LA RECONNAISSANCE

PYTHON POUR LA CYBERSÉCURITÉ

POURQUOI?

- Etablir un état des lieux pour planifier une attaque et déterminer les outils nécessaires
- Décrire la topologie réseau, la structure des systèmes, les systèmes d'opérations et les applications
- Découvrir où se trouvent les vulnérabilités avant de sélectionner et de choisir des exploits pour une cible.
- Ne se limite pas à la technique mais aussi à rechercher sur Internet ou autre:
  - La liste des adresses email et le format utilisé pour la nomenclature
  - L'hiérarchie organisationnelle des personnes cibles
  - Les vendeur et les clients
  - Backdoors possibles via les fournisseurs de maintenance et de support
  - Etc.

Être un guerrier n'est pas une simple question de vouloir l'être. C'est plutôt une lutte sans fin qui se poursuivra jusqu'au tout dernier moment de notre vie. Personne ne naît guerrier, exactement de la même manière que personne ne naît homme moyen. On se fait l'un ou l'autre

—Kokoro de Natsume So-sek, 1914, Japon.

LA RECONNAISSANCE RÉSEAU

LA RECONNAISSANCE
PYTHON POUR LA CYBERSÉCURITÉ

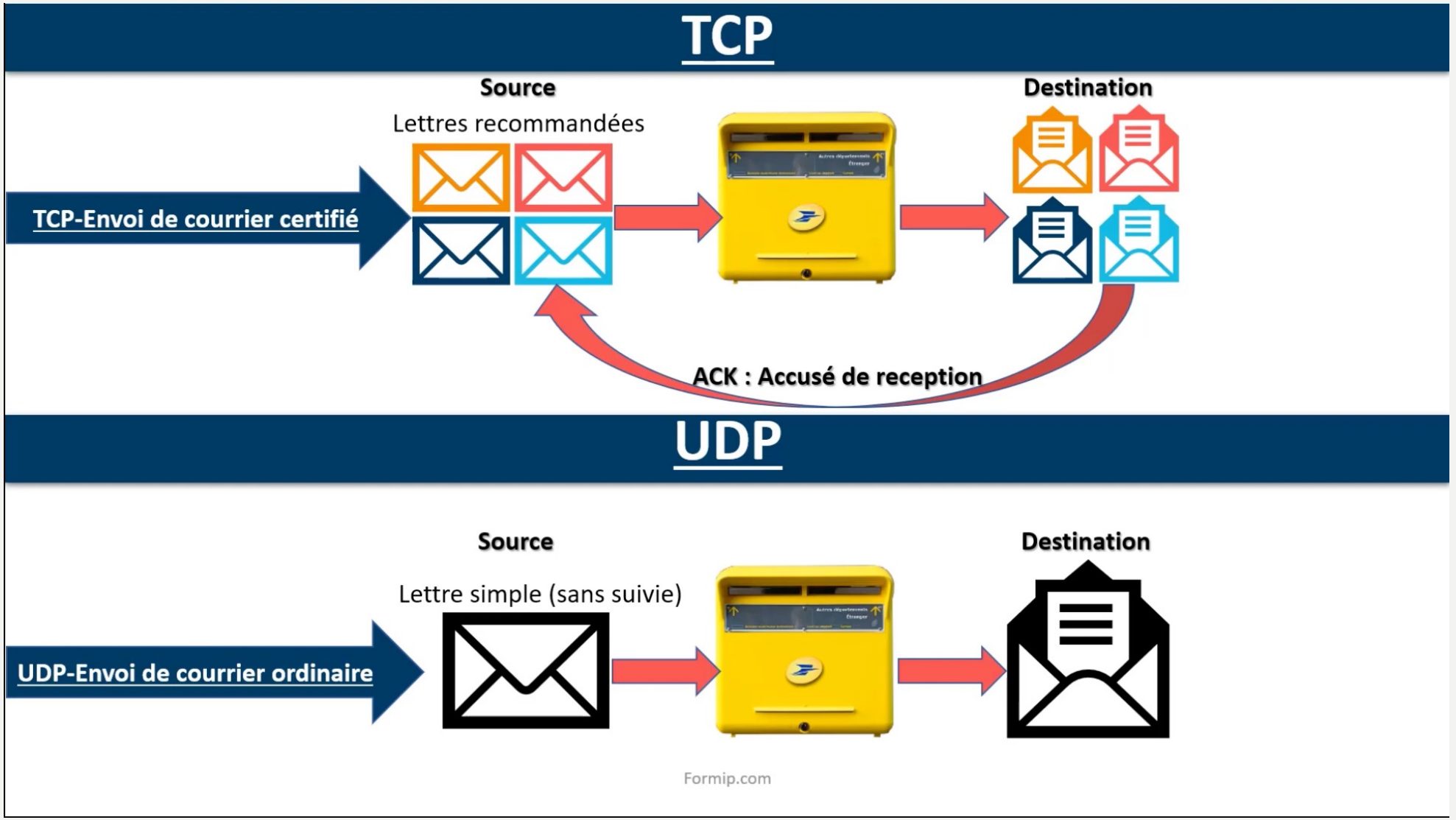
OBJECTIFS

- A la fin de cette partie, vous devriez être en mesure de :
  - Expliquer le but de la reconnaissance du réseau en Python
  - Décrire les fonctions et les informations fournies par la reconnaissance du réseau
  - Résumer les considérations importantes lors de la réalisation d'analyses de port
  - Identifier les dangers de travailler avec des outils bruyants pour les scans de port
  - Décrire les options de traitement des ports réseau en Python
  - Discuter du concept de capture de bannière en Python
  - Analyser l'importance de Nmap dans les scans de ports en Python
  - Analyser l'importance de Scapy & DPKT dans la capture des paquets en Python

OBJECTIFS

- Décrire un paquet IP et un paquet ICMP
- Décrire le concept de scan d'hôte en mode UDP
- Discuter du concept de capture et d'analyse de paquets en Python
- Ecrire un serveur/client tcp/udp en Python
- Ecrire un « scanner » de ports en Python
- Ecrire un « analyseur » de paquets en Python
- Utiliser la librairie Scapy et DPKT pour l'inspection de paquets en Python
- Utiliser la librairie NMAP pour l'inspection de paquets en Python

RAPPEL TCP / UDP



SERVEUR TCP

```python
import socket

server = socket.socket(family=socket.AF_INET, type=socket.SOCK_STREAM)

host = socket.gethostname()
port = 444

server.bind((host, port))
server.listen(5)

print(f"Running TCP server on {host}:{port}")
while True:
    client, address = server.accept()
    print(f"Received connection from {client} {address}")
    message = "Hello!\r\n"
    client.send(message.encode('ascii'))
    print(f"Sent TCP to {client} {address}")
    client.close()
    print(f"Closed TCP with {client} {address}")
```

CLIENT TCP

```python
import socket

client = socket.socket(family=socket.AF_INET, type=socket.SOCK_STREAM)

host = socket.gethostname()
port = 444

print(f"Connection to {host}:{port}...")
client.connect((host, port))
print(f"Connected to {host}:{port}, grabbing header")
message = client.recv(1024)
print(f"Received from {host}:{port}")
client.close()
print(f"Closed connection with {host}:{port}")

print("Received from {}:{} : {}".format(host,port,message))
```

SERVEUR UDP

```python
import socket

server = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

host = socket.gethostname()
port = 666

server.bind((host, port))

print(f"Running UDP server on {host}:{port}")
while True:
    message, address = server.recvfrom(1024)
    print(f"Received UDP from {address} {message}")
    message = "Hello!\r\n"
    server.sendto(message.encode('ascii'),address)
    print(f"Sent UDP {address}")
```

CLIENT UDP

```python
import socket

client = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

host = socket.gethostname()
port = 666

print(f"Sending UDP to {host}:{port} ")
client.sendto(b'Banner query\r\n',(host,port))
print(f"Waiting for response...")
message = client.recvfrom(1024)
print(f"Received from {host}:{port} : {message}")
```

LE SCAN DES PORTS

- Permet de dresser une liste des cibles, des ports disponibles et par conséquent des applications potentiellement vulnérables qui tournent sur la cible

- **Comment exécuter un scan des ports?**
  - Identifier les adresses IP ou nom de domaine des cibles potentielles
  - Identifier les ports ouverts sur cette cible

LE SCAN DES PORTS

- Identifier les ports ouverts sur une cible
  - Passer en commande la liste des IP et ports à vérifier
  - Envoyer un message TCP sur chaque port
  - Envoyer un message UDP sur chaque port
  - Sauvegarder les résultats dans un fichier
  - Utiliser le multi-threading pour optimiser

LE SCAN DES HÔTES EN UDP

- Technique qui repose sur le comportement des OS
- Un port UDP est fermé => message ICMP : port inaccessible
- Le but: définir une liste de cible potentielle
- Le comment: Envoie de message UDP sur un port
  - Une réponse = un hôte
  - Pas de réponse = pas d'hôte
- Le pourquoi: UDP = pas de « overhead » = découverte rapide et peu bruyante

LE SCAN DES HÔTES EN UDP

```python
import socket
import os

HOST = "192.168.0.87"

def main():
    if os.name == 'nt':
        socket_protocol = socket.IPPROTO_IP
    else:
        socket_protocol = socket.IPPROTO_ICMP

    sniffer = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket_protocol)
    sniffer.bind((HOST, 0))
    sniffer.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)

    if os.name == 'nt':
        sniffer.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)

    print(sniffer.recvfrom(65565))

    if os.name == 'nt':
        sniffer.ioctl(socket.SIO_RCVALL, socket.RCVALL_OFF)


if __name__ == '__main__':
    main()
```

LE SCAN DES HÔTES EN UDP

b'E\x00@\x00\xe1\x93\x00\x000\x01uw\\\xcd\x15\xd2\xc0\xa8\x00W'
=
45 00 40 00 e1 93 00 00 30 01 75 77 5c cd 15 d2 c0 a8 00 57

Internet Protocol					
Bit Offset	0–3	4–7	8–15	16–18	19–31
0	Version	HDR Length	Type of Service	Total Length	
32	Identification			Flags	Fragment Offset
64	Time to Live		Protocol	Header Checksum	
96	Source IP Address				
128	Destination IP Address				
160	Options				

LE SCAN DES HÔTES EN UDP

b'E\x00@\x00\xe1\x93\x00\x000\x01uw\\\xcd\x15\xd2\xc0\xa8\x00W'
=
45 00 40 00 e1 93 00 00 30 01 75 77 5c cd 15 d2 c0 a8 00 57

Bit Offset	0-3	4-7	8-15	16-18	19-31
0	Version 4	HDR L. 5	Type of Service 00	Length 40 00	
32	Identification E1 93			Flags b000	Fragment Offset b00000
64	TTL 30		Protocol 01	Header Checksum 75 77	
96	Source IP 5C CD 15 D2				
128	Destination IP C0 A8 00 57				

LE SCAN DES HÔTES EN UDP

```python
def sniff(host):
    if os.name == 'nt':
        socket_protocol = socket.IPPROTO_IP
    else:
        socket_protocol = socket.IPPROTO_ICMP

    sniffer = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket_protocol)
    sniffer.bind((host, 0))
    sniffer.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)

    if  os.name == 'nt':
        sniffer.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)

    try:
        while True:
            raw_buffer = sniffer.recvfrom(65535)[0]
            ip_header = IP(raw_buffer[0:20])
            print('Protocol: %s %s -> %s' % (ip_header.protocol, ip_header.src_address, ip_header.dst_address))
            # print(f'Version: {ip_header.ver} Header Length: {ip_header.ihl}  TTL: {ip_header.ttl}')

    except KeyboardInterrupt:
        if  os.name == 'nt':
            sniffer.ioctl(socket.SIO_RCVALL, socket.RCVALL_OFF)
        sys.exit()

if __name__ == '__main__':
    if len(sys.argv) == 2:
        host = sys.argv[1]
    else:
        host = HOST
    sniff(host)

```

LE SCAN DES HÔTES EN UDP

```
 8  class IP:
 9      def __init__(self, buff=None):
10          header = struct.unpack('<BBHHHBBH4s4s', buff)
11          self.ver = header[0] >> 4
12          self.ihl = header[0] & 0xF
13
14          self.tos = header[1]
15          self.len = header[2]
16          self.id = header[3]
17          self.offset = header[4]
18          self.ttl = header[5]
19          self.protocol_num = header[6]
20          self.sum = header[7]
21          self.src = header[8]
22          self.dst = header[9]
23
24          # human readable IP addresses
25          self.src_address = ipaddress.ip_address(self.src)
26          self.dst_address = ipaddress.ip_address(self.dst)
27
28          # map protocol constants to their names
29          self.protocol_map = {1: "ICMP", 6: "TCP", 17: "UDP"}
30          try:
31              self.protocol = self.protocol_map[self.protocol_num]
32          except Exception as e:
33              print('%s No protocol for %s' % (e, self.protocol_num))
34              self.protocol = str(self.protocol_num)
```

LE SCAN DES HÔTES EN UDP

WINDOWS

```
Protocol: UDP 192.168.0.190 -> 192.168.0.1
Protocol: UDP 192.168.0.1 -> 192.168.0.190
Protocol: UDP 192.168.0.190 -> 192.168.0.187
Protocol: TCP 192.168.0.187 -> 74.125.225.183
Protocol: TCP 192.168.0.187 -> 74.125.225.183
Protocol: TCP 74.125.225.183 -> 192.168.0.187
Protocol: TCP 192.168.0.187 -> 74.125.225.183
```

LINUX

```
Protocol: ICMP 74.125.226.78 -> 192.168.0.190
Protocol: ICMP 74.125.226.78 -> 192.168.0.190
Protocol: ICMP 74.125.226.78 -> 192.168.0.190
```

LE SCAN DES HÔTES EN UDP

```
12   0000  3C 15 C2 C3 B1 02 68 02 B8 8F 16 8A 08 00 45 C0  <.....h.......E.
13   0010  00 70 71 0A 00 00 40 01 87 1A C0 A8 00 01 C0 A8  .pq...@.........
14   0020  00 57 03 01 FC FE 00 00 00 00 45 00 00 54 C7 5D  .W........E..T.]
15   0030  00 00 3F 01 32 72 C0 A8 00 57 C0 A8 00 32 08 00  ..?.2r...W...2..
16   0040  96 88 CE 08 01 87 64 3F 88 A8 00 04 B9 F8 08 09  ......d?........
17   0050  0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19  ................
18   0060  1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29  ...... !"#$%&'()
19   0070  2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37        *+,-./01234567
```

Destination Unreachable Message		
0–7	8–15	16–31
Type = 3	Code	Header Checksum
Unused		Next-hop MTU
IP Header and First 8 Bytes of Original Datagram's Data		

LE SCAN DES HÔTES EN UDP

03 01 FC FE 00 00 00 00 45 00 00 54 C7 5D 00 00 3F 01
32 72 C0 A8 00 57 C0 A8 00 32 08 00 96 88 CE 08 01 87

Bit Offset	0-7	8-15	16-31
0	Type 03	Code 01	Header Checksum FC FE
32	Unused 00 00		Next-hop MTU 00 00
64 > 128	Original IP Header 45 00 00 54 C7 5D 00 00 3F 01 32 72		
160 > 192	Source IP > Destination IP C0 A8 00 57 > C0 A8 00 32		
224	First 8 bytes of datagram 08 00 96 88 CE 08 01 87		

LE SCAN DES HÔTES EN UDP

- Définir le sous-reseau à scanner
- Envoyer un paquet UDP à chaque hôte du sous-réseau
- Lancer le sniffer pour capturer les paquets ICMP retournés par le réseau
- Pour chaque paquet reçu, imprimé l'adresse de l'hôte

NMAP

- Une librairie python pour le scan des ports et des vulnérabilités
- Permet de faire des scans avancés comme les scans ACK, RST, FIN, or SYN-ACK
  - Prend énormément de temps à développer toutes les fonctionnalités
- Il faut installer NMAP sur la machine et la librairie python-nmap
  - Donc peut limiter où on peut lancer/inclure NMAP

NMAP

- Documentation complète des options sur le site officiel de NMAP
  - TCP SYN SCAN – aussi connu comme scan semi-ouvert, ce type de scan lance une connexion TCP avec un paquet SYN et attend la réponse: un paquet RST indique un port fermé, un paquet SYN/ACK indique un port ouvert
  - TCP NULL SCAN – consiste à mettre le TCP flag à zéro et attendre la réponse: un paquet RST indique un port fermé.
  - TCP FIN SCAN – consiste à envoyer un FIN qui termine une connexion TCP active: un paquet RST indique un port fermé.
  - TCP XMAS SCAN – consiste à mettre les bits PSH, FIN et URG à 1 dans le TCP flag: un paquet RST indique un port fermé.

SCAPY & DPKT

- Deux outils de capture de paquets
- Incluent differents outils pour la manipulation des paquets
- Capacité d'injection de paquets

```
1  from scapy.all import *
2
3  def show_packet(packet):
4      print(packet.show())
5
6  sniff(filter="icmp",iface="en0",prn=show_packet,count=10)
```

SCAPY

- Un outil de capture de paquets
- Differents outils pour la manipulation des paquets
- Capacité d'injection de paquets

```python
from scapy.all import *

def show_packet(packet):
    print(packet.show())

sniff(filter="icmp",iface="en0",prn=show_packet,count=10)
```

SCAPY

```
###[ Ethernet ]###
  dst       = 68:02:b8:8f:16:8a
  src       = 3c:15:c2:c3:b1:02
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 40
     id        = 5687
     flags     =
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xe2bc
     src       = 192.168.0.87
     dst       = 20.189.172.32
     \options   \
###[ TCP ]###
        sport     = 59073
        dport     = https
        seq       = 495785817
        ack       = 2784945395
        dataofs   = 5
        reserved  = 0
        flags     = A
        window    = 4096
        chksum    = 0x75a2
        urgptr    = 0
        options   = ''
```

```
###[ Ethernet ]###
  dst       = 3c:15:c2:c3:b1:02
  src       = 68:02:b8:8f:16:8a
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0xc0
     len       = 112
     id        = 28994
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x86e2
     src       = 192.168.0.1
     dst       = 192.168.0.87
     \options   \
###[ ICMP ]###
        type      = dest-unreach
        code      = host-unreachable
        chksum    = 0xfcfe
        reserved  = 0
        length    = 0
        nexthopmtu= 0
        unused    = ''
###[ IP in ICMP ]###
           version   = 4
           ihl       = 5
           tos       = 0x0
           len       = 84
           id        = 29916
           flags     =
           frag      = 0
           ttl       = 63
           proto     = icmp
           chksum    = 0x84f3
           src       = 192.168.0.87
           dst       = 192.168.0.50
           \options   \
###[ ICMP in ICMP ]###
              type      = echo-request
              code      = 0
              chksum    = 0x88e
              id        = 0xba0f
              seq       = 0x3
              unused    = ''
```

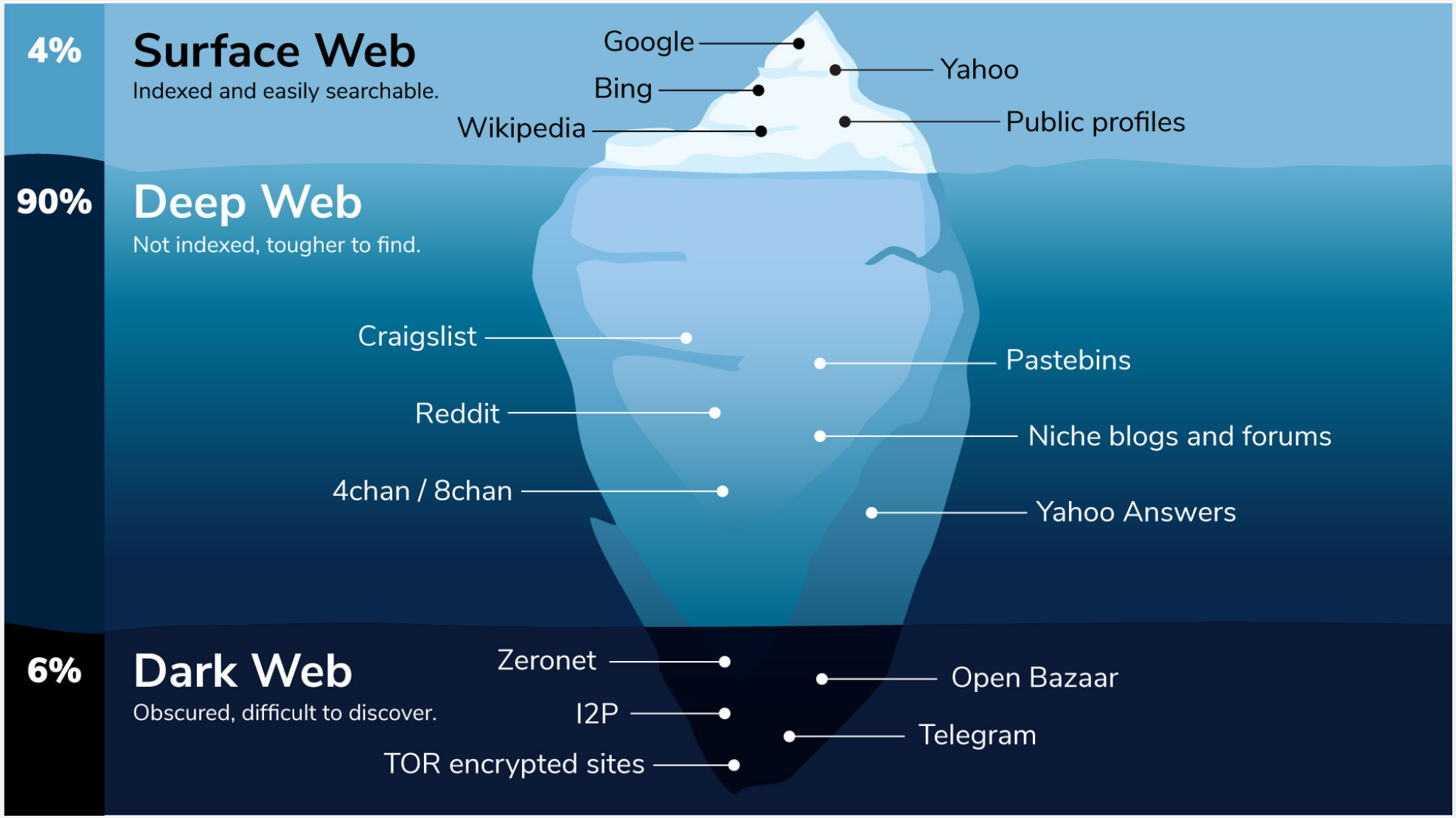
LA RECONNAISSANCE WEB

LA RECONNAISSANCE
PYTHON POUR LA CYBERSÉCURITÉ

OBJECTIFS

- A la fin de cette partie, vous devriez être en mesure de :
  - Décrire le concept de web scraping en Python
  - Expliquer comment écrire des programmes pour le Web scraping à l'aide de Python
  - Identifier diverses fonctions de Python dans l'automatisation des requêtes de recherche
  - Décrire les considérations clés lors de la réalisation de web scraping
  - Discuter du concept de phishing par e-mail

LE WEB


4%
Surface Web
Indexed and easily searchable.
Google
Yahoo
Bing
Wikipedia
Public profiles
90%
Deep Web
Not indexed, tougher to find.
Craigslist
Pastebins
Reddit
Niche blogs and forums
4chan / 8chan
Yahoo Answers
6%
Dark Web
Obscured, difficult to discover.
Zeronet
Open Bazaar
I2P
Telegram
TOR encrypted sites

LE WEB

- La meilleure resource pour s'informer sur votre cible
- Et aussi la meilleure pour éviter de sonner l'alarme
  - Detection prématurée
- Les pages web contiennent de l'information utile, souvent caché dans la source HTML
  - **Le web scraping** permet de télécharger les données pour analyse
  - Ce qui a pour effet aussi de minimiser la détection prématurée
  - Les données peuvent contenir aussi des liens utiles, des images, des fichiers, etc.

LE WEB SCRAPING

- Consiste à télécharger les pages web pour les analyser dans une deuxième phase
- Plusieurs outils existent déjà dont certains en ligne
  - Payants
  - Laissent des traces
- Un bon web scraper:
  - Utilise l'anonymisation à l'aide de proxy
  - Change l'agent (user-agent) fréquemment
  - Efface les cookies à chaque requête

LE WEB SCRAPING

```python
import requests

r = requests.get("https://bing.com")

print(r.text)
print(r.headers)
for cookie in r.cookies:
    print(cookie)

```

LE WEB SCRAPING

```
3  import requests
4
5  proxies = {"http":"120.234.135.251:9002"}
6  headers = {"user-agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36"}
7
8  r = requests.get("http://testing-ground.scraping.pro/whomai", proxies=proxies, headers=headers)
9
10 print(r.text)
11 for cookie in r.cookies:
12     print(cookie)
13
```

- Créer une liste de proxies et une fonction qui en rend un aléatoirement de la liste
- Créer une liste de user-agent et une fonction qui en rend un aléatoirement de la liste
- Effectuer votre requête en prenant un proxy et un user-agent aléatoire à chaque appel
- Sauvegarder la réponse dans un fichier, les cookies et les headers dans un autre

LE WEB SCRAPING

```python
from bs4 import BeautifulSoup

FILE = "_ob-s.xyz.html"

with open(FILE) as f:
    soup = BeautifulSoup(f, features="html.parser")

print(soup.title)
print("=" * 64)
print(soup.title.string)
print("=" * 64)
print("=" * 64)
print(soup.prettify())
print("=" * 64)
print(soup.get_text())
for script in soup.find_all("script"):
    print(script.get('src'))
    print("-" * 64)
print("=" * 64)
for cell in soup.find_all("td"):
    print(cell)
    print("-" * 64)
else:
    print("no cell")

```

- BeautifulSoup
  - Permet d'extraire les données
  - Permet de manipuler HTML
  - Ainsi de sauvegarder une page de phishing

MECHANIZE

- En plus des fonctions de bases couvertes par requests, mechanize permet de:
  - Manipuler les formulaires
  - Transmettre des formulaires valides programmatiquement
  - Ceci inclut les « hidden fields » ou ceux pour lesquels vous n'avez pas choisi de valeur en utilisant des valeurs par défaut

```
1  import mechanize
2
3  def viewPage(url):
4      browser = mechanize.Browser()
5      page = browser.open(url)
6      print("*" * 16 + " PAGE HTML " + "*" * 16)
7      source_code = page.read()
8      print (source_code)
9      print("*" * 16 + " END PAGE HTML " + "*" * 16)
10     print("*" * 16 + " FORM " + "*" * 16)
11     browser.select_form(name="form1")
12     print (browser.form)
13     print("*" * 16 + " END FORM " + "*" * 16)
14
15 viewPage('http://testphp.vulnweb.com/signup.php')
```