

Оглавление

Предисловие	16
Введение	18
Об авторе	20
О научном редакторе	21
Вступление	22
Кому подойдет эта книга	23
Какие темы здесь освещаются	23
Как извлечь максимум из этой книги	25
Загрузка файлов с примерами кода	25
Загрузка полноцветных иллюстраций	26
Условные обозначения	26
От издательства	26
Глава 1. Обзор TCP/IP и Python	27
Краткий обзор интернета	29
Серверы, хосты и сетевые компоненты	30
Появление дата-центров	31
Модель OSI	35
Клиент-серверная модель	37
Наборы сетевых протоколов	38
Протокол управления передачей (TCP)	39
Протокол пользовательских датаграмм (UDP)	40
Межсетевой протокол (IP)	41
Обзор языка Python	43
Версии Python	45
Операционные системы	46

Выполнение программы на Python	47
Встроенные в Python типы данных	48
Операторы в Python	54
Средства управления потоком выполнения в Python	55
Функции в Python	56
Классы в Python	57
Модули и пакеты в Python	58
Резюме	60
Глава 2. Низкоуровневое взаимодействие с сетевыми устройствами	61
Трудности работы с CLI	62
Создание виртуальной лаборатории	64
Физические устройства	64
Виртуальные устройства	65
Cisco VIRL	66
Cisco DevNet и dCloud	71
GNS3	73
Библиотека Python Pexrest	75
Виртуальная среда Python	75
Установка Pexrest	76
Краткий обзор Pexrest	76
Наша первая программа на основе Pexrest	81
Другие возможности Pexrest	82
Pexrest и SSH	84
Итоговая программа на основе Pexrest	85
Библиотека Python Paramiko	86
Установка Paramiko	86
Краткий обзор Paramiko	87
Наша первая программа, написанная с использованием Paramiko	90
Другие возможности Paramiko	91
Итоговая программа на основе Paramiko	93
Библиотека Netmiko	94
Фреймворк Nornir	96
Недостатки Pexrest и Paramiko по сравнению с другими инструментами ...	98
Резюме	99

Глава 3. API и IDN-сети	100
Инфраструктура как код	101
Сети, ориентированные на намерения	102
Консольный вывод и структурированные результаты API-запроса	103
Моделирование данных для IaC	106
YANG и NETCONF	108
API и платформа ACI от Cisco	108
Cisco NX-API	109
Модель Cisco YANG	115
Cisco ACI и APIC-EM	116
Контроллер Cisco Meraki	119
API на языке Python для Juniper Networks	120
Juniper и NETCONF	121
Juniper PyEZ для разработчиков	125
API на языке Python для устройств Arista	130
Работа с eAPI от Arista	130
Библиотека Arista Pyeapi	135
Пример работы с VyOS	140
Другие библиотеки	141
Резюме	141
Глава 4. Основы Ansible	142
Ansible: более декларативный фреймворк	143
Короткий пример с Ansible	146
Установка управляющего узла	146
Установка разных версий Ansible из исходного кода	147
Подготовка лаборатории	148
Ваш первый сценарий Ansible	149
Преимущества Ansible	153
Отсутствие агентов	154
Идемпотентность	155
Простота и расширяемость	155
Поддержка от производителей сетевого оборудования	156
Архитектура Ansible	158
YAML	159
Файлы реестров	159

Переменные	161
Шаблоны Jinja2	165
Сетевые модули Ansible	165
Локальные соединения и факты	166
Переменная provider	166
Пример Ansible с устройствами Cisco	168
Пример сценария для Ansible 2.8	171
Пример Ansible с устройствами Juniper	174
Пример Ansible с устройствами Arista	175
Резюме	176
Глава 5. Ansible: следующий уровень	177
Подготовка лаборатории	178
Условные выражения в Ansible	178
Выражение when	179
Факты о сетевых устройствах в Ansible	181
Условные выражения в сетевых модулях	184
Циклы в Ansible	185
Стандартные циклы	186
Циклический перебор словарей	188
Шаблоны	190
Переменные в шаблонах Jinja2	192
Циклы в Jinja2	193
Условные выражения в Jinja2	193
Переменные групп и хостов	196
Переменные группы	196
Переменные хоста	197
Ansible Vault	198
Подключение файлов и роли в Ansible	200
Инструкции include в Ansible	201
Роли Ansible	202
Написание собственного модуля	206
Ваш первый модуль	206
Ваш второй модуль	208
Резюме	210

Глава 6. Сетевая безопасность с использованием Python	211
Подготовка лаборатории	212
Python Scapy	216
Установка Scapy	216
Интерактивные примеры	218
Захват пакетов с помощью Scapy	220
Сканирование TCP-портов	221
Коллекция пакетов для проверки связи	225
Распространенные атаки	226
Ресурсы о Scapy	226
Списки доступа	227
Реализация списков доступа с помощью Ansible	228
Списки доступа по MAC-адресам	231
Поиск в Syslog	233
Поиск с помощью модуля регулярных выражений	234
Другие инструменты	236
Приватные VLAN	236
UFW и Python	237
Дополнительный материал	238
Резюме	239
Глава 7. Сетевой мониторинг с использованием Python: часть 1	240
Подготовка лаборатории	241
SNMP	242
Подготовка	244
PySNMP	246
Python для визуализации данных	251
Matplotlib	252
Pygal	259
Работа с Sacti в Python	264
Установка	265
Сценарий на Python в качестве источника данных	267
Резюме	269
Глава 8. Сетевой мониторинг с использованием Python: часть 2	270
Graphviz	271
Подготовка лаборатории	272

Установка	274
Примеры работы с Graphviz	274
Примеры с Graphviz и Python	277
Создание графа ближайших соседей с помощью LLDP	278
Потоковый мониторинг	287
Разбор NetFlow с помощью Python	288
Мониторинг трафика с помощью ntop	293
Расширение ntop с помощью Python	296
sFlow	300
Резюме	304
Глава 9. Создание сетевых веб-сервисов с помощью Python	305
Сравнение веб-фреймворков для Python	307
Flask и подготовка лаборатории	309
Введение в фреймворк Flask	310
Клиент HTTPie	312
Маршрутизация URL	313
URL-переменные	314
Генерация URL	316
Возвращение результата с помощью jsonify	317
API для сетевых ресурсов	318
Flask-SQLAlchemy	318
API для работы с содержимым сети	320
API для работы с устройствами	323
API для работы с отдельными устройствами	325
Динамические сетевые операции	326
Асинхронные операции	328
Аутентификация и авторизация	331
Выполнение Flask в контейнерах	333
Резюме	337
Глава 10. Облачные сетевые технологии AWS	338
Подготовка к работе с AWS	339
AWS CLI и Python SDK	340
Обзор сети AWS	344
Виртуальное частное облако	351
Таблицы и цели маршрутизации	355

Автоматизация с использованием CloudFormation	357
Группы безопасности и списки доступа к сети	361
Elastic IP	363
NAT-шлюзы	364
Direct Connect и VPN	366
VPN-шлюзы	366
Direct Connect	367
Сервисы для масштабирования сетей	368
Elastic Load Balancing	369
Сервис Route 53 DNS	370
Доставка содержимого с использованием CloudFront	370
Другие сетевые сервисы от AWS	371
Резюме	371
Глава 11. Облачные сетевые технологии Azure	373
Сравнение сетевых сервисов в Azure и AWS	374
Подготовка к работе с Azure	375
Администрирование Azure и API	378
Субъекты-службы в Azure	381
Сравнение Python и PowerShell	383
Глобальная инфраструктура Azure	384
Виртуальные сети Azure	386
Доступ к интернету	389
Создание сетевых ресурсов	392
Конечные точки сервисов для VNet	394
VNet-пиринг	395
Маршрутизация в виртуальных сетях	397
Сетевые группы безопасности	402
Azure VPN	405
Azure ExpressRoute	408
Сетевые балансировщики нагрузки в Azure	409
Другие сетевые сервисы Azure	411
Резюме	411
Глава 12. Анализ сетевых данных с помощью Elastic Stack	412
Что такое Elastic Stack	413
Топология лаборатории	415

Elastic Stack как услуга	420
Первый полный пример	421
Elasticsearch и клиент на языке Python	425
Прием данных с помощью Logstash	427
Прием данных с использованием Beats	430
Поиск с помощью Elasticsearch	435
Визуализация данных с использованием Kibana	440
Резюме	445
Глава 13. Работа с Git	446
Git и разные аспекты управления контентом	447
Введение в Git	448
Преимущества Git	449
Терминология Git	450
Git и GitHub	451
Подготовка Git к работе	451
Gitignore	452
Примеры работы с Git	454
Ветви в Git	458
Пример работы с GitHub	460
Git и Python	467
GitPython	467
PyGitHub	468
Автоматизация резервного копирования конфигурационных файлов	470
Совместная работа с использованием Git	472
Резюме	473
Глава 14. Непрерывная интеграция с помощью Jenkins	474
Традиционный процесс управления изменениями	475
Введение в непрерывную интеграцию	477
Установка Jenkins	478
Пример с Jenkins	481
Первое задание для сценария на Python	481
Плагины Jenkins	487
Пример непрерывной интеграции в контексте сетевых технологий	489
Jenkins и Python	496

Непрерывная интеграция в контексте администрирования сети	497
Резюме	498
Глава 15. TDD для сетей	499
Обзор разработки через тестирование	500
Разные виды тестов	501
Топология как код	502
Модуль unittest	507
Еще о тестировании в Python	510
Примеры с pytest	511
Написание тестов для сетей	513
Тестирование доступности	514
Тестирование задержек сети	515
Тестирование безопасности	516
Тестирование транзакций	517
Тестирование сетевой конфигурации	517
Тестирование сценариев Ansible	518
Интеграция pytest с Jenkins	519
Интеграция с Jenkins	519
pyATS и Genie	524
Резюме	527

6

Сетевая безопасность с использованием Python

Писать на тему сетевой безопасности очень непросто. И проблема не в технической сложности, а в правильном выборе охватываемой области. Сфера сетевой безопасности чрезвычайно обширна и пронизывает все семь уровней модели OSI. Речь может идти и о перехвате информации на физическом уровне, и об уязвимостях в транспортных протоколах, и об атаках вида «человек посередине» на прикладном уровне. Эта проблема усложняется обнаружением все новых уязвимостей, которые, как иногда может показаться, становятся чем-то обыденным. И это не говоря уже о таком аспекте сетевой безопасности, как социальная инженерия.

По этой причине я бы хотел сразу определить рамки обсуждения в этой главе. Как и прежде, основное внимание будет уделено использованию Python для защиты сетевых устройств на сетевом и транспортном уровнях модели OSI. Мы рассмотрим инструменты на языке Python, подходящие для обеспечения безопасности отдельных сетевых устройств, а также использование Python для связывания разных компонентов. Надеюсь, применение Python на разных уровнях модели OSI позволит нам выработать общий подход к сетевой безопасности.

Эта глава охватывает такие темы, как:

- подготовка лаборатории;
- тестирование безопасности с помощью Python Scapy;
- списки доступа;

- ретроспективный анализ на основе Syslog и UFW (*Uncomplicated Firewall*) с использованием Python;
- другие инструменты: списки фильтрации MAC-адресов, приватные интерфейсы VLAN и Python-пакет для работы с Iptables.

Подготовка лаборатории

Здесь нам понадобятся устройства несколько другого рода, чем те, с которыми мы имели дело ранее. В предыдущих главах мы изолировали устройство, чтобы сосредоточиться на теме. В этой главе наша лаборатория будет содержать чуть больше устройств, что позволит нам проиллюстрировать возможности инструментов. Важна будет информация о соединении и операционной системе, так как на ее основе мы выбираем средства безопасности. Например, если мы хотим применить список доступа, чтобы защитить сервер, нам нужно иметь представление о топологии сети и о том, откуда устанавливается клиентское соединение. Соединения с хостами под управлением Ubuntu немного отличаются от тех, которые мы видели до сих пор, поэтому позже при необходимости вы можете сверяться с этим разделом.

Мы используем тот же инструмент Cisco VIRL с четырьмя узлами: двумя хостами и двумя сетевыми устройствами. Если вам нужно освежить навыки работы с Cisco VIRL, перечитайте главу 2, в которой мы познакомились с этой системой (рис. 6.1).

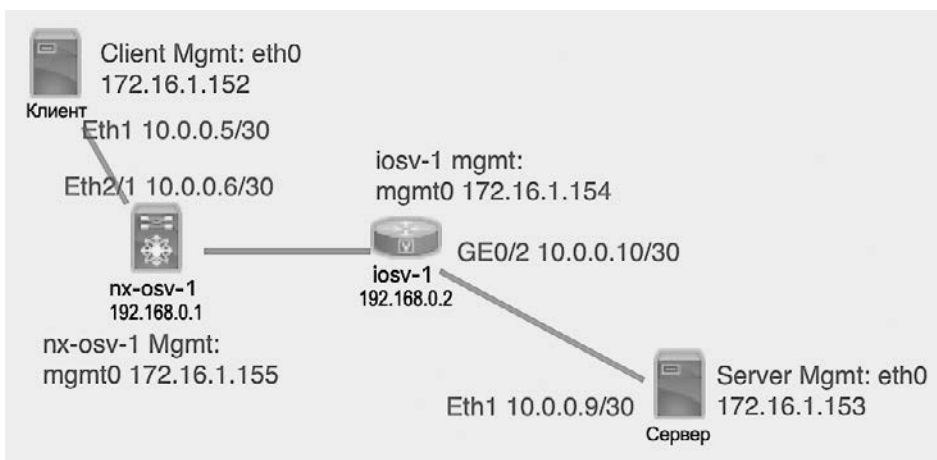


Рис. 6.1. Топология лаборатории



IP-адреса в вашей лаборатории могут отличаться от указанных выше. Мы привели их для того, чтобы вам было легче ориентироваться в примерах кода, представленных в этой главе.

Следуя этой иллюстрации, мы переименуем верхний и нижний хосты в Client и Server соответственно. Это похоже на то, как интернет-клиент пытается обратиться к корпоративному серверу внутри нашей сети. Мы снова выберем пункт Shared flat network (Разделяемая плоская сеть) в списке Management Network (Управляющая сеть), чтобы получить возможность управлять устройствами по дополнительному каналу (рис. 6.2).

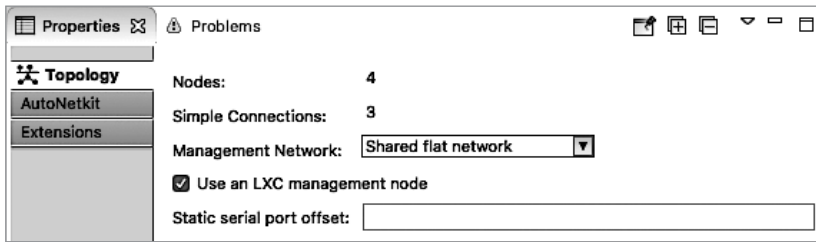


Рис. 6.2. Вариант управляющей сети в лаборатории



В этом примере клиентский хост должен быть доступен извне через VMnet2. В своей лаборатории я использую внешний USB-интерфейс, подключенный к моему хосту ESXi, предоставляющему такое соединение. Вам также может понадобиться добавить внешние DNS-серверы в /etc/resolvconf/resolv.conf.d/base:

```
cisco@Client:~$ cat /etc/resolvconf/resolv.conf.d/
base
nameserver 8.8.8.8
nameserver 8.8.4.4
```

В двух коммутаторах в качестве протокола внутреннего шлюза (Interior Gateway Protocol, IGP) используется алгоритм маршрутизации OSPF (Open Shortest Path First), и оба устройства находятся в зоне 0. Протокол BGP включен по умолчанию, и в обоих случаях используется AS 1.

Согласно сгенерированной конфигурации, интерфейсы, соединенные с хостами под управлением Ubuntu, находятся в OSPF-зоне 1, поэтому они представлены как межзональные маршруты (inter-area routes). Ниже показана конфигурация для NX-OSv (устройства IOSv имеют аналогичные конфигурацию и вывод):

```

interface Ethernet2/1
  description to Client
  no switchport
  mac-address fa16.3e00.0001
  ip address 10.0.0.6/30
  ip router ospf 1 area 0.0.0.0
  no shutdown
!
interface Ethernet2/2
  description to iosv-1
  no switchport
  mac-address fa16.3e00.0002
  ip address 10.0.0.14/30
  ip router ospf 1 area 0.0.0.0
  no shutdown
!
nx-osv-1# sh ip route
<опущено>
10.0.0.8/30, ubest/mbest: 1/0
    *via 10.0.0.13, Eth2/2, [110/41], 14:10:28, ospf-1, intra
192.168.0.2/32, ubest/mbest: 1/0
    *via 10.0.0.13, Eth2/2, [110/41], 14:10:28, ospf-1, intra
<опущено>

```

Ниже можно видеть OSPF-соседа и вывод BGP для NX-OSv (IOSv имеет похожий вывод):

```

nx-osv-1# sh ip ospf neighbors
  OSPF Process ID 1 VRF default
  Total number of neighbors: 1
  Neighbor ID      Pri State                Up Time  Address           Interface
  192.168.0.2      1 FULL/DR              14:12:31 10.0.0.13         Eth2/2
!
nx-osv-1# sh ip bgp summary
BGP summary information for VRF default, address family IPv4 Unicast
BGP router identifier 192.168.0.1, local AS number 1
BGP table version is 5, IPv4 Unicast config peers 1, capable peers 1
2 network entries and 2 paths using 288 bytes of memory
BGP attribute entries [2/288], BGP AS path entries [0/0]
BGP community entries [0/0], BGP clusterlist entries [0/0]

Neighbor      V   AS MsgRcvd MsgSent  TblVer  InQ  OutQ  Up/Down
State/PfxRcd
192.168.0.2   4   1   936    857     5     0     0 14:12:33 1

```

Хосты в нашей сети работают под управлением ОС Ubuntu 16.04; она похожа на Ubuntu VM 18.04, которую мы использовали до этого момента:

```

cisco@Server:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description: Ubuntu 16.04.3 LTS
Release: 16.04
Codename: xenial

```

Оба хоста с Ubuntu оснащены двумя сетевыми интерфейсами, `eth0` и `eth1`. Первый соединен с управляющей сетью (172.16.1.0/24), а второй — с сетевыми устройствами (10.0.0.x/30). Маршруты, ведущие к петлевому адресу устройства, напрямую соединены с сетевым блоком, и сети удаленных хостов статически направлены к `eth1` так, чтобы маршрут по умолчанию вел к управляющей сети:

```
cisco@Client:~$ route -n
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use
Iface
0.0.0.0          172.16.1.254   0.0.0.0        UG    0      0      0
eth0
10.0.0.0         10.0.0.6       255.0.0.0      UG    0      0      0
eth1
10.0.0.4         0.0.0.0        255.255.255.252 U    0      0      0
eth1
172.16.1.0       0.0.0.0        255.255.255.0  U    0      0      0
eth0
192.168.0.0      10.0.0.6       255.255.255.248 UG    0      0      0
eth1
```

Чтобы проверить путь от клиента к серверу, используем утилиту `ping` и отследим маршрут. Так мы убедимся в том, что трафик между нашими хостами проходит через сетевые устройства, а не по стандартному маршруту:

```
# IP-адрес серверного интерфейса Eth1 10.0.0.9/30
cisco@Server:~$ ifconfig eth1
eth1      Link encap:Ethernet  HWaddr fa:16:3e:68:bb:ce
          inet addr:10.0.0.9  Bcast:10.0.0.11  Mask:255.255.255.252
<опущено>

# эхо-запрос с IP-адреса клиентского интерфейса Eth1 (10.0.0.5/30) на сервер
cisco@Client:~$ ifconfig eth1
eth1      Link encap:Ethernet  HWaddr fa:16:3e:7c:75:ec
          inet addr:10.0.0.5  Bcast:10.0.0.7  Mask:255.255.255.252
<опущено>

cisco@Client:~$ ping -c 1 10.0.0.9
PING 10.0.0.9 (10.0.0.9) 56(84) bytes of data.
64 bytes from 10.0.0.9: icmp_seq=1 ttl=62 time=7.00 ms
--- 10.0.0.9 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 7.007/7.007/7.007/0.000 ms

# Трассируем маршрут от клиента к серверу
cisco@Client:~$ traceroute 10.0.0.9
traceroute to 10.0.0.9 (10.0.0.9), 30 hops max, 60 byte packets
 1 10.0.0.6 (10.0.0.6)  5.694 ms  10.632 ms  10.599 ms
 2 10.0.0.13 (10.0.0.13) 13.078 ms  19.132 ms  19.101 ms
 3 10.0.0.9 (10.0.0.9) 14.929 ms  19.026 ms  19.004 ms
```

Отлично! Мы настроили лабораторию; теперь познакомимся с некоторыми средствами и мерами безопасности на основе Python.

Python Scapy

Scapy (<https://scapy.net>) — это мощная интерактивная программа на языке Python для генерации сетевых пакетов. Если не считать некоторые дорогие коммерческие аналоги, существует не так много инструментов с похожими возможностями, насколько мне известно. Это один из моих любимых проектов в мире Python.

Основное преимущество программы Scapy: она позволяет создать собственный сетевой пакет на очень низком уровне. По словам создателя Scapy:

«Scapy — мощная интерактивная программа для управления пакетами. Она способна конструировать или декодировать пакеты самых разных протоколов, отправлять их по сети, перехватывать, сопоставлять запросы и ответы и многое другое... большинство других инструментов не позволяет создавать то, что не предусмотрено их авторами. Эти инструменты были созданы для конкретной цели и не могут отклоняться от нее».

Познакомимся с этим инструментом.

Установка Scapy

В попытке внедрить поддержку Python 3 проект Scapy пошел по интересному пути. В 2015 году для реализации поддержки Python 3 от Scapy 2.2.0 отпочковалась независимая версия, получившая название Scapy3k. В этой книге мы используем основную кодовую базу оригинального проекта Scapy. Если вы читали предыдущие издания и использовали версию Scapy, совместимую только с Python 2, то посмотрите, как разные выпуски этого инструмента поддерживают Python 3 (рис. 6.3).

Поскольку мы хотим генерировать пакеты на клиенте и отправлять их на сервер, Scapy следует установить на клиентской стороне:

```
cisco@Client:~$ git clone github.com/secdev/scapy.git
cisco@Client:~$ cd scapy/
cisco@Client:~/scapy$ sudo python3 setup.py install
```

Вслед за установкой мы запустим интерактивную оболочку Scapy, введя в командной строке scapy (рис. 6.4).

Python versions support

Scapy version	Python 2		Python 3		
	Python 2.5-2.6	Python 2.7	Python 3.4-3.6	Python 3.7	Python 3.8
2.2.X					
2.3.3					
2.4.0					
2.4.2					

Рис. 6.3. Поддержка версий Python (источник: <https://scapy.net/download/>)



Подробнее о том, как поддержка Python 3 появилась в Scapy. Все началось с ответвления независимой версии от Scapy 2.2.0 в 2015 году. Проект был назван Scapy3k. Это ответвление разошлось с основной кодовой базой Scapy. В первом издании данной книги история на этом заканчивалась. Позже возникло недоразумение по поводу пакета python3-scapy в PyPI и официальной поддержки Scapy. Основная цель данной главы — изучение этого инструмента. Поэтому я решил использовать более старую его версию, основанную на Python 2.

```

cisco@Client:~/scapy$ sudo scapy
INFO: Can't import matplotlib. Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
INFO: Can't import python-cryptography v1.7+. Disabled WEP decryption/encryption. (Dot11)
INFO: Can't import python-cryptography v1.7+. Disabled IPsec encryption/authentication.
WARNING: IPython not available. Using standard Python shell instead.
AutoCompletion, History are disabled.

      aSPY//YASa
      apyyyyCY/////////YCa      |
      sY////////YSpcs  scpCY//Pp  | Welcome to Scapy
ayp ayyyyyySCP//Pp      syY//C      | Version 2.4.3.dev61
AYAsAYYYYYYYY//Ps      cY//S      |
      pCCCCY//p      cSSps y//Y      | https://github.com/secdev/scapy
      SPPPP//a      pP//AC//Y      |
      A//A      cyP////C      | Have fun!
      p//Ac      sC//a      |
      P////YCpc      A//A      | Craft packets before they craft
      sccccp//pSP//p      p//Y      | you.
      sY/////////y caa      S//P      | -- Socrate
      cayCyayP//Ya      pY/Ya      |
      sY/PsY////////YCc      aC//Yp      |
      sc sccaCY//PCyaaPyCP//YSs
      spCPY////////YPSps
      ccaacs
>>>

```

Рис. 6.4. Тестирование Python Scapy

Простая проверка доступности библиотеки Scapy в Python 3:

```
cisco@Client:~$ python3
Python 3.5.2 (default, Jul 10 2019, 11:58:48)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> exit()
```

Отлично! Пакет Scapy установлен и доступен для использования в интерпретаторе Python. В следующем разделе вы увидите, как можно использовать интерактивную оболочку.

Интерактивные примеры

В первом примере мы сгенерируем на клиенте ICMP-пакет и отправим его серверу. На серверной стороне мы воспользуемся утилитой `tcpdump` с фильтром хостов, чтобы увидеть этот пакет:

```
## Сторона клиента
cisco@Client:~/scapy$ sudo scapy
>>> send(IP(dst="10.0.0.9")/ICMP())
.
Sent 1 packets.

# Сторона сервера
cisco@Server:~$ sudo tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol
decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
17:19:24.812184 IP 10.0.0.5 > 10.0.0.9: ICMP echo request, id 0, seq 0,
length 8
17:19:24.812205 IP 10.0.0.9 > 10.0.0.5: ICMP echo reply, id 0, seq 0,
length 8
```

Как видите, это очень простой процесс. Scapy позволяет сформировать пакет, добавляя заголовки протоколов через косую черту (/).

Функция `send` работает на сетевом уровне, поэтому вам не нужно беспокоиться о маршрутизации и физической адресации. У нее есть альтернатива, `sendp()`, которая работает на канальном уровне; это означает, что вам нужно будет указать интерфейс и протокол соединения.

Попробуем захватить ответный пакет с помощью функции `send-request (sr)`. Воспользуемся специальной версией `sr` с именем `sr1`, которая возвращает только один пакет:

```
>>> p = sr1(IP(dst="10.0.0.9")/ICMP())
Begin emission:
.Finished sending 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
>>> p
<IP version=4 ihl=5 tos=0x0 len=28 id=44710 flags= frag=0 ttl=62
proto=icmp chksum=0xba2d src=10.0.0.9 dst=10.0.0.5 |<ICMP type=echoreply
code=0 chksum=0xffff id=0x0 seq=0x0 |>>
```

Функция `sr()` возвращает кортеж со списками пакетов, на которые был и не был получен ответ:

```
>>> p = sr(IP(dst="10.0.0.9")/ICMP())
.Begin emission:
....Finished sending 1 packets.
*
Received 7 packets, got 1 answers, remaining 0 packets
>>> type(p)
<class 'tuple'>
```

Заглянем внутрь этого кортежа:

```
>>> ans, unans = sr(IP(dst="10.0.0.9")/ICMP())
.Begin emission:
...Finished sending 1 packets.
..*
Received 7 packets, got 1 answers, remaining 0 packets
>>> type(ans)
<class 'scapy.plist.SndRcvList'>
>>> type(unans)
<class 'scapy.plist.PacketList'>
```

Если вывести список пакетов, на которые был получен ответ, то мы увидим, что это еще один кортеж, содержащий отправленные и полученные пакеты:

```
>>> for i in ans:
...     print(type(i))
...
<class 'tuple'>
>>>
>>>
>>> for i in ans:
...     print(i)
...
(<IP frag=0 proto=icmp dst=10.0.0.9 |<ICMP |>>, <IP version=4 ihl=5
tos=0x0 len=28 id=19027 flags= frag=0 ttl=62 proto=icmp chksum=0x1e81
src=10.0.0.9 dst=10.0.0.5 |<ICMP type=echo-reply code=0 chksum=0xffff
id=0x0 seq=0x0 |>>)
```

Scapy также поддерживает конструирование пакетов для протоколов прикладного уровня, таких как DNS-запросы. В следующем примере мы обращаемся

к публичному DNS-серверу, чтобы получить IP-адрес доменного имени `www.google.com`:

```
>>> p = sr1(IP(dst="8.8.8.8")/UDP()/DNS(rd=1,qd=DNSQR(qname="www.google.com")))
Begin emission:
.....Finished sending 1 packets.
.....*
Received 17 packets, got 1 answers, remaining 0 packets
>>> p
<IP version=4 ihl=5 tos=0x20 len=76 id=17713 flags= frag=0 ttl=121
proto=udp chksum=0x28c5 src=8.8.8.8 dst=192.168.2.211 |<UDP sport=domain
dport=domain len=56 chksum=0xa9db |<DNS id=0 qr=1 opcode=QUERY aa=0
tc=0 rd=1 ra=1 z=0 ad=0 cd=0 rcode=ok qdcount=1 ancount=1 nscount=0
arcount=0 qd=<DNSQR qname='www.google.com.' qtype=A qclass=IN |>
an=<DNSRR rrtype='www.google.com.' type=A rclass=IN ttl=274 rdlen=None
rdata=216.58.217.36 |> ns=None ar=None |>>>
```

Рассмотрим другие возможности Scapy. Начнем с захвата пакетов.

Захват пакетов с помощью Scapy

В процессе устранения неполадок нам, сетевым инженерам, постоянно приходится захватывать пакеты, проходящие через соединение. Обычно для этого используется Wireshark или похожие инструменты, но Scapy тоже позволяет легко выполнять захват пакетов:

```
>>> a = sniff(filter="icmp", count=5)
>>> a.show()
0000 Ether / IP / ICMP 192.168.2.211 > 8.8.8.8 echo-request 0 / Raw
0001 Ether / IP / ICMP 8.8.8.8 > 192.168.2.211 echo-reply 0 / Raw
0002 Ether / IP / ICMP 192.168.2.211 > 8.8.8.8 echo-request 0 / Raw
0003 Ether / IP / ICMP 8.8.8.8 > 192.168.2.211 echo-reply 0 / Raw
0004 Ether / IP / ICMP 192.168.2.211 > 8.8.8.8 echo-request 0 / Raw
```

Можно просматривать пакеты более подробно, даже в исходном формате:

```
>>> for packet in a:
...     print(packet.show())
...
###[ Ethernet ]###
dst= 70:4f:57:94:7f:86
src= 5e:00:00:02:00:00
type= IPv4
###[ IP ]###
version= 4
ihl= 5
tos= 0x0
len= 84
```

```

id= 1856
flags= DF
frag= 0
ttl= 64
proto= icmp
chksum= 0x5fde
src= 192.168.2.211
dst= 8.8.8.8
\options\
###[ ICMP ]###
    type= echo-request
    code= 0
    chksum= 0x4616
    id= 0x4a84
    seq= 0x1
###[ Raw ]###
    load= 'k\x9a\x8f]\x00\x00\x00\x00\xac\x99\x01\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f!#$%&\'()*+,-./01234567'
<опущено>

```

Итак, мы познакомились с основными принципами работы Scapy. Теперь посмотрим, как этот инструмент применяется для проверки безопасности сети.

Сканирование TCP-портов

Любая попытка взлома почти всегда начинается с поиска в сети открытых сервисов; это позволяет атаковать определенную цель. Конечно, чтобы обслуживать наших клиентов, нам необходимо открыть некоторые порты; это риск, который мы должны принять. Вместе с тем мы должны закрыть все остальные порты, которые могли бы послужить дополнительными целями для атак. Мы можем использовать Scapy для сканирования нашего собственного хоста на предмет открытых TCP-портов.

Отправим пакет SYN и посмотрим, вернет ли сервер в ответ SYN-ACK. Начнем с TCP-порта 23, принадлежащего Telnet:

```

>>> p = sr1(IP(dst="10.0.0.9")/TCP(sport=666,dport=23,flags="S"))
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
>>> p.show()
###[ IP ]###
    version= 4
    ihl= 5
    tos= 0x0
    len= 40

```

```

id= 14089
flags= DF
frag= 0
ttl= 62
proto= tcp
chksum= 0xf1b9
src= 10.0.0.9
dst= 10.0.0.5
\options\
###[ TCP ]###
    sport= telnet
    dport= 666
    seq= 0
    ack= 1
    dataofs= 5
    reserved= 0
    flags= RA
    window= 0
    chksum= 0x9911
    urgptr= 0
    options= []

```

Обратите внимание, что в данном случае на попытку установить соединение с TCP-портом 23 сервер ответил пакетом **RESET+ACK**, потому что на хосте нет Telnet. Однако TCP-порт 22 (SSH) открыт, поэтому для него возвращается пакет **SYN-ACK**:

```

>>> p = sr1(IP(dst="10.0.0.9")/TCP(sport=666,dport=22,flags="S"))
>>> p = sr1(IP(dst.show())
###[ IP ]###
    version= 4
<опущено>
    proto= tcp
    chksum= 0x28bf
    src= 10.0.0.9
    dst= 10.0.0.5
    \options\
###[ TCP ]###
    sport= ssh
    dport= 666
    seq= 1671401418
    ack= 1
    dataofs= 6
    reserved= 0
    flags= SA
<опущено>

```

Мы также можем просканировать диапазон портов от 20 до 22; заметьте, что мы отправляем/принимает сразу несколько пакетов, поэтому здесь используется функция `sr()`, а не `sr1()` (которая отправляет/принимает одиночные пакеты):

```
>>> ans,unans = sr(IP(dst="10.0.0.9")/TCP(sport=666,dport=(20,22),flags="S"))
>>> for i in ans:
...     print(i)
...
(<IP frag=0 proto=tcp dst=10.0.0.9 |<TCP sport=666 dport=ftp_data flags=S |>>, <IP version=4 ihl=5 tos=0x0 len=40 id=59720 flags=DF frag=0 ttl=62 proto=tcp chksum=0x3f7a src=10.0.0.9 dst=10.0.0.5 |<TCP sport=ftp_data dport=666 seq=0 ack=1 dataofs=5 reserved=0 flags=RA window=0 chksum=0x9914 urgptr=0 |>>)
(<IP frag=0 proto=tcp dst=10.0.0.9 |<TCP sport=666 dport=ftp flags=S |>>, <IP version=4 ihl=5 tos=0x0 len=40 id=59721 flags=DF frag=0 ttl=62 proto=tcp chksum=0x3f79 src=10.0.0.9 dst=10.0.0.5 |<TCP sport=ftp dport=666 seq=0 ack=1 dataofs=5 reserved=0 flags=RA window=0 chksum=0x9913 urgptr=0 |>>)
(<IP frag=0 proto=tcp dst=10.0.0.9 |<TCP sport=666 dport=ssh flags=S |>>, <IP version=4 ihl=5 tos=0x0 len=44 id=0 flags=DF frag=0 ttl=62 proto=tcp chksum=0x28bf src=10.0.0.9 dst=10.0.0.5 |<TCP sport=ssh dport=666 seq=3932520059 ack=1 dataofs=6 reserved=0 flags=SA window=29200 chksum=0xa666 urgptr=0 options=[('MSS', 1460)] |>>)
>>>
```

Вместо отдельного хоста можно просканировать целую сеть. Как видите, в диапазоне 10.0.0.8/29 ответ SA вернули хосты 10.0.0.9, 10.0.0.10 и 10.0.0.14; это два сетевых устройства и сервер:

```
>>> ans,unans = sr(IP(dst="10.0.0.8/29")/TCP(sport=666,dport=(22),flags="S"))
>>> for i in ans:
...     print(i)
...
(<IP frag=0 proto=tcp dst=10.0.0.14 |<TCP sport=666 dport=ssh flags=S |>>, <IP version=4 ihl=5 tos=0x0 len=44 id=7289 flags= frag=0 ttl=64 proto=tcp chksum=0x4a41 src=10.0.0.14 dst=10.0.0.5 |<TCP sport=ssh dport=666 seq=1652640556 ack=1 dataofs=6 reserved=0 flags=SA window=17292 chksum=0x9029 urgptr=0 options=[('MSS', 1444)] |>>)
(<IP frag=0 proto=tcp dst=10.0.0.9 |<TCP sport=666 dport=ssh flags=S |>>, <IP version=4 ihl=5 tos=0x0 len=44 id=0 flags=DF frag=0 ttl=62 proto=tcp chksum=0x28bf src=10.0.0.9 dst=10.0.0.5 |<TCP sport=ssh dport=666 seq=898054835 ack=1 dataofs=6 reserved=0 flags=SA window=29200 chksum=0x9f0d urgptr=0 options=[('MSS', 1460)] |>>)
(<IP frag=0 proto=tcp dst=10.0.0.10 |<TCP sport=666 dport=ssh flags=S |>>, <IP version=4 ihl=5 tos=0x0 len=44 id=38021 flags= frag=0 ttl=254 proto=tcp chksum=0x1438 src=10.0.0.10 dst=10.0.0.5 |<TCP sport=ssh dport=666 seq=371720489 ack=1 dataofs=6 reserved=0 flags=SA window=4128 chksum=0x5d82 urgptr=0 options=[('MSS', 536)] |>>)
>>>
```

Основываясь на этих результатах, напомним простой сценарий, `scapy_tcp_scan_1.py`, пригодный для многократного использования:

```
#!/usr/bin/env python3
from scapy.all import *
import sys

def tcp_scan(destination, dport):
    ans, unans = sr(IP(dst=destination)/TCP(sport=666,dport=dport,flags="S"))
    for sending, returned in ans:
        if 'SA' in str(returned[TCP].flags):
            return destination + " port " + str(sending[TCP].dport) + " is
open."
        else:
            return destination + " port " + str(sending[TCP].dport) + " is
not open."

def main():
    destination = sys.argv[1]
    port = int(sys.argv[2])
    scan_result = tcp_scan(destination, port)
    print(scan_result)

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

Сначала мы импортируем `scapy` и модуль `sys` для приема аргументов. Функция `tcp_scan()` похожа на код, который мы использовали до сих пор; разница лишь в том, что мы можем получить ввод из аргументов командной строки и затем вызвать `tcp_scan()` внутри `main()`.

Помните, что доступ к низкоуровневой сети требует привилегий суперпользователя, поэтому сценарий нужно запускать с помощью `sudo`. Просканируем с его помощью порты 22 (SSH) и 80 (HTTP):

```
cisco@Client:~$ sudo python3 scapy_tcp_scan_1.py "10.0.0.14" 22
Begin emission:
.....Finished sending 1 packets.
*
Received 7 packets, got 1 answers, remaining 0 packets
10.0.0.14 port 22 is open.

cisco@Client:~$ sudo python3 scapy_tcp_scan_1.py "10.0.0.14" 80
Begin emission:
...Finished sending 1 packets.
*
Received 4 packets, got 1 answers, remaining 0 packets
10.0.0.14 port 80 is not open.
```

Это был относительно длинный пример сценария для сканирования TCP-портов, который показал, чего можно достичь, генерируя собственные пакеты с помощью Scapy. Мы протестировали отдельные шаги в интерактивной оболочке и в конце оформили их в виде простого сценария. Теперь рассмотрим примеры использования Scapy для проверки безопасности.