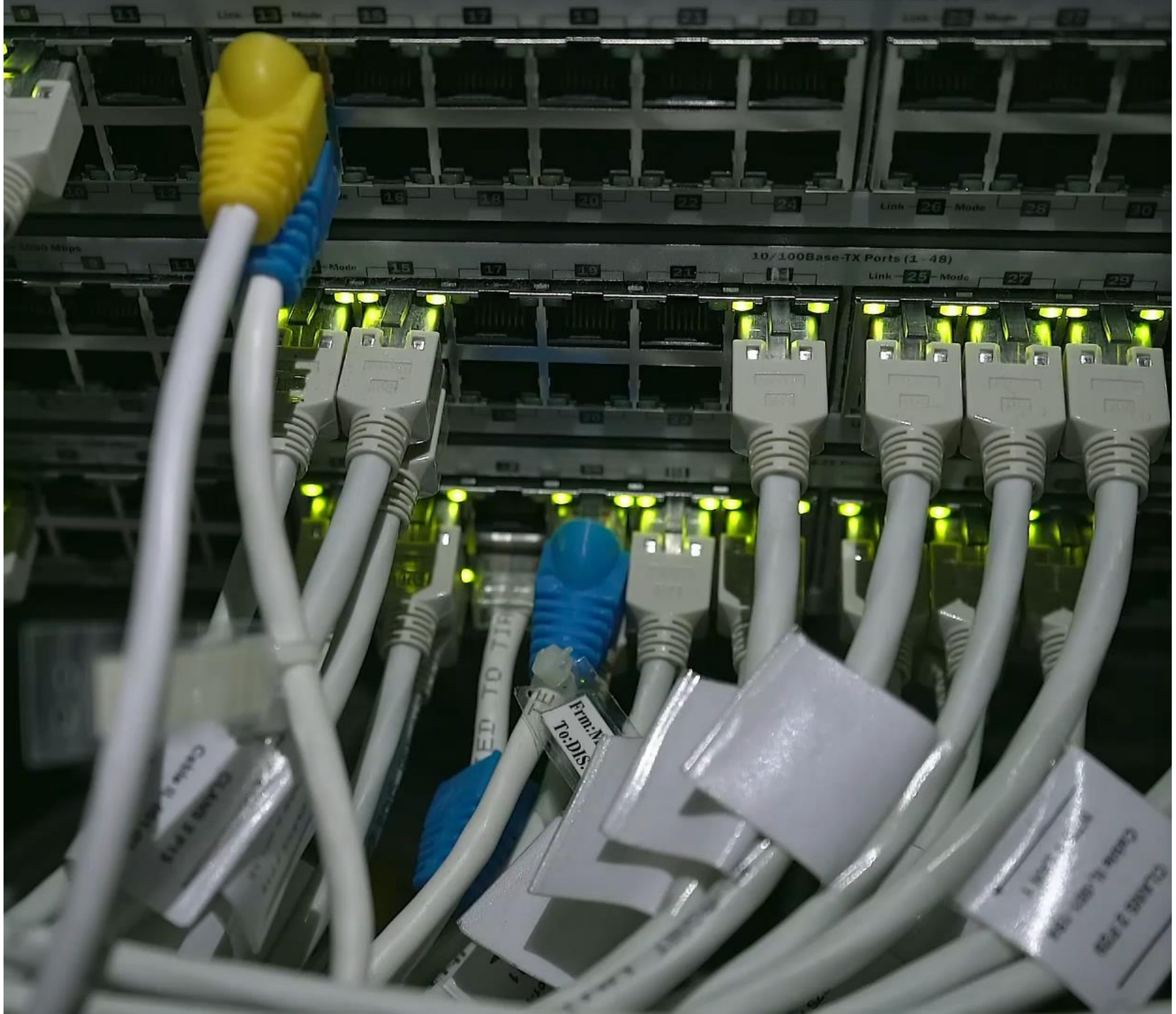
Prof. Ricardo Mesquita

Programação com Sockets



Sockets de Rede em Python

Sockets são componentes que permitem que se aproveite os recursos do sistema operacional para interagir com uma rede.

Pode-se considerá-los como um canal de comunicação ponto-a-ponto entre um cliente e um servidor.

Constituem uma forma simples de estabelecer comunicação entre processos na mesma máquina ou em máquinas diferentes.

O conceito de socket é muito semelhante ao uso de descritores de arquivos para sistemas operacionais UNIX.

Um socket é dado por um endereço IP e um número de porta.

Sockets de Rede em Python

- Quando duas aplicações ou processos interagem, eles utilizam um canal de comunicação específico.
- Sockets são as ***extremidades*** dos canais de comunicação.
- *Atenção*: podemos utilizar sockets para estabelecer um canal de comunicação *entre dois processos na mesma máquina*, *dentro de um processo*, ou *entre processos em máquinas diferentes*.

Sockets de Rede em Python

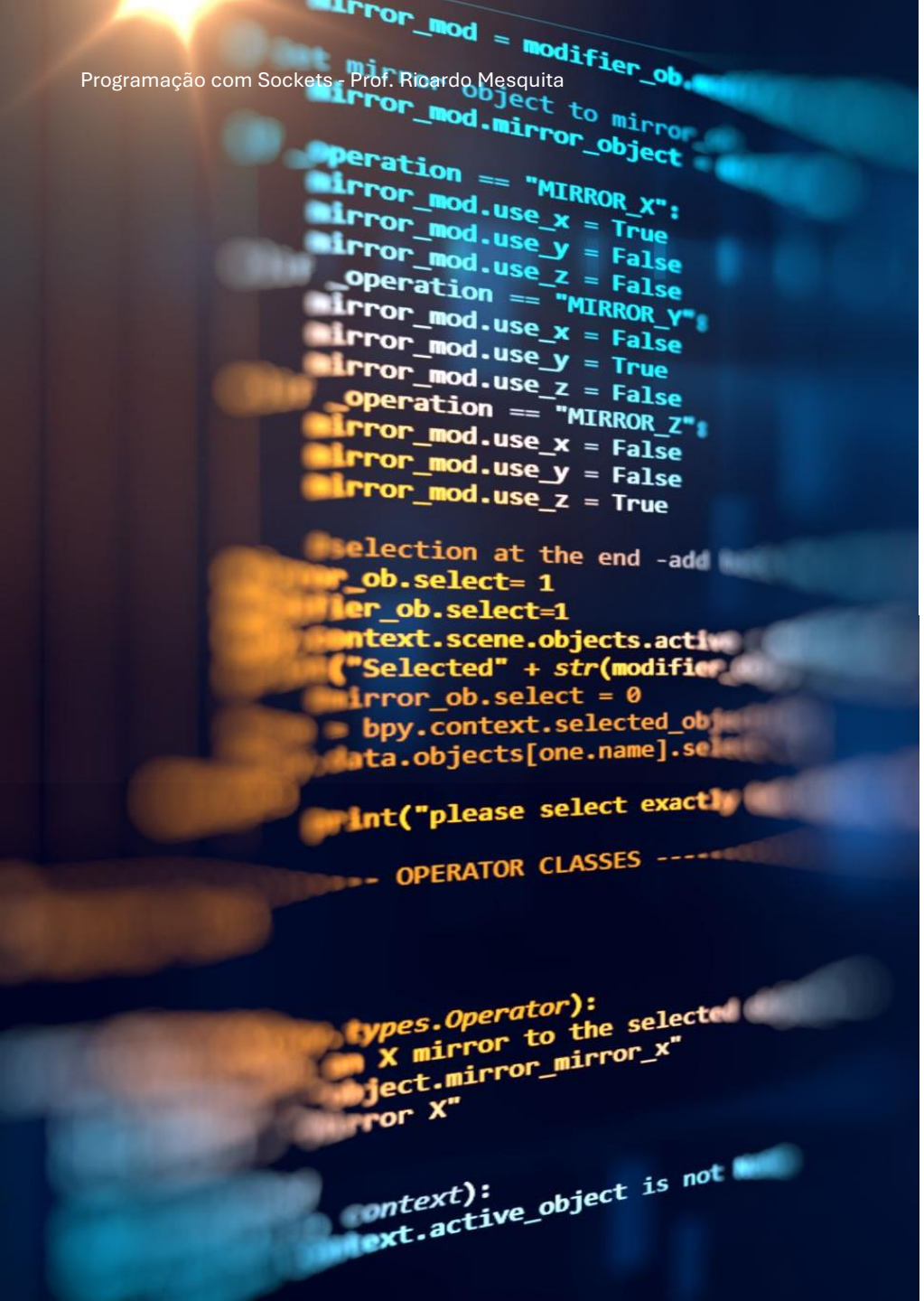
- A criação de um soquete em Python é feita através do método socket.socket().
- A sintaxe geral do método socket é a seguinte:

```
s = socket.socket(socket_family, socket_type, protocol=0)
```

- Note que a sintaxe anterior representa as famílias de endereços e o protocolo da camada de transporte.

Sockets de Rede em Python

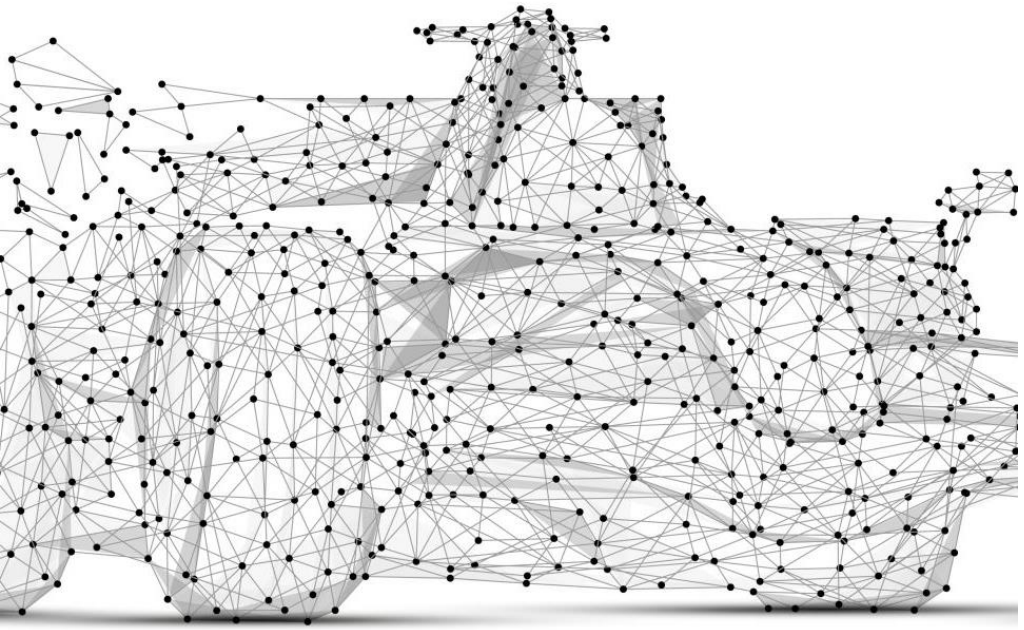
- Com base no tipo de comunicação, os soquetes são classificados da seguinte forma:
  - TCP sockets (`socket.SOCK_STREAM`)
  - UDP sockets (`socket.SOCK_DGRAM`)



Sockets de Rede em Python

- Os soquetes também podem ser categorizados por família. As seguintes opções estão disponíveis:
  - Sockets UNIX (`socket.AF_UNIX`), que foram criados antes da definição da rede e são baseados em dados.
  - `socket.AF_INET` para trabalhar com o protocolo IPv4.
  - `socket.AF_INET6` para trabalhar com o protocolo IPv6.

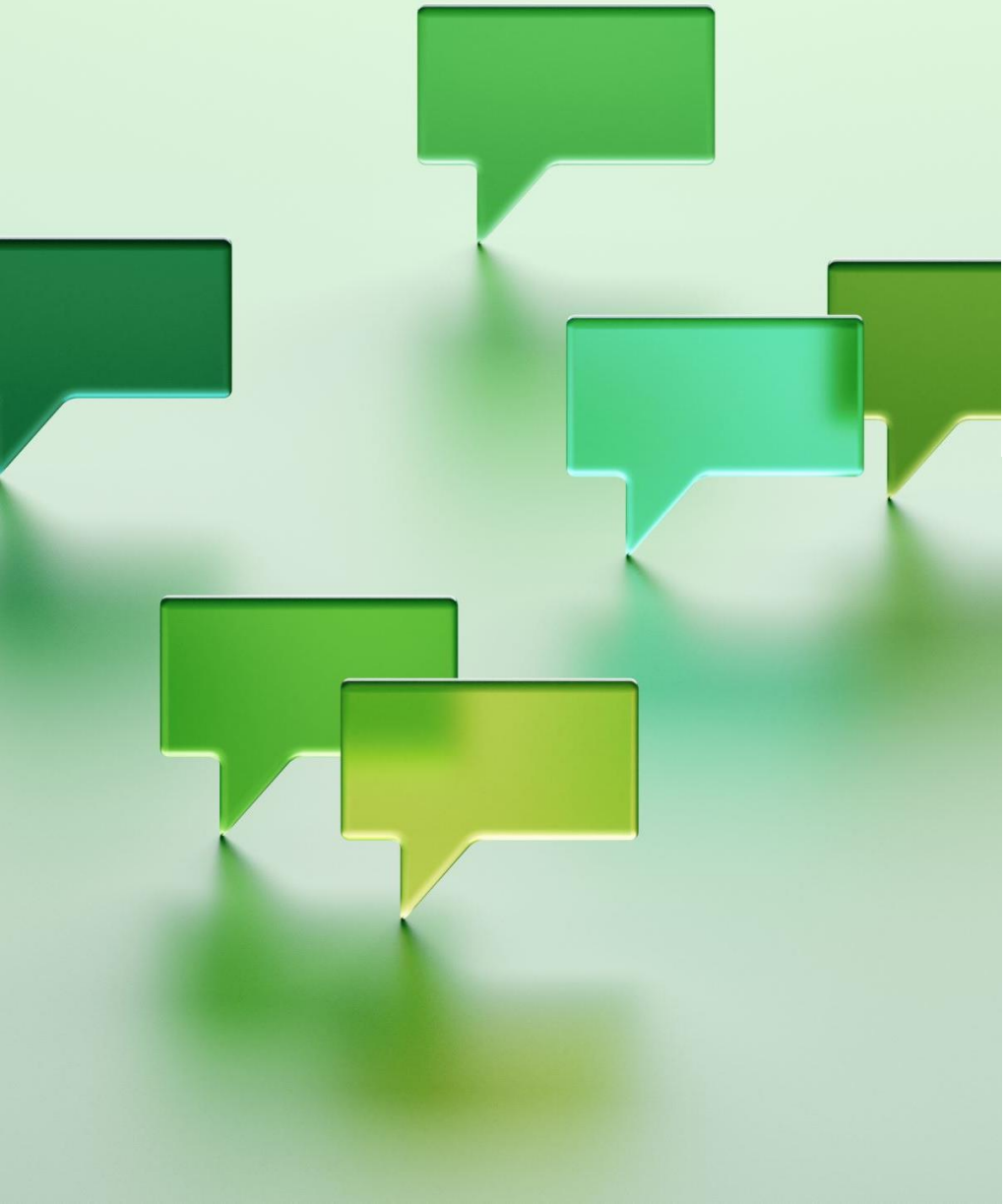
Sockets de Rede em Python



- Existe outro tipo de soquete chamado **raw socket** ("socket bruto").
- Estes soquetes permitem acesso aos protocolos de comunicação, tanto de camada 3 (nível de rede) quanto de camada 4 (nível de transporte.
- A utilização de sockets deste permite implementar novos protocolos e modificar os existentes, contornando os protocolos TCP/IP regulares.

Observação

- Para manipulação de pacotes de rede, acesse https://scapy.net.
- O **Scapy** módulo escrito em Python para manipulação de pacotes com suporte a múltiplos protocolos de rede.
- Esta ferramenta permite a criação e modificação de pacotes de rede de diversos tipos, implementando funções de captura e *sniffing* de pacotes.



O Módulo Socket

- Quando trabalha-se com sockets no desenvolvimento de aplicações, utiliza-se o conceito de cliente/servidor onde existem duas aplicações, uma atuando como servidor e outra como cliente, com ambas se comunicando através de passagem de mensagens usando protocolos como TCP ou UDP.
- O módulo socket fornece todas as funcionalidades necessárias para escrever clientes e servidores TCP e UDP.

O Módulo Socket

Para testar a instalação do módulo, faça:

```
>>> import socket
>>> dir(socket)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', '_blocking_errnos', '_intenum_converter', '_realsocket',
'_socket', 'close', 'create_connection', 'create_server', 'dup', 'errno', 'error',
'fromfd', 'gaierror', 'getaddrinfo', 'getdefaulttimeout', 'getfqdn', 'gethostbyaddr',
'gethostbyname', 'gethostbyname_ex', 'gethostname', 'getnameinfo', 'getprotobyname',
'getservbyname', 'getservbyport', 'has_dualstack_ipv6', 'has_ipv6', 'herror',
'htonl', 'htons', 'if_indextoname', 'if_nameindex', 'if_nametoindex', 'inet_aton',
'inet_ntoa', 'inet_ntop', 'inet_pton', 'io', 'ntohl', 'ntohs', 'os', 'selectors',
'setdefaulttimeout', 'sethostname', 'socket', 'socketpair', 'sys', 'timeout']
```

O Módulo Socket

- Note que na saída anterior, pode-se ver todos os métodos disponíveis no módulo.
- Dentre as constantes mais utilizadas, pode-se destacar as seguintes:
  - `socket.AF_INET`
  - `socket.SOCK_STREAM`

O Módulo Socket

- Para criar um socket em uma determinada máquina, chama-se o construtor da classe socket com a família, o tipo e o protocolo como parâmetros.
- Segue uma chamada típica para criar um socket:

```
>>> socket.socket(socket.AF_INET,socket.SOCK_STREAM)
```

O Módulo Socket

- Destacam-se os seguintes métodos para implementação de clientes e servidores:
  - **socket.accept()**: é usado para aceitar conexões e retorna um par de valores como (conn, endereço).
  - **socket.bind()**: é usado para vincular endereços especificados como parâmetro.
  - **socket.connect()**: é usado para conectar ao endereço especificado como parâmetro.
  - **socket.listen()**: é usado para escutar comandos no servidor ou cliente.
  - **socket.recv(buflen)**: é usado para receber dados do soquete.
    - O argumento do método indica a quantidade máxima de dados que pode receber.

O Módulo Socket

- Destacam-se os seguintes métodos para implementação de clientes e servidores:
  - **socket.recvfrom(buflen)**: é usado para receber dados e o endereço do remetente.
  - **socket.recv_into(buffer)**: é usado para receber dados em um buffer.
  - **socket.send(bytes)**: é usado para enviar bytes de dados para o destino especificado.
  - **socket.sendto(data, address)**: é usado para enviar dados para um determinado endereço.
  - **socket.sendall(data)**: é usado para enviar todos os dados do buffer para o soquete.
  - **socket.close()**: é usado para liberar memória e finalizar a conexão.

Métodos para Clientes de Servidores

Para o servidor:

- **socket.bind(address)**: Este método permite conectar o endereço ao socket, com a exigência de que o socket esteja aberto antes de estabelecer a conexão com o endereço.
- **socket.listen(count)**: Este método aceita como parâmetro o número máximo de conexões de clientes e inicia o listener TCP para conexões de entrada.
- **socket.accept()**: Este método permite aceitar conexões de clientes e retorna uma tupla com dois valores que representam client_socket e client_address.
  - É preciso chamar os métodos `socket.bind()` e `socket.listen()` antes de usar este método!

Métodos para Clientes de Servidores

Para o cliente:

- **socket.connect(ip_address)**: Este método conecta o cliente ao endereço IP do servidor.
- **socket.connect_ext(ip_address)**: Este método tem a mesma funcionalidade do método anterior e oferece a possibilidade de retornar um erro caso não consiga se conectar com aquele endereço.

Métodos para Clientes de Servidores

- O método `socket.connect_ex(address)` é muito útil para implementar varredura de portas com sockets.
- O script de exemplo a seguir mostra as portas que estão abertas em localhost com a interface de endereço IP de loopback 127.0.0.1.

Exemplo 1

```
import socket
ip ='127.0.0.1’
portlist = [21,22,23,80]
for port in portlist:
        sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        result = sock.connect_ex((ip,port))
        print(port,":", result)
        sock.close()
```

verifica as portas para serviços ftp, ssh, telnet e http na interface localhost.

Exemplo 1

- Ao lado, um exemplo saída do script anterior, onde o resultado para cada porta é um número que representa se a porta está aberta ou não.
- Nesta execução, a porta 80 retorna o valor 0, o que significa que a porta está aberta.
- Todas as outras portas retornam um valor diferente de zero, o que significa que as portas estão fechadas.

```
$ python socket_ports_open.py
21 : 111
22 : 111
23 : 111
80 : 0
```

Métodos para Clientes de Servidores

- Os sockets também podem ser usados para a comunicação com um servidor Web, um servidor de e-mails ou muitos outros tipos de servidores.
- Para isso, basta encontrar o documento que descreve o protocolo correspondente e escrever o código para enviar e receber os dados de acordo com esse protocolo.
- No exemplo a seguir, faremos uma conexão com um servidor Web que escuta na porta 80 e acessaremos uma rota específica dentro deste servidor para solicitar um documento de texto.

Exemplo 2

```
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('ftp.debian.org', 80))
cmd = 'GET http://ftp.debian.org/debian/README.mirrors.txt HTTP/1.0\r\n\r\n'.encode()
sock.send(cmd)
while True:
        data = sock.recv(512)
        if len(data) < 1: break
        print(data.decode(),end=‘’)
sock.close()
```

Exemplo 2

- A execução do script anterior começa com o cabeçalho que o servidor envia para descrever o documento.
- Por exemplo, o cabeçalho Content-Type indica que o documento é um documento de texto/simples.
- Depois que o servidor envia o cabeçalho, ele adiciona uma linha em branco para indicar o final do cabeçalho e então envia os dados do arquivo usando uma solicitação GET:

```
$ python socket_web_server.py
HTTP/1.1 200 OK
Connection: close
Content-Length: 86
Server: Apache
X-Content-Type-Options: nosniff
X-Frame-Options: sameorigin
Referrer-Policy: no-referrer
X-Xss-Protection: 1
Permissions-Policy: interest-cohort=()
Last-Modified: Sat, 04 Mar 2017 20:08:51 GMT
ETag: "56-549ed3b25abfb"
X-Clacks-Overhead: GNU Terry Pratchett
Content-Type: text/plain; charset=utf-8
Via: 1.1 varnish, 1.1 varnish
...
```

Coletando Informações com Sockets

Métodos úteis:

- **socket.gethostbyname(hostname)**: Este método retorna uma string convertendo um nome de host para o formato de endereço IPv4.
  - Este método é equivalente ao comando nslookup que podemos encontrar em alguns sistemas operacionais.
- **socket.gethostbyname_ex(name)**: Este método retorna uma tupla que contém um endereço IP para um nome de domínio específico.
  - Se virmos mais de um endereço IP, isso significa que um domínio é executado em vários endereços IP:
- **socket.getfqdn([domínio])**: É usado para encontrar o nome totalmente qualificado de um domínio.

Coletando Informações com Sockets

Métodos úteis:

- **socket.gethostbyaddr (ip_address)**: Este método retorna uma tupla com três valores (hostname, name, ip_address_list):
  - *hostname* representa o host que corresponde ao endereço IP fornecido; *name* é uma lista de nomes associados a este endereço IP; e *ip_address_list* é uma lista de endereços IP que estão disponíveis no mesmo host.
- **socket.getservbyname(servicename[, protocol_name])**: Este método permite obter o número da porta a partir do nome da porta.
- **socket.getservbyport(port[, protocol_name])**: Este método realiza a operação inversa ao anterior, permitindo obter o nome da porta a partir do número da porta.

Exemplo 3

```
import socket
try:
        hostname = socket.gethostname()
        print("gethostname:",hostname)
        ip_address = socket.gethostbyname(hostname)
        print("Local IP address: %s" %ip_address)
        print("gethostbyname:",socket.gethostbyname('www.python.org'))
        print("gethostbyname_ex:",socket.gethostbyname_ex('www.python.org'))
        print("gethostbyaddr:",socket.gethostbyaddr('8.8.8.8'))
        print("getfqdn:",socket.getfqdn('www.google.com'))
        print("getaddrinfo:",socket.getaddrinfo("www.google.com",None,0,socket.SOCK_STREAM))
except socket.error as error:
        print (str(error))
        print ("Connection error")
```

Exemplo 3

```
$ python socket_methods.py

gethostname: linux-hpelitebook8470p

Local IP address: 127.0.1.1

gethostbyname: 151.101.132.223

gethostbyname_ex: ('dualstack.python.map.fastly.net', ['www.python.org'],
['151.101.132.223’])

gethostbyaddr: ('dns.google', [], ['8.8.8.8’])

getfqdn: mad41s08-in-f4.1e100.net

getaddrinfo: [(<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '',
('142.250.178.164', 0)), (<AddressFamily.AF_INET6: 10>, <SocketKind.SOCK_STREAM: 1>, 6, '',
('2a00:1450:4003:807::2004', 0, 0, 0))]
```

Exemplo 4

```
import socket
def find_services_name():
    for port in [21,22,23,25,80]:
        print("Port: %s => service name: %s" %(port, socket.getservbyport(port, 'tcp')))
        print("Port: %s => service name: %s" %(53, socket.getservbyport(53, 'udp')))
if __name__ == '__main__’:
    find_services_name()
```

Exemplo 4

```
$ python socket_service_names.py
Port: 21 => service name: ftp
Port: 22 => service name: ssh
Port: 23 => service name: telnet
Port: 25 => service name: smtp
Port: 80 => service name: http
Port: 53 => service name: domain
```

Gerenciamento de Exceções

- Diferentes tipos de exceções são definidas na biblioteca de sockets do Python para diferentes erros.
- Para lidar com essas exceções, usam-se os blocos `try` e `accept`.
  - **socket.timeout**: exceções relacionadas à expiração dos tempos de espera.
  - **socket.gaierror**: erros durante a busca por informações sobre endereços IP.
  - **socket.error**: erros genéricos de entrada e saída e comunicação.
    - Este é um bloco genérico onde você pode capturar qualquer tipo de exceção.

Exemplo 5

```
import socket
host = "domain/ip_address"
port = 80
try:
        mysocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        print(mysocket)
        mysocket.settimeout(5)
except socket.error as error:
        print("socket create error: %s" %error)
try:
        mysocket.connect((host,port))
        print(mysocket)
except socket.timeout as error:
        print("Timeout %s" %error)
except socket.gaierror as error:
        print("connection error to the server:%s" %error)
except socket.error as error:
        print("Connection error: %s" %error)
```

Cliente com o Módulo Socket

- Uma vez apresentados os métodos para cliente e servidor, podemos testar como enviar e receber dados de um servidor.
- Assim que a conexão for estabelecida, podemos enviar e receber dados usando os métodos `send()` e `recv()` para comunicações TCP.
- Para comunicação UDP, poderíamos usar os métodos `sendto()` e `recvfrom()`.
- A seguir, um exemplo.

Exemplo 6


```
import socket
host = input("Enter host name: ")
port = int(input("Enter port number: "))
try:
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as socket_tcp:
        socket_tcp.settimeout(10)
        if (socket_tcp.connect_ex((host,port)) == 0):
            print("Established connection to the server %s in the port
                                                    %s" % (host, port))
            request = "GET / HTTP/1.1\r\nHost:%s\r\n\r\n" % host
            socket_tcp.send(request.encode())
            data = socket_tcp.recv(4096)
            print("Data:",repr(data))
            print("Length data:",len(data))
except socket.timeout as error:
    print("Timeout %s" %error)
except socket.gaierror as error:
    print("connection error to the server:%s" %error)
except socket.error as error:
    print("Connection error: %s" %error)
```

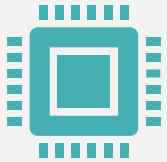
Cliente com o Módulo Socket

No script anterior:

- Usamos um bloco `try:except` para capturar uma exceção caso ele não consiga se conectar e exibir uma mensagem.
- Verificamos se a porta está aberta antes de fazer a solicitação e receber os dados do servidor.
- Criamos um objeto de socket TCP, conectamos o cliente ao host remoto e enviamos alguns dados.
- A última etapa é receber alguns dados de volta e imprimir a resposta.
- Para esta tarefa, usamos o método `recv()` do objeto socket para receber a resposta do servidor.

Escaneamento de Portas com Sockets

Temos ferramentas como o Nmap para verificar as portas abertas em uma máquina.



Poderíamos implementar funcionalidade semelhante para detectar portas abertas com vulnerabilidades em uma máquina de destino usando o módulo socket.

Implementado um *Port Scanner*

Os sockets são o alicerce fundamental para a comunicação de rede e, ao chamar o método connect_ex(), podemos testar facilmente se uma porta específica está aberta, fechada ou filtrada.

O código Python a seguir permite procurar portas abertas em um host local ou remoto.

O script procura portas selecionadas em um determinado endereço IP inserido pelo usuário e reflete as portas abertas de volta para o usuário.

Se a porta estiver bloqueada, também revela o motivo disso.

Exemplo 7

```
import socket
import sys
from datetime import datetime
import errno
remoteServer = input("Enter a remote host to scan: ")
remoteServerIP = socket.gethostbyname(remoteServer)
print("Please enter the range of ports you would like to scan on the machine")
startPort = input("Enter start port: ")
endPort = input("Enter end port: ")
print("Please wait, scanning remote host", remoteServerIP)
time_init = datetime.now()
```

Exemplo 7

- No código anterior, observa-se que o script procura obter informações relacionadas ao endereço IP e às portas da máquina alvo.
- Continuamos iterando por todas as portas usando um *loop* for de startPort a endPort para analisar cada porta intermediária.
- Concluímos o script mostrando o tempo total para concluir a verificação da porta:

```
try:
    for port in range(int(startPort),int(endPort)):
        print ("Checking port {} ...".format(port))
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(5)
        result = sock.connect_ex((remoteServerIP, port))
        if result == 0:
            print("Port {}: Open".format(port))
        else:
            print("Port {}: Closed".format(port))
            print("Reason:",errno.errorcode[result])
        sock.close()
except KeyboardInterrupt:
    print("You pressed Ctrl+C")
    sys.exit()
except socket.gaierror:
    print('Hostname could not be resolved. Exiting’)
    sys.exit()
except socket.error:
    print("Couldn't connect to server")
    sys.exit()
time_finish = datetime.now()
total = time_finish - time_init
print('Port Scanning Completed in: ', total)
```

Implementado um *Port Scanner*

- Note que o código anterior realizará uma varredura em cada uma das portas indicadas no host de destino.
- Para fazer isso, usa-se o método `connect_ex()` para determinar se estão abertas ou fechadas.
- Se esse método retornar 0 como resposta, a porta será classificada como Aberta.
- Caso retorne outro valor de resposta, a porta é classificada como Fechada e o código de erro retornado é exibido.
- Na execução do script anterior, pode-se ver as portas que estão abertas e o tempo, em segundos, de varredura completa das portas.
- A seguir uma possível saída.

Exemplo 7 - Saída

```
$ python socket_port_scanner.py
Enter a remote host to scan: scanme.nmap.org
Please enter the range of ports you would like to scan on the machine
Enter start port: 80
Enter end port: 82
Please wait, scanning remote host 45.33.32.156
Checking port 80 ...
Port 80: Open
Checking port 81 ...
Port 81: Closed
Reason: ECONNREFUSED
Port Scanning Completed in: 0:00:00.307595
```

Port Scanner mais Avançado

- O script Python a seguir nos permitirá verificar um endereço IP com as funções `portScanning` e `socketScan`.
- O programa busca as portas selecionadas em um domínio específico resolvido a partir do endereço IP informado pelo usuário por parâmetro.
- O usuário deverá introduzir como parâmetros obrigatórios o host e pelo menos uma porta ou uma lista de portas, cada uma separada por vírgula.

Exemplo 8

```
import optparse
from socket import *
from threading import *
def socketScan(host, port):
try:
    socket_connect = socket(AF_INET, SOCK_STREAM)
    socket_connect.settimeout(5)
    result = socket_connect.connect((host, port))
    print('[+] %d/tcp open' % port)
except Exception as exception:
    print('[-] %d/tcp closed' % port)
    print('[-] Reason:%s' % str(exception))
finally:
    socket_connect.close()
```

Continua...

Exemplo 8

```
def portScanning(host, ports):
    try:
        ip = gethostbyname(host)
    except:
        print("[-] Cannot resolve '%s': Unknown host" %host)
        return
    try:
        name = gethostbyaddr(ip)
        print('[+] Scan Results for: ' + ip + " " + name[0])
    except:
        print('[+] Scan Results for: ' + ip)
        for port in ports:
            t = Thread(target=socketScan,args=(ip,int(port)))
            t.start()
```

Port Scanner mais Avançado

- No script anterior implementando dois métodos que nos permitem escanear um endereço IP com os métodos portScanning e socketScan, onde podemos destacar a utilização de threads para lançar as diferentes solicitações para cada uma das portas a serem analisadas.
- A seguir, implementamos nosso método main():

Exemplo 8

```
def main():
   parser = optparse.OptionParser('socket_portScan '+ '-H <Host> -P <Port>’)
   parser.add_option('-H', dest='host', type='string', help='specify host’)
   parser.add_option('-P', dest='port', type='string', help='specify port[s]
                                                    separated by comma’)
   (options, args) = parser.parse_args()
   host = options.host
   ports = str(options.port).split(',’)
   if (host == None) | (ports[0] == None):
      print(parser.usage)
   exit(0)
   portScanning(host, ports)
if __name__ == '__main__’:
   main()
```

Port Scanner mais Avançado

- No código anterior, configuram-se os argumentos obrigatórios para execução do script.
- Quando esses parâmetros forem coletados, o método `portScanning` é chamado para resolver o endereço IP e o nome do host.
- Em seguida, o método `socketScan` é chamando para avaliar o estado da porta.
- Para executar o script anterior, é preciso passar como parâmetros o endereço IP ou domínio e a lista de portas separados por vírgula.
- Na execução do script anterior, podemos ver o status de todas as portas especificadas para o domínio scanme.nmap.org:

Exemplo 8 - Saída

```
$ python socket_advanced_port_scanner.py -H scanme.nmap.org -P 22,23,80,81
[+] Scan Results for: 45.33.32.156 scanme.nmap.org
[-] 23/tcp closed
[+] 80/tcp open
[-] Reason:[Errno 111] Connection refused
[+] 22/tcp open
[-] 81/tcp closed
[-] Reason:[Errno 111] Connection refused
```

Continua...