap olunur.

Bu sadə nümunəni fərqli axtarış kriteriyaları və daha mürəkkəb log təhlili üçün genişləndirmək mümkündür. Məsələn, müəyyən bir tarix aralığında olan xətalara baxmaq, xüsusi IP ünvanlarından gələn sorğuları təhlil etmək, və s. üçün əlavə şərtlər və regular ifadələr (regex) istifadə edilə bilər.

1. Python ilə bir zərərli proqram təhlili skripti necə hazırlanır?

Zərərli proqramların təhlili, həssas və mürəkkəb bir sahədir və bu sahədə işləmək üçün xüsusi təlim və biliklər tələb olunur. Zərərli proqramları təhlil edərkən həm təhlükəsizlik məsələləri, həm də qanuni aspektlər nəzərə alınmalıdır. Bununla birlikdə, Python ilə zərərli proqramları təhlil etmək üçün istifadə edilə biləcək əsas yanaşmaları qısaca izah edə bilərəm.

Python ilə Zərərli Proqram Təhlili Üçün Ümumi Yanaşmalar

**Statik Təhlil:**

- Fayl Şifrələnməsini Yoxlama:Zərərli proqramların tanınması üçün MD5, SHA-1 və ya SHA-256 kimi hash dəyərlərinin yoxlanması.

- String Təhlili: Faylda mətn stringlərini (URL-lər, IP ünvanları, şübhəli sözlər) axtarmaq.

- Fayl Formatı Təhlili: PE başlıqları, DLL asılılıqları və s. kimi fayl strukturlarını təhlil etmək.

**Dinamik Təhlil:**

- Şəbəkə Fəaliyyətinin İzlənməsi:Zərərli proqram tərəfindən yaradılan şəbəkə trafikini təhlil etmək.

- Sistem Çağırışlarının İzlənməsi: Zərərli proqramın sistemə etdiyi çağırışları və fayl əməliyyatlarını izləmək.

**Avtomatlaşdırılmış Təhlil Alətləri:**

- Zərərli fayllar üzərində nümunələrə əsaslanan təhlil aparmaq üçün qaydalar yaratmaq.

Python Skripti Nümunəsi: Statik String Təhlili

Bu sadə nümunə, zərərli proqram təhlili üçün bir faylda şübhəli stringləri axtarır.

import re

def scan\_file(file\_path):

# Şübhəli stringlərin siyahısı

suspicious\_patterns = ['http://', 'https://', 'cmd.exe', 'nc.exe', 'powershell']

# Faylı açıb təhlil etmək

with open(file\_path, 'rb') as file:

contents = file.read().decode(errors='ignore')

for pattern in suspicious\_patterns:

if

re.search(pattern, contents):

print(f"Şübhəli string tapıldı: {pattern}")

# Fayl yolu

file\_path = 'path/to/malware.exe'

scan\_file(file\_path)

```

Bu skript, faylda şübhəli stringlər axtarır və tapıldıqda bildirir. Lakin, bu yalnız çox sadə bir nümunədir və həqiqi zərərli proqram təhlili üçün daha mürəkkəb yanaşmalar və alətlər tələb olunur.

1. Python ilə bir faylın kriptoanalizi üçün skript necə yazılır?

Kriptoanaliz, şifrəli məlumatların şifrəsini açmaq və ya şifrələmə alqoritmlərinin zəifliklərini tapmaq üçün istifadə edilən bir texnikadır. Python ilə sadə bir kriptoanaliz skripti yazmaq mümkündür, ancaq bu, həqiqətən də geniş və mürəkkəb bir sahədir. Ən yaygın və sadə nümunə olaraq, bir mətnin Caesar Cipher ilə şifrələndiyini düşünərək, bu şifrəni açmaq üçün bir skript yaza bilərik.

Bu nümunədə, mətnin hər bir hərfini əlifbanın bütün mümkün mövqeləri boyunca sürüşdürərək (brute force metodu ilə), orijinal mətni tapmağa çalışacağıq. Bu metod, xüsusilə Caesar Cipher kimi sadə şifrələmə üsulları üçün effektivdir.

Aşağıda, Caesar Cipher ilə şifrələnmiş bir mətnin kriptoanalizini aparmaq üçün Python skripti verilmişdir:

```python

def caesar\_cipher\_decrypt(cipher\_text):

alphabet = 'abcdefghijklmnopqrstuvwxyz'

for key in range(len(alphabet)):

decrypted\_text = ''

for char in cipher\_text:

if char in alphabet:

position = alphabet.find(char)

new\_position = (position - key) % 26

decrypted\_text += alphabet[new\_position]

else:

decrypted\_text += char

print(f"Key {key}: {decrypted\_text}")

# Şifrələnmiş mətn

cipher\_text = "Khoor, Zruog!"

# Şifrənin açılması

caesar\_cipher\_decrypt(cipher\_text.lower())

```

Bu skript, verilmiş şifrəli mətn üçün əlifbanın bütün mümkün sürüşmələrini (0-dan 25-ə qədər) sınaqdan keçirir və hər bir açar üçün mümkün deşifrə edilmiş mətni göstərir. Caesar Cipher üçün doğru açar tapıldıqda, mətn mənasını almış olacaq.

1. Python-da bir şəbəkə trafikini izləmək üçün skript necə yazılır?

Python ilə şəbəkə trafikini izləmək üçün `scapy` kimi paketləri istifadə etmək mümkündür. `scapy` güclü və interaktiv bir şəbəkə paketi manipulyasiya kitabxanasıdır ki, şəbəkə paketlərini yaratmaq, göndərmək, təhlil etmək və qəbul etmək üçün istifadə olunur.

Aşağıda, `scapy` istifadə edərək sadə bir şəbəkə trafik izləmə skripti nümunəsini verəcəyəm. Bu skript, şəbəkə interfeysi vasitəsilə keçən paketləri dinləyir və onları ekrana çıxarır.

Əvvəlcə, `scapy` kitabxanasını quraşdırılması:

pip install scapy

Daha sonra aşağıdakı skripti yazaraq şəbəkə trafikini izləyə bilərik:

from scapy.all import sniff

# Paketləri təhlil etmək üçün funksiya

def process\_packet(packet):

print(packet.show())

# Şəbəkə trafikini izləmək

sniff(prn=process\_packet, store=False)

```

Bu skript `sniff` funksiyasını istifadə edərək şəbəkə trafikini dinləyir və hər bir paket üzərində `process\_packet` funksiyasını çağırır. `process\_packet` funksiyası hər bir paketi ekrana çıxarır. `store=False` parametri, paketlərin yaddaşda saxlanmasını dayandırır ki, bu, yaddaş istifadəsini azaldır.

1. Python ilə e-poçt təhlükəsizliyi üçün skript necə yazılır?

E-poçt təhlükəsizliyi üçün Python ilə skript yazmaq, e-poçtların şifrələnməsi, doğrulanması və təhlükəsiz bir şəkildə göndərilməsi üzərində fokuslanır. Bu, genellikle SSL/TLS şifrələməsini, güclü parol doğrulamasını və mümkünsə e-poçt mətninin şifrələnməsini əhatə edir.

Python'un `smtplib` və `email` modulları, e-poçtları təhlükəsiz bir şəkildə göndərmək üçün istifadə edilə bilər. Aşağıda, SSL ilə qorunan bir SMTP serveri vasitəsilə e-poçt göndərən sadə bir Python skripti nümunəsi verilmişdir.

Əvvəlcə, lazımi modulları idxal edin:

import smtplib

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

```

Sonra, e-poçt məlumatlarını və mesajı hazırlayın:

```

# Göndərən və alıcı e-poçt ünvanları

sender\_email = "your\_email@example.com"

receiver\_email = "receiver\_email@example.com"

password = input("Type your password and press enter: ")

# E-poçtun məzmununu yaratmaq

message = MIMEMultipart("alternative")

message["Subject"] = "Test Email"

message["From"] = sender\_email

message["To"] = receiver\_email

# E-poçtun mətnini və HTML versiyasını yaratmaq

text = """\

Hi,

How are you?

Real Python has many great tutorials."""

html = """\

<html>

<body>

<p>Hi,<br>

How are you?<br>

<a href="https://www.realpython.com">Real Python</a> has many great tutorials.

</p>

</body>

</html>

"""

# Mətn və HTML hissələrini MIMEText obyektləri kimi əlavə etmək

part1 = MIMEText(text, "plain")

part2 = MIMEText(html, "html")

# Mesaja hər iki hissəni əlavə etmək

message.attach(part1)

message.attach(part2)

```

Daha sonra, SMTP serveri ilə bağlantı qurun və mesajı göndərin:

```python

# Təhlükəsiz bağlantı ilə SMTP serverinə qoşulmaq

context = ssl.create\_default\_context()

with smtplib.SMTP\_SSL("smtp.example.com", 465, context=context) as server:

server.login(sender\_email, password)

server.sendmail(sender\_email, receiver\_email, message.as\_string())

```

Bu skripti işlətməzdən əvvəl, `your\_email@example.com`, `receiver\_email@example.com` və `smtp.example.com` kimi yer tutucuları öz məlumatlarınızla əvəz etməlisiniz. Həmçinin, SMTP serverinizin portunu və ünvanını dəqiq bilirsinizsə, onları da düzgün şəkildə daxil etməlisiniz.

1. Bir web saytının təhlükəsizliyini yoxlamaq üçün Python skripti necə yazılır?

Web saytının təhlükəsizliyini yoxlamaq üçün Python skripti yazmaq, saytın müxtəlif təhlükəsizlik aspektlərini yoxlamağı əhatə edə bilər. Bu, adətən, saytın SSL/TLS sertifikatlarını, başlıq məlumatlarını, açıq portları və potensial zəiflikləri yoxlamağı daxil edir. Bununla birlikdə, bu cür təhlillər mütəxəssis biliklər və təhlükəsizlik sahəsində təcrübə tələb edir.

Python'da `requests`, `ssl` və `socket` kimi standart kitabxanalardan istifadə edərək əsas bir web saytı təhlükəsizlik yoxlaması skripti yazmaq mümkündür.

**SSL/TLS Sertifikat Yoxlaması**

SSL/TLS sertifikatlarını yoxlamaq, web saytının şifrələmə təhlükəsizliyini yoxlamaq üçün vacibdir. Aşağıdakı nümunədə, `requests` kitabxanası istifadə edilərək bir saytın SSL/TLS sertifikatının etibarlılığını yoxlayırıq:

```python

import requests

def check\_ssl\_certificate(url):

try:

response = requests.get(url)

if response.history:

print(f"Redirected from {url} to {response.url}")

if response.ok:

print(f"SSL/TLS sertifikatı {url} üçün etibarlıdır.")

else:

print(f"SSL/TLS sertifikatı {url} üçün etibarlı deyil.")

except requests.exceptions.SSLError as e:

print(f"SSL/TLS xəta: {e}")

# Yoxlanılacaq URL

check\_ssl\_certificate("https://www.example.com")

```

Bu skript, verilmiş URL üçün SSL/TLS sertifikatının etibarlılığını yoxlayır.

### Bağlantı Portlarının Yoxlanılması

Aşağıdakı nümunədə, `socket` kitabxanası istifadə edilərək bir serverin müəyyən portlarının açıq olub-olmadığını yoxlayırıq:

```python

import socket

def check\_port(host, port):

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

sock.settimeout(1) # Timeout müddətini 1 saniyəyə qoyuruq

result = sock.connect\_ex((host, port))

if result == 0:

print(f"Port {port} açıqdır.")

else:

print(f"Port {port} bağlıdır.")

sock.close()

# Yoxlanılacaq host və portlar

host = "example.com"

ports = [80, 443, 8080, 8443]

for port in ports:

check\_port(host, port)

```

Bu skript, verilmiş hostun müəyyən portlarının açıq olub-olmadığını yoxlayır.

1. SSH vasitəsilə şəbəkə cihazlarını idarə etmək üçün Python skripti necə hazırlanır?

SSH vasitəsilə şəbəkə cihazlarını idarə etmək üçün Python skripti yazmaq, `paramiko` kimi SSH protokolu üçün nəzərdə tutulmuş kitabxanalardan istifadə etməyi tələb edir. `Paramiko` geniş istifadə olunan bir Python kitabxanasıdır ki, SSH2 protokolu ilə əlaqə qurmaq və şəbəkə cihazları ilə əməliyyatlar aparmaq üçün istifadə edilir.

Aşağıda, SSH vasitəsilə bir şəbəkə cihazına bağlanıb, bəzi əmrləri icra etmək və nəticələri oxumaq üçün sadə bir `paramiko` skripti nümunəsi verilmişdir.

Əvvəlcə, `paramiko` kitabxanasını quraşdırın:

```bash

pip install paramiko

```

Daha sonra aşağıdakı skripti yazaraq şəbəkə cihazı ilə əlaqə qura bilərsiniz:

```python

import paramiko

def ssh\_command(host, port, username, password, command):

# SSH bağlantısını yaradırıq

ssh = paramiko.SSHClient()

ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

ssh.connect(host, port, username, password)

# Komandaları icra edirik

stdin, stdout, stderr = ssh.exec\_command(command)

output = stdout.read().decode('utf-8')

# Bağlantını bağlayırıq

ssh.close()

return output

# Şəbəkə cihazının detalları

host = "192.168.1.1"

port = 22

username = "admin"

password = "yourpassword"

command = "show ip interface brief"

# SSH vasitəsilə komanda icra etmək

result = ssh\_command(host, port, username, password, command)

print(result)

```

Bu skript, verilmiş şəbəkə cihazına (host, port, username, password) SSH vasitəsilə bağlanır, göndərilmiş komandanı (`command`) icra edir və nəticələri çap edir.

1. Python ilə bir təhlükəsizlik audit skripti necə yazılır?

Bir təhlükəsizlik audit skripti yazmaq, mürəkkəb və spesifik bir prosesdir və bu skriptlər, çox fərqli təhlükəsizlik yoxlamalarını əhatə edə bilər. Python ilə təhlükəsizlik audit skripti yazarkən, adətən, aşağıdakı kimi yoxlamaları həyata keçirmək olar:

1. Açıq Portların Yoxlanılması: Sistemdəki açıq portları aşkarlamaq üçün `socket` modulundan istifadə edə bilərsiniz.

2. SSL/TLS Sertifikatlarının Yoxlanılması: Web serverlərin SSL/TLS sertifikatlarını yoxlamaq üçün `requests` və `ssl` modullarını istifadə edə bilərsiniz.

3. Zəif Parolların Yoxlanılması: Sistemdəki zəif və ya standart parolları aşkarlamaq üçün brute-force yanaşmasını istifadə edə bilərsiniz.

4. Tətbiqetmə Zəifliklərinin Yoxlanılması: XSS, SQL injeksiyası kimi web tətbiqetmə zəifliklərini yoxlamaq üçün `requests` və özəl yoxlama skriptləri istifadə edə bilərsiniz.

**Açıq Portların Yoxlanılması**

Aşağıdakı Python skripti, verilmiş bir IP ünvanındakı spesifik portların açıq olub-olmadığını yoxlayır:

import socket

def scan\_port(ip, port):

try:

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

sock.settimeout(1)

result = sock.connect\_ex((ip, port))

if result == 0:

print(f"Port {port} açıqdır.")

else:

print(f"Port {port} bağlıdır.")

sock.close()

except socket.error as e:

print(f"Xəta: {e}")

# Yoxlanılacaq IP ünvanı və port siyahısı

target\_ip = "192.168.1.1"

ports\_to\_scan = [22, 80, 443]

for port in ports\_to\_scan:

scan\_port(target\_ip, port)

```

Bu skript, `target\_ip` ünvanındakı hər bir portu (`ports\_to\_scan` siyahısında göstərilmiş) yoxlayır və portun açıq olub-olmadığını bildirir.

13. Python-da fayl şifrələmək üçün skript necə yazılır?

Python ilə fayl şifrələmək üçün, kriptografiya kitabxanalarından birini, məsələn, `cryptography` kitabxanasını istifadə etmək məqsədəuyğundur. `cryptography` kitabxanası, həm asanlıqla istifadə oluna bilən yüksək səviyyəli şifrələmə interfeysləri, həm də aşağı səviyyəli kriptografiya əməliyyatları üçün vasitələr təqdim edir.

**cryptography Kitabxanasını Quraşdırmaq**

pip install cryptography

**Fayl Şifrələmə Skripti**

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

from cryptography.hazmat.backends import default\_backend

import os

def encrypt\_file(file\_path, key, iv):

# AES Cipher obyektini yaradırıq

cipher = Cipher(algorithms.AES(key), modes.CFB(iv), backend=default\_backend())

# Faylı oxuyub şifrələyirik

with open(file\_path, 'rb') as f:

file\_data = f.read()

encryptor = cipher.encryptor()

encrypted\_data = encryptor.update(file\_data) + encryptor.finalize()

# Şifrələnmiş məlumatı eyni fayla yazırıq

with open(file\_path, 'wb') as f:

f.write(encrypted\_data)

# Şifrələmə açarı və IV (Initialization Vector) yaradırıq

key = os.urandom(32) # AES üçün 256-bit açar

iv = os.urandom(16) # AES CFB üçün IV

# Şifrələmək istədiyiniz faylın yolu

file\_to\_encrypt = 'path/to/your/file.txt'

encrypt\_file(file\_to\_encrypt, key, iv)

```

Bu skript, AES şifrələmə alqoritmini CFB rejimi ilə birlikdə istifadə edərək faylı şifrələyir. Açar (`key`) və ilkinləşdirmə vektoru (`iv`) təsadüfi yaradılır.

- Şifrələmə açarını və IV-ni təhlükəsiz bir şəkildə saxlamaq çox vacibdir, çünki bunlar olmadan şifrələnmiş məlumatları deşifrə etmək mümkün olmayacaq.

- Bu nümunədə açar və IV hər dəfə yeni yaradılır. Praktik istifadədə, bu məlumatları bir yerdə saxlayıb lazım olduqda istifadə etməlisiniz.

14. Python ilə bir şəbəkə monitorinq skripti necə yazılır?

Python ilə şəbəkə monitorinq skripti yazmaq, şəbəkə trafikini, cihaz vəziyyətlərini, və ya şəbəkədəki xüsusi hadisələri izləməyi əhatə edə bilər. Bir şəbəkə monitorinq skripti yazmaq üçün `ping` əməliyyatları, port tarama, və ya şəbəkə paketlərinin dinlənilməsi kimi metodlardan istifadə edilə bilər. Aşağıda, şəbəkədəki cihazların onlayn olub-olmadığını yoxlayan sadə bir Python skripti nümunəsi verilmişdir.

Bu skript, IP ünvanları siyahısını `ping` edərək hər bir cihazın şəbəkədə olub-olmamasını yoxlayır. Python'un standart kitabxanasında `ping` funksiyası olmadığı üçün, əməliyyat sisteminin komanda sətri vasitəsilə `ping` əməliyyatını həyata keçirəcəyik.

### Şəbəkə Monitorinq Skripti

```python

import os

import subprocess

def ping\_host(ip):

# Əməliyyat sisteminə görə `ping` əmri

param = '-n' if os.name == 'nt' else '-c'

# `ping` əmrini icra etmək

command = ['ping', param, '1', ip]

response = subprocess.run(command, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)

return response.returncode == 0

# Yoxlanılacaq IP ünvanları siyahısı

hosts = ["192.168.1.1", "192.168.1.2", "192.168.1.3"]

# Hər bir hostu `ping` edərək yoxlamaq

for host in hosts:

is\_up = ping\_host(host)

status = "Onlayn" if is\_up else "Ofsayn"

print(f"{host} {status}")

```

Bu skript, `hosts` siyahısındakı hər bir IP ünvanı üçün `ping` əməliyyatını həyata keçirir və hər bir cihazın şəbəkədə olub-olmadığını yoxlayır. `subprocess.run` funksiyası ilə əməliyyat sisteminin `ping` əmrini icra edirik.

- Bəzi sistemlər və ya şəbəkə konfiqurasiyaları `ping` cavablarını bloka edə bilər, bu halda skriptin nəticələri düzgün olmaya bilər.

- Şəbəkə monitorinqi, daha geniş və mürəkkəb bir sahədir və bu sadə nümunə, şəbəkə monitorinqinin bütün aspektlərini əhatə etmir.

15. Python-da bir şəbəkə qurğusundan məlumat toplama skripti necə yazılır?

Şəbəkə qurğusundan məlumat toplamaq üçün Python skripti yazmaq, adətən, şəbəkə qurğuları ilə SSH və ya SNMP protokolları vasitəsilə rabitə qurmağı əhatə edir. SSH üçün `paramiko` kitabxanasından, SNMP üçün isə `pysnmp` kimi kitabxanalardan istifadə edə bilərsiniz.

Aşağıda, `paramiko` kitabxanasını istifadə edərək bir şəbəkə qurğusuna SSH vasitəsilə bağlanıb məlumat toplamaq üçün bir Python skripti nümunəsi verilmişdir. Bu skript, SSH vasitəsilə şəbəkə qurğusuna bağlanır, müəyyən bir əmr icra edir (məsələn, `show version` və ya `show ip interface brief` kimi) və çıxışı oxuyur.

Əvvəlcə `paramiko` kitabxanasını quraşdırın:

```bash

pip install paramiko

```

Sonra aşağıdakı skripti yazaraq şəbəkə qurğusundan məlumat toplaya bilərsiniz:

```python

import paramiko

def get\_device\_info(host, port, username, password, command):

# SSH bağlantısını yaradırıq

ssh = paramiko.SSHClient()

ssh.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

ssh.connect(host, port, username, password)

# Əmri icra edirik və çıxışı oxuyuruq

stdin, stdout, stderr = ssh.exec\_command(command)

output = stdout.read().decode('utf-8')

ssh.close()

return output

# Şəbəkə qurğusunun detalları

host = "192.168.1.1"

port = 22

username = "your\_username"

password = "your\_password"

command = "show version" # Məsələn, Cisco IOS üçün

# SSH vasitəsilə məlumat toplamaq

device\_info = get\_device\_info(host, port, username, password, command)

print(device\_info)

```

Bu skriptdə `get\_device\_info` funksiyası, verilmiş şəbəkə qurğusuna bağlanır və göndərilmiş əmri icra edir. Nəticədə alınan məlumatları dönür.

16. Python ilə bir faylın inteqrasiyasını yoxlamaq üçün skript necə yazılır?

Faylın inteqrasiyasını yoxlamaq üçün Python skripti yazmaq, adətən, faylın kriptografik hash dəyərini (məsələn, MD5, SHA-1, SHA-256) hesablamaq və bu dəyəri gözlənilən bir dəyərlə müqayisə etmək prosesini əhatə edir. Bu yanaşma, faylın dəyişdirilmədən qaldığını yoxlamaq üçün istifadə olunur.

Aşağıda, Python ilə bir faylın SHA-256 hash dəyərini hesablayıb, bu dəyəri gözlənilən bir hash ilə müqayisə edən bir skript nümunəsi verilmişdir:

```python

import hashlib

def calculate\_file\_hash(file\_path, hash\_type='sha256'):

hash\_obj = hashlib.new(hash\_type)

with open(file\_path, 'rb') as file:

for chunk in iter(lambda: file.read(4096), b""):

hash\_obj.update(chunk)

return hash\_obj.hexdigest()

def verify\_file\_integrity(file\_path, expected\_hash, hash\_type='sha256'):

calculated\_hash = calculate\_file\_hash(file\_path, hash\_type)

if calculated\_hash == expected\_hash:

return True

else:

return False

# Fayl yolu və gözlənilən hash dəyəri

file\_path = 'path/to/your/file.txt'

expected\_hash = 'your\_expected\_sha256\_hash\_here'

# Fayl inteqrasiyasını yoxlamaq

is\_intact = verify\_file\_integrity(file\_path, expected\_hash)

if is\_intact:

print(f"Fayl {file\_path} inteqrasiyası qorunub.")

else:

print(f"Fayl {file\_path} inteqrasiyası pozulub.")

```

Bu skriptdə `calculate\_file\_hash` funksiyası verilmiş faylın hash dəyərini hesablayır, `verify\_file\_integrity` funksiyası isə hesablanan hash dəyərini gözlənilən dəyərlə müqayisə edir.

- Faylın inteqrasiyasını yoxlamaq üçün istifadə edilən hash metodunu və dəyərini etibarlı bir mənbədən əldə etmək vacibdir.

- MD5 və SHA-1 kimi köhnə hash funksiyaları artıq təhlükəsiz sayılmır və mümkün olduqda SHA-256 kimi daha güclü hash funksiyalarından istifadə etmək tövsiyə olunur.

- Bu yanaşma, faylın təsadüfi və ya qəsdən dəyişdirilməməsi barədə bir təminat verir, lakin faylın məzmununu və ya məqsədini qiymətləndirmir.

17. Python ilə bir şifrə yaradıcı skript necə yazılır?

Python ilə güclü və təsadüfi şifrələr yaradmaq üçün `secrets` və `string` modullarından istifadə edə bilərsiniz. `secrets` modulu, təsadüfi məlumatlar yaratmaq üçün kriptografik cəhətdən etibarlı vasitələr təqdim edir, ki bu da şifrələrin güclü və təhlükəsiz olmasını təmin edir.

Aşağıdakı Python skripti, istifadəçi tərəfindən müəyyən edilmiş uzunluqda təsadüfi, güclü bir şifrə yaradır:

```python

import secrets

import string

def generate\_password(length, complexity=3):

# Mümkün simvolların siyahıları

characters = [

string.ascii\_lowercase, # Kiçik hərflər

string.ascii\_uppercase, # Böyük hərflər

string.digits, # Rəqəmlər

string.punctuation # Xüsusi simvollar

]

# Şifrənin mürəkkəbliyinə görə simvolları seçmək

if complexity < 1 or complexity > 4:

raise ValueError("Complexity must be between 1 and 4")

chosen\_characters = ''.join(characters[:complexity])

# Təsadüfi şifrə yaratmaq

password = ''.join(secrets.choice(chosen\_characters) for i in range(length))

return password

# Şifrənin uzunluğu və mürəkkəbliyi

password\_length = 12

password\_complexity = 4 # 1-dən 4-ə qədər (daha çox simvol növü istifadə etmək)

# Şifrə yaratmaq

password = generate\_password(password\_length, password\_complexity)

print(password)

```

Bu skriptdə `generate\_password` funksiyası, təyin edilmiş uzunluq və mürəkkəblik dərəcəsinə görə şifrə yaradır. `complexity` dəyişəni 1-dən 4-ə qədər ola bilər və bu, şifrənin yalnız hərflərdən, hərflər və rəqəmlərdən, yaxud hərflər, rəqəmlər və xüsusi simvolların birləşməsindən ibarət olmasını müəyyən edir.

- Şifrələrin etibarlı və təhlükəsiz saxlanılması çox vacibdir. Şifrələri açıq mətndə saxlamaq və ya şifrələri təkrar istifadə etmək təhlükəsiz deyil.

- İstifadə edəcəyiniz sistem və ya servis üçün xüsusi şifrə tələblərini nəzərə alın (məsələn, xüsusi simvolların istifadəsi və ya maksimum uzunluq məhdudiyyətləri).

18. Python ilə bir təhlükəsizlik siyasəti yoxlamaq üçün skript necə yazılır?

Bir təhlükəsizlik siyasətini yoxlamaq üçün Python skripti yazmaq, şirkətin və ya təşkilatın təhlükəsizlik standartlarına uyğunluğu yoxlamaq üçün bir sıra yoxlamaları əhatə edə bilər. Bu yoxlamalar, parol siyasətləri, şəbəkə konfiqurasiyaları, sistem təhlükəsizlik tədbirləri və digər təhlükəsizliklə əlaqədar aspektlər ola bilər.

Aşağıda, müəyyən bir təhlükəsizlik siyasətinin tətbiq olunub-olunmadığını yoxlayan Python skriptinin nümunəsi verilmişdir. Bu nümunə, sistemin müəyyən bir şəbəkə portunun açıq olub-olmamasını və müəyyən bir faylın mövcudluğunu yoxlayır, ki bu da təhlükəsizlik siyasətinin bir hissəsi ola bilər.

**Təhlükəsizlik Siyasəti Yoxlama Skripti**

```python

import socket

import os

def check\_port\_open(host, port):

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

return s.connect\_ex((host, port)) == 0

def check\_file\_exists(file\_path):

return os.path.exists(file\_path)

# Təhlükəsizlik siyasəti yoxlamaları

security\_policy\_checks = {

"Şəbəkə Portu Yoxlaması": check\_port\_open("localhost", 22),

"Mühüm Faylın Mövcudluğu Yoxlaması": check\_file\_exists("/path/to/important/file")

}

# Nəticələri çap etmək

for check, result in security\_policy\_checks.items():

status = "Keçdi" if result else "Uğursuz oldu"

print(f"{check}: {status}")

```

Bu skriptdə iki funksiya var: `check\_port\_open` və `check\_file\_exists`. `check\_port\_open` funksiyası verilmiş portun açıq olub-olmamasını yoxlayır, `check\_file\_exists` isə verilmiş yolda faylın mövcudluğunu yoxlayır. Daha sonra bu yoxlamalar `security\_policy\_checks` lüğətində yer alır və hər bir yoxlamanın nəticəsi çap edilir.

Şəbəkə paketlərinin təhlili üçün Python skripti yazmaq, `scapy` kimi güclü bir şəbəkə təhlili kitabxanasından istifadə etməyi tələb edir. `scapy` vasitəsilə şəbəkə trafikini dinləyə bilər və HTTP istəklərinin başlıqlarını təhlil edə bilərsiniz. Aşağıda, HTTP istəklərinin başlıqlarını təhlil edən və müəyyən kriteriyalara əsasən məlumat çap edən Python skriptinin nümunəsi verilmişdir.

İlk olaraq, `scapy` kitabxanasını quraşdırın:

```bash

pip install scapy

```

Sonra, aşağıdakı skripti yazın:

```python

from scapy.all import sniff

from scapy.layers.http import HTTPRequest

def packet\_callback(packet):

if not packet.haslayer(HTTPRequest):

return

http\_layer = packet.getlayer(HTTPRequest)

headers = http\_layer.fields

# İstədiyiniz kriteriyalara uyğun paketləri filtrləyin

user\_agent = headers.get('User-Agent')

host = headers.get('Host')

if user\_agent is not None and host is not None:

print(f"Host: {host}, User-Agent: {user\_agent}")

# Şəbəkə paketlərini dinləmək

sniff(filter="tcp port 80", prn=packet\_callback, store=False)

```

Bu skript, şəbəkədəki TCP port 80-dən keçən trafiki dinləyir və HTTP istəklərinin başlıqlarını təhlil edir. `packet\_callback` funksiyası hər bir paketi təhlil edir, HTTP istəklərinin `Host` və `User-Agent` başlıqlarını yoxlayır və onları çap edir.

- `sniff` funksiyasının `filter` parametri, yalnız HTTP trafikini dinləmək üçün `tcp port 80` kimi təyin olunmuşdur. HTTPS trafiki üçün bu metod işləməyəcək, çünki HTTPS məlumatları şifrələnmişdir.

Şəbəkə paketlərinin təhlili üçün Python skripti yazmaq, `scapy` kimi güclü bir şəbəkə təhlili kitabxanasından istifadə etməyi tələb edir. `scapy` vasitəsilə şəbəkə trafikini dinləyə bilər və HTTP istəklərinin başlıqlarını təhlil edə bilərsiniz. Aşağıda, HTTP istəklərinin başlıqlarını təhlil edən və müəyyən kriteriyalara əsasən məlumat çap edən Python skriptinin nümunəsi verilmişdir.

İlk olaraq, `scapy` kitabxanasını quraşdırın:

```bash

pip install scapy

```

Sonra, aşağıdakı skripti yazın:

```python

from scapy.all import sniff

from scapy.layers.http import HTTPRequest

def packet\_callback(packet):

if not packet.haslayer(HTTPRequest):

return

http\_layer = packet.getlayer(HTTPRequest)

headers = http\_layer.fields

# İstədiyiniz kriteriyalara uyğun paketləri filtrləyin

user\_agent = headers.get('User-Agent')

host = headers.get('Host')

if user\_agent is not None and host is not None:

print(f"Host: {host}, User-Agent: {user\_agent}")

# Şəbəkə paketlərini dinləmək

sniff(filter="tcp port 80", prn=packet\_callback, store=False)

```

Bu skript, şəbəkədəki TCP port 80-dən keçən trafiki dinləyir və HTTP istəklərinin başlıqlarını təhlil edir. `packet\_callback` funksiyası hər bir paketi təhlil edir, HTTP istəklərinin `Host` və `User-Agent` başlıqlarını yoxlayır və onları çap edir.

- `sniff` funksiyasının `filter` parametri, yalnız HTTP trafikini dinləmək üçün `tcp port 80` kimi təyin olunmuşdur. HTTPS trafiki üçün bu metod işləməyəcək, çünki HTTPS məlumatları şifrələnmişdir.