Reverse shell на Python. Осваиваем навыки работы с сетью на Python на примере обратного шелла

Автор: Алексей Кузнецов · Опубликовано: 2017-07-17 · Сложность: Средняя

Time to read: 14 min

Содержание статьи

- Переходим к практике
- Используем UDP
  - Сервер слушает
  - Отправляем клиенту
  - Отправляем клиенту
- Используем TCP
  - Сервер слушает
  - Отправляем клиенту
  - Тестируем
- Переносим скрипт на Windows
  - Отправляем клиенту (исправляем)
  - Отправляем клиенту (исправляем)
  - Тестируем
- Шлем серверу сообщение
  - Тестируем
- Заключение

В этой статье мы разберемся, как при помощи Python создавать простейший аналог двустороннего шифрованного. Инструментарием к нам. Эта задача могла встречаться на экзамене OSCP, но в реальной жизни, конечно, гораздо интереснее. Мы будем использовать библиотеки для сокета обмена, как при этом организовать всё взаимодействие, как обмениваться данными и как сделать, чтобы сервер не отваливался после первого соединения.

Существует два интерфейсных протокола. Не отлично выверенного ремесла и стопроцентного гарантированного доставки пакетов — это UDP. Более надежный и стабильный — TCP.

Протокол UDP рассылается для передачи данных и прием ответа с другого без проверки доставки. Если пакеты потеряются, то ничего страшного, не проверяем. Зато быстро и эффективно, а это нужно для передачи потоков аудио и видео, а также игр онлайн. В целом для передачи информации, которая не критична к потерям.

Иначе работает **TCP**. Он гарантирует доставку пакета подтверждение. Иначе, если пакеты потеряются, то TCP не передаст данные дальше. Если пакеты пришли не в порядке, TCP сам расставит их в нужный порядок. Если пакеты пришли с ошибкой, TCP попросит отправить заново. Медленно, но надежно. Подходит для передачи файлов, почты, веб-страниц и прочего, где нужна полная гарантия доставки.

Протокол TCP также рассылает сообщения, но при этом проверяет, что пакет дошел до получателя. Если пакет потерялся, то TCP его отправит заново.

Переходим к практике

Для начала нам нужно определиться с IP-адресом. Я использую Linux в Debian, поэтому смотрим адреса командой ip addr show. Нас интересует eth0:

```
$ ip addr show
```

Запоминаем наш адресок. Его нам нужно в скрипт, где клиентом обычно берут свои адреса, а серверу — адресовать. В дальнейшем мы его отредактируем, а пока берем и пишем. Для UDP-сервера в терминале запускаем nano udp_server.py, пишем:

```
$ nano -c udp_server.py
```

Теперь в nano пишем:

```
# udp_server.py
import socket, sys
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(("", 1234))
while 1:
    data, addr = s.recvfrom(1024)
    print "Connected by", addr
    s.sendto("Hello, " + addr[0] + ":" + str(addr[1]) + " - I received your data: " + data, addr)
```

Теперь в nano нажимаем Ctrl + X, Y, Enter — записали, что мы разрешаем нам использовать порт 1234 для прослушивания. Теперь пишем в терминале:

```
$ chmod +x udp_server.py
$ ./udp_server.py
```

Теперь в терминале у нас сервер слушает порт 1234 и ждет подключения. Для клиента — в новом окне или закладке пишем:

```
$ nano -c udp_client.py
```

Теперь в nano пишем:

```
# udp_client.py
import socket, sys
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto("Hello, server", ("127.0.0.1", 1234))
data, addr = s.recvfrom(1024)
print "Received from", addr, "data", data
```

Сохраняем: Ctrl + X, Y, Enter. Теперь запускаем:

```
$ chmod +x udp_client.py
$ ./udp_client.py
```

Данные клиентского компьютера сервер получил, отправил клиенту, тот получил. Теперь нужно сделать, чтобы сервер не отваливался после первого подключения. Для этого нужно ему сказать, что пакеты принимать бесконечно. Для этого в nano пишем:

```
# udp_server.py
import socket, sys
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(("", 1234))
while 1:
    data, addr = s.recvfrom(1024)
    print "Connected by", addr
    s.sendto("Hello, " + addr[0] + ":" + str(addr[1]) + " - I received your data: " + data, addr)
```

Сохраняем: Ctrl + X, Y, Enter. Запускаем:

```
$ chmod +x udp_server.py
$ ./udp_server.py
```

Теперь сервер слушает порт 1234 и ждет подключения. Для клиента — в новом окне или закладке пишем:

```
$ nano -c udp_client.py
```

Теперь в nano пишем:

```
# udp_client.py
import socket, sys
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto("Hello, server", ("127.0.0.1", 1234))
data, addr = s.recvfrom(1024)
print "Received from", addr, "data", data
```

Сохраняем: Ctrl + X, Y, Enter. Запускаем:

```
$ chmod +x udp_client.py
$ ./udp_client.py
```

Теперь клиентский скрипт отправил сообщение, получил ответ и завершил работу. Сервер же продолжает слушать порт 1234 и ждать новых подключений