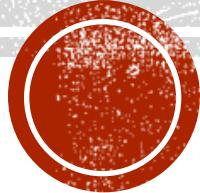


Scapyで作る・解析する パケット

@takahoyo



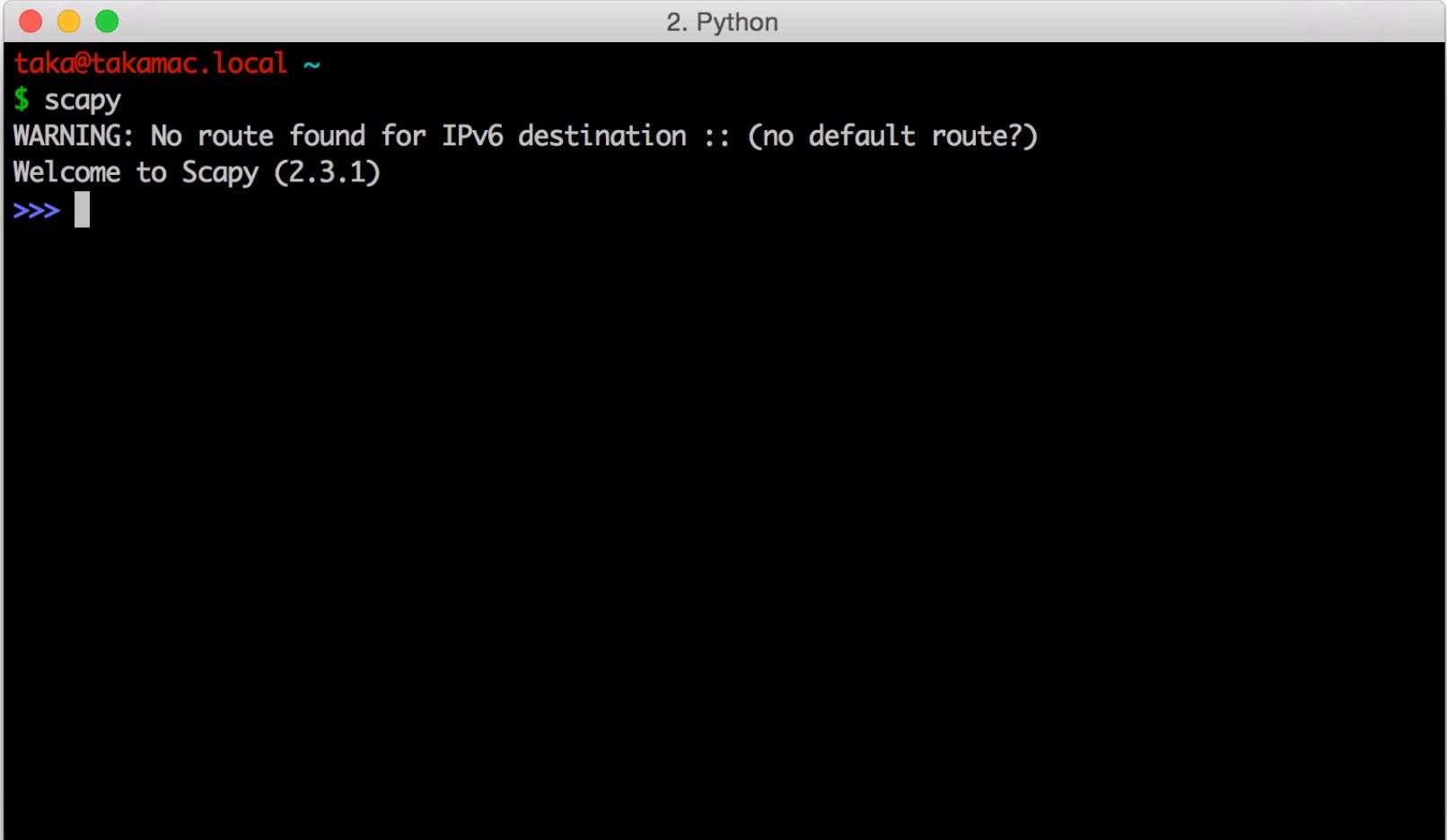
Scapyとは

- Pythonで書かれた対話型のパケット操作プログラム
 - Pythonのライブラリとしてimportして使うことも
 - ドキュメント（英語）が充実している
 - <http://www.secdev.org/projects/scapy/>
 - <http://www.secdev.org/projects/scapy/doc/index.html>
- 何ができる？
 - パケットの生成
 - パケットの送受信
 - pcapファイルの読み込み、書き込み
 - など
- 読み方は、「すけいぴー」「すきやっぴー」？
 - <http://yomikata.org/word/scapy>

インストール

- Python 2系がインストールされてる環境で試した
 - 3系は非公式で対応してるらしい
- Linux
 - Kali Linuxならデフォルトでインストール
 - Ubuntu : `sudo apt-get install python-scapy`
- Mac
 - `brew : brew install libdnet scapy`
- 他のインストール方法について
 - 公式のドキュメントを参照（情報が少し古い）
 - <http://www.secdev.org/projects/scapy/doc/installation.html>

Scapyを起動



taka@takamac.local ~

\$ scapy

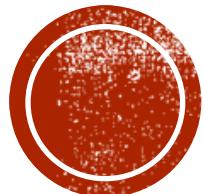
WARNING: No route found for IPv6 destination :: (no default route?)

Welcome to Scapy (2.3.1)

>>> [REDACTED]

Scapyで今回やること

- パケットを作る
- パケットを送る
- パケットを解析する



パケットを作る

パケットを作る

- まずはこういうパケットを作つてみよう



パケットを作る

```
>>> Ether()/IP()/TCP()  
<Ether type=0x800 |<IP frag=0 proto=tcp |<TCP |>>>  
>>> |
```

- プロトコルがクラスとして定義されている
 - Ex. Ethernet → Etherクラス
- “/”を使うことで、複数のレイヤのパケットを作る
 - こういうパケットを作る場合



- Ether()/IP()/TCP()

パケットを作る

- 次は各プロトコルフィールドに値を指定
 - Ethernet
 - Src MAC Addr: 00:00:00:00:00:00
 - Dst MAC Addr: 11:11:11:11:11:11
 - IP
 - Src IP Addr: 192.168.1.1
 - Dst IP Addr: 192.168.1.2
 - TCP
 - Src port: 1234
 - Dst port: 4321
 - Flag: SYN

パケットを作る

```
>>> Ether(src="11:11:11:11:11:11", dst="00:00:00:00:00:00")
<Ether  dst=00:00:00:00:00:00  src=11:11:11:11:11:11 |>
```

- 引数を指定して、フィールドを特定の値で初期化
- フィールドに対応する変数を調べる
 - **ls(クラス名)**
 - Ex. Etherクラス → ls(Ether)

```
>>> ls(Ether)
dst      : DestMACField      = (None)
src      : SourceMACField    = (None)
type     : XShortEnumField   = (36864)
```

パケットを作る

```
>>> Ether(src="11:11:11:11:11:11", dst="00:00:00:00:00:00")/IP(src="192.168.1.1", dst="192.168.1.2")/TCP(sport=1234, dport=4321, flags="S")
<Ether  dst=00:00:00:00:00:00  src=11:11:11:11:11:11  type=0x800 |<IP  frag=0 proto=tcp src=192.168.1.1 dst=192.168.1.2 |<TCP  sport=search_agent dport=rwhois flags=S |>>>
>>>
```

- 前述のパケットを作ると次のようになる

- Ether(src="11:11:11:11:11:11",
dst="00:00:00:00:00:00")/IP(src="192.168.1.1",
dst="192.168.1.2")/TCP(sport=1234, dport=4321, flags="S")

パケットを作る

```
>>> packet = Ether(src="11:11:11:11:11:11", dst="00:00:00:00:00:00")/IP(src="192.168.1.1", ds t="192.168.1.2")/TCP(sport=1234, dport=4321, flags="S")
>>> packet
<Ether  dst=00:00:00:00:00:00  src=11:11:11:11:11:11  type=0x800 |<IP  frag=0 proto=tcp src=192 .168.1.1 dst=192.168.1.2 |<TCP  sport=search_agent dport=rwhois flags=S |>>>
>>> wireshark(packet)
>>> wrpcap('example.pcap', packet)
>>> █
```

- 変数packetにパケットを保存
- 変数packetの中身を表示
- 変数packetの中身をWiresharkで表示
 - wireshark(packet)
- 変数packetの中身をexample.pcapに書き込み
 - wrpcap('example.pcap', packet)

パケットを作る

■ Wiresharkで表示

The screenshot shows the Wireshark interface with a single captured frame. The packet details pane shows:

- Frame 1: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)
- Ethernet II, Src: 11:11:11:11:11:11 (11:11:11:11:11:11), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
- Internet Protocol Version 4, Src: 192.168.1.1 (192.168.1.1), Dst: 192.168.1.2 (192.168.1.2)
- Transmission Control Protocol, Src Port: 1234 (1234), Dst Port: 4321 (4321), Seq: 0, Len: 0
 - Source Port: 1234 (1234)
 - Destination Port: 4321 (4321)
 - [Stream index: 0]
 - [TCP Segment Len: 0]
 - Sequence number: 0
 - Acknowledgment number: 0
 - Header Length: 20 bytes
 - Flags: 0x0002 (SYN)
 - Window size value: 8192
 - [Calculated window size: 8192]
 - Checksum: 0xf6db [validation disabled]
 - Urgent pointer: 0

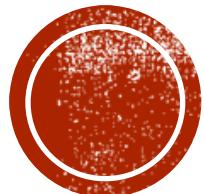
The bytes pane shows the raw hex and ASCII data for the captured frame.

パケットを作る

- ここまで、”Ether/IP/TCP”のパケットを作った
- 他のプロトコルにも対応している
 - `ls()`でどのようなプロトコルが対応しているか表示できる

```
>>> ls()
AH      : AH
ARP     : ARP
ASN1_Packet : None
BOOTP   : BOOTP
CookedLinux : cooked linux
DHCP    : DHCP options
DHCP6   : DHCPv6 Generic Message)
```

- L1~L4のプロトコルであればほぼ対応
- L5~L7は、あまり対応していない雰囲気...



パケットを送る

パケットを送る

- まずパケットを作る
 - “ICMP Echo Request”パケットを作る

```
>>> packet = IP(dst="192.168.8.1")/ICMP(type=8)
>>> packet
<IP frag=0 proto=icmp dst=192.168.8.1 |<ICMP type=echo-request |>
```

- そのパケットを送る
 - **send()**で送信することができる
 - **send()**の場合、L2のプロトコルは自動で調整

```
>>> send(packet)
.
Sent 1 packets.
>>> []
```

パケットを送る

■ Wiresharkで見てみる

The screenshot shows a Wireshark capture window with the following details:

- Interface: Wi-Fi: en0
- Protocol: icmp
- Selected frame: ICMP Echo (ping) request (Frame 11)
- Frame details:
 - Frame 11: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
 - Ethernet II, Src: Apple_0a:3a:58 (80:e6:50:0a:3a:58), Dst: HuaweiTe_d6:d4:3c (24:09:95:d6:d4:3c)
 - Internet Protocol Version 4, Src: 192.168.8.100 (192.168.8.100), Dst: 192.168.8.1 (192.168.8.1)
 - Internet Control Message Protocol
 - Type: 8 (Echo (ping) request)
 - Code: 0
 - Checksum: 0xf7ff [correct]
 - Identifier (BE): 0 (0x0000)
 - Identifier (LE): 0 (0x0000)
 - Sequence number (BE): 0 (0x0000)
 - Sequence number (LE): 0 (0x0000)
 - Response frame: 12
- Hex dump:

0000	24 09 95 d6 d4 3c 80 e6 50 0a 3a 58 08 00 45 00	\$....<.. P:X..E.
0010	00 1c 00 01 00 00 40 01 e9 2a c0 a8 08 64 c0 a8@..*...d..
0020	08 01 08 00 f7 ff 00 00 00 00

パケットを送る

- 送るだけでなく、返ってきたパケットも見る
 - **sr()**を使う

```
>>> sr(packet)
Begin emission:
..Finished to send 1 packets.
.*
Received 4 packets, got 1 answers, remaining 0 packets
(<Results: TCP:0 UDP:0 ICMP:1 Other:0>, <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)
>>> 
```

- **sr1()**を使うと最初に返ってきた一つのパケットだけ見る

```
>>> sr1(packet)
Begin emission:
...Finished to send 1 packets.
.*
Received 5 packets, got 1 answers, remaining 0 packets
<IP version=4L ihl=5L tos=0x0 len=28 id=45266 flags= frag=0L ttl=64 proto=icmp checksum=0x3859
 src=192.168.8.1 dst=192.168.8.100 options=[<ICMP type=echo-reply code=0 checksum=0xffff id
 =0x0 seq=0x0 |>>
>>> 
```

パケットを送る

- 返ってきたパケットを変数に保存する
 - `sr()` や `sr1()` の戻り値は、返ってきたパケットの情報

```
>>> r_packet = sr(packet)
Begin emission:
..Finished to send 1 packets.
.*
Received 4 packets, got 1 answers, remaining 0 packets
>>> r_packet
(<Results: TCP:0 UDP:0 ICMP:1 Other:0>, <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)
>>> r_packet[0][0]
(<IP frag=0 proto=icmp dst=192.168.8.1 |<ICMP type=echo-request |>, <IP version=4L ihl=5L
tos=0x0 len=28 id=45258 flags= frag=0L ttl=64 proto=icmp checksum=0x3861 src=192.168.8.1 dst=1
92.168.8.100 options=□ |<ICMP type=echo-reply code=0 checksum=0xffff id=0x0 seq=0x0 |>>)
>>> █
>>> r_packet1 = sr1(packet)
Begin emission:
...Finished to send 1 packets.
.*
Received 5 packets, got 1 answers, remaining 0 packets
>>> r_packet1
<IP version=4L ihl=5L tos=0x0 len=28 id=45261 flags= frag=0L ttl=64 proto=icmp checksum=0x385e
src=192.168.8.1 dst=192.168.8.100 options=□ |<ICMP type=echo-reply code=0 checksum=0xffff id
=0x0 seq=0x0 |>>
```

パケットを送る

- 他にもいろいろな送信方法

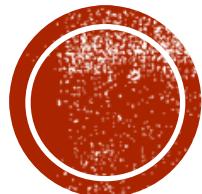
function (引数は初期値, N:None)	説明
send(pkt, count=1, inter=1, iface=N)	L3レベルでのパケット送信
sendp(pkt, count=1, inter=1, iface=N)	L2レベルでのパケット送信
sendfast(pkt, pps=N, mbps=N, iface=N)	tcp replayを使って送るらしい
sr(pkt, filter=N, iface=N) : L3 srp(pkt, filter=N, iface=N) : L2	パケット送信 返答をすべて受信
sr1(pkt, filter=N, iface=N) : L3 srp1(pkt, filter=N, iface=N) : L2	パケット送信 返答を1つだけ受信
srflood(pkt, filter=N, iface=N) : L3 srpflood(pkt, filter=N, iface=N) : L2	パケットを大量に送信(Flood) 返答をすべて受信 (危なそう)

パケット送る

- パケット送信の応用例
- Pythonでデータを処理して送信する
 - JPEGファイルをpythonで読み込む
 - 読み込んだファイルを分割する
 - ICMPのデータ部に分割したデータを入れたパケットを作る
 - そのパケットを送信する
 - <https://www.cloudshark.org/captures/48a2a5e3d98e>
 - このように変なパケットも作ることが出来る
- 応用すればもっと変なパケットを作ることも...

パケットを送る

```
1  #! /usr/bin/env python
2
3  from scapy.all import *
4
5  f = open('flag.jpg', 'rb')
6  jpg = f.read()
7  f.close()
8
9  start = 0
10 end = len(jpg)/40 + 1
11
12 for i in range(1,41):
13     raw = jpg[start:end]
14     packet = (IP(dst="192.168.1.154")/ICMP(id=0x1234, seq=(i-1))/raw)
15
16     sr1(packet)
17
18     start = end
19     end = start + len(jpg)/40 + 1
20
```



パケットを解析する

パケットを解析する

- pcapファイルを読み込む
 - rdpcap()を使う

```
>>> packets = rdpcap('example.pcap')
>>> packets
<example.pcap: TCP:5326 UDP:548 ICMP:13 Other:102>
>>> [
```

- 変数packetsに、 pcapファイル内のパケットが読み込まれる
- packets[n]でn-1番目にあるパケットにアクセスできる

```
>>> packets[0]
<Ether dst=00:0c:29:b1:3d:e8 src=00:0c:29:a4:d3:86 type=0x800 |<IP version=4L ihl=5L tos=0x0 len=84 id=57363 flags=DF frag=0L ttl=64 proto=icmp checksum=0x2f53 src=172.16.105.144 dst=172.16.105.145 options=□ |<ICMP type=echo-request code=0 checksum=0x92ee id=0x268f seq=0x1 |<Raw load='\x1c\x94\xb9U\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*, -./01234567' |>>>
>>> [
```



パケットを解析する

- WiresharkとScapyでパケットを見比べてみる

No.	Time	Source	Destination	Protocol	Length	Info
→ 1	0.000000	172.16.105.144	172.16.105.1...	ICMP	98	Echo (ping) request id=0x268f, seq=1/256, ttl=64 (reply in 2)
← 2	0.000615	172.16.105.145	172.16.105.1...	ICMP	98	Echo (ping) reply id=0x268f, seq=1/256, ttl=64 (request in 1)
3	1.000662	172.16.105.144	172.16.105.1...	ICMP	98	Echo (ping) request id=0x268f, seq=2/512, ttl=64 (reply in 4)
4	1.000856	172.16.105.145	172.16.105.1...	ICMP	98	Echo (ping) reply id=0x268f, seq=2/512, ttl=64 (request in 3)
5	2.000047	172.16.105.144	172.16.105.1...	ICMP	98	Echo (ping) request id=0x268f, seq=3/768, ttl=64 (reply in 6)

```
>>> packets.summary()
Ether / IP / ICMP 172.16.105.144 > 172.16.105.145 echo-request 0 / Raw
Ether / IP / ICMP 172.16.105.145 > 172.16.105.144 echo-reply 0 / Raw
Ether / IP / ICMP 172.16.105.144 > 172.16.105.145 echo-request 0 / Raw
Ether / IP / ICMP 172.16.105.145 > 172.16.105.144 echo-reply 0 / Raw
Ether / IP / ICMP 172.16.105.144 > 172.16.105.145 echo-request 0 / Raw
```

パケットを解析する

- 各フィールドへのアクセス
 - 変数`packets`にパケットが読み込まれている場合
 - `packets[n]['layername'].fieldname`
 - 例1: 1番目のパケットの送信元IPアドレスを表示

```
>>> packets[0]['IP']
<IP version=4L ihl=5L tos=0x0 len=84 id=57363 flags=DF frag=0L ttl=64 proto=icmp checksum=0x2f
53 src=172.16.105.144 dst=172.16.105.145 options= |<ICMP type=echo-request code=0 checksum=0
x92ee id=0x268f seq=0x1 |<Raw load='\x1c\x94\xb9U\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*,-.0123456
7' |>>>
>>> packets[0]['IP'].src
'172.16.105.144'
```

- 例2: 1番目のパケットのICMPデータを表示

パケットを解析する

- 処理を工夫することで様々な処理が可能
 - for文を使った複数のパケットの処理

```
>>> for p in packets:  
...     if p['Ethernet'].type == 0x800:  
...         if p['IP'].proto == 1:  
...             print p['Raw'].load  
...  
    ! "#$%&'()*, -./01234567  
    ! "#$%&'()*, -./01234567  
    ! "#$%&'()*, -./01234567  
    ! "#$%&'()*, -./01234567  
  E ! "#$%&'()*, -./01234567  
  E ! "#$%&'()*, -./01234567  
    ! "#$%&'()*, -./01234567  
    ! "#$%&'()*, -./01234567  
  X ! "#$%&'()*, -./01234567  
  X ! "#$%&'()*, -./01234567
```

パケットを解析する

- これを出来ると何が嬉しい?
 - Wiresharkで手が届かない処理が出来る
- 例.
 - <https://www.cloudshark.org/captures/20532c9a3305>
 - ptunnelのパケット
 - 参考 : <http://mrt-k.hateblo.jp/entry/2014/02/02/205332>
 - TCPなら、Wiresharkで”Follow TCP Stream”を使える
 - ICMPの場合、それに該当する機能がWiresharkにない
 - ScapyならICMPのデータ部分を繋げることが出来る
 - さらにそのデータを処理出来る

パケットを解析する

- まずWiresharkで見てみる

ptunnel.pcap [Wireshark 1.99.9 (SVN Rev Unknown from unknown)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply 保存

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.75.129	192.168.75.136	ICMP	70	Echo (ping) request id=0x2b18, seq=0/0, ttl=64 (reply in)
2	0.000098	192.168.75.129	192.168.75.136	ICMP	156	Echo (ping) request id=0x2b18, seq=1/256, ttl=64 (reply in)
3	0.000030	192.168.75.136	192.168.75.129	ICMP	70	Echo (ping) reply id=0x2b18, seq=0/0, ttl=64 (request in)
4	0.000341	192.168.75.136	192.168.75.129	ICMP	156	Echo (ping) reply id=0x2b18, seq=1/256, ttl=64 (request in)
5	0.001452	192.168.75.136	192.168.75.129	ICMP	70	Echo (ping) reply id=0x2b18, seq=0/0, ttl=64
6	0.001467	192.168.75.136	192.168.75.129	ICMP	1094	Echo (ping) reply id=0x2b18, seq=1/256, ttl=64
7	0.001495	192.168.75.136	192.168.75.129	ICMP	1094	Echo (ping) reply id=0x2b18, seq=2/512, ttl=64
8	0.001591	192.168.75.136	192.168.75.129	ICMP	1094	Echo (ping) reply id=0x2b18, seq=3/768, ttl=64

Frame 2: 156 bytes on wire (1248 bits), 156 bytes captured (1248 bits)
Ethernet II, Src: VMware_a4:d3:86 (00:0c:29:a4:d3:86), Dst: VMware_ce:8b:17 (00:0c:29:ce:8b:17)
Internet Protocol Version 4, Src: 192.168.75.129 (192.168.75.129), Dst: 192.168.75.136 (192.168.75.136)
Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0x74e4 [correct]
Identifier (BE): 11032 (0x2b18)
Identifier (LE): 6187 (0x182b)
Sequence number (BE): 1 (0x0001)
Sequence number (LE): 256 (0x0100)
[Response frame: 4]
Data (114 bytes)

28byteのヘッダ

0000 00 0c 29 ce 8b 17 00 0c 29 a4 d3 86 08 00 ..)....)....E.
0010 00 8e 85 bd 40 00 40 01 9c 57 c0 a8 4b 8a c0 a8@. @. W.K..
0020 4b 88 08 00 74 e4 2b 18 00 01 d5 20 08 80 00 K.t.t.+
0030 00 00 00 00 00 40 00 00 02 00 ff ff 00 04@.
0040 00 56 00 01 2b 18 47 45 54 20 2f 66 6c 61 67 2e .V.+.GE T /flag.
0050 6a 70 67 20 48 54 54 50 2f 31 2e 31 0d 0a 55 73 jpg HTTP /1.1.Us
0060 65 72 2d 41 67 65 6e 74 3a 20 63 75 72 6c 2f 37 er-Agent : curl/7
0070 2e 32 36 2e 30 0d 0a 48 6f 73 74 3a 20 6c 6f 63 26.0..H ost: loc
0080 61 6c 68 6f 73 74 3a 38 30 38 30 0d 0a 41 63 63 alhost:8 080..Acc
0090 65 70 74 3a 20 2a 2f 2a 0d 0a 0d 0a

29byte目以降は
HTTPのデータ

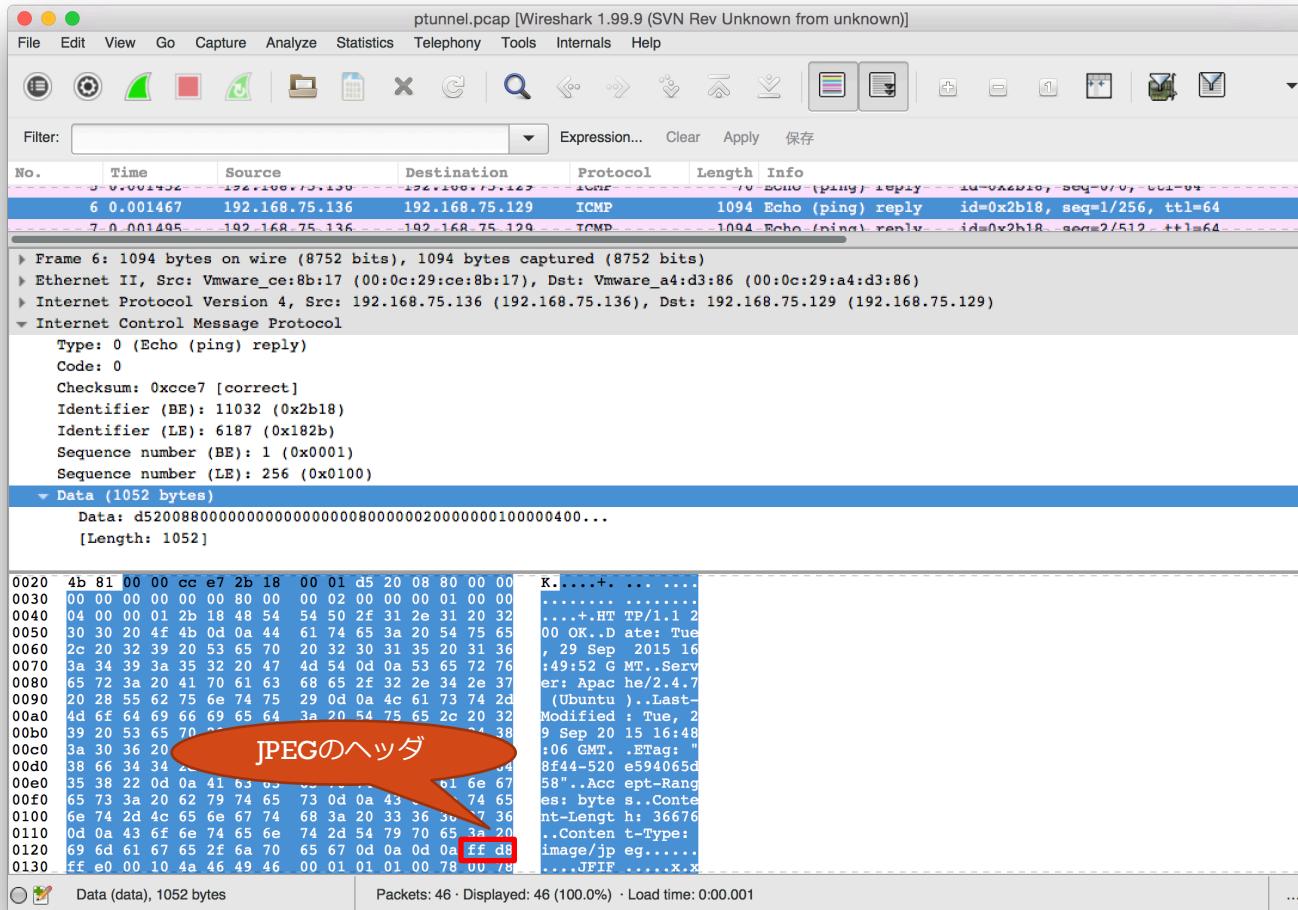
Data (data), 114 bytes

Packets: 46 · Displayed: 46 (100.0%) · Load time: 0:00.001

2015/9/30

パケットを解析する

- まずWiresharkで見てみる



パケットを解析する

- まずWiresharkで見てみる（結果）
 - ICMPのデータ部分にptunnel使ったデータ
 - データの最初の28byteはヘッダ、それ以外はHTTP通信
 - “GET /flag.jpg”からflag.jpgを取得してあるパケットと推測
 - 複数のパケットを見ると、JPEGのデータが確認できる
- Scapy使って何をやるか
 - ICMPのデータを抽出
 - 28byteのヘッダを取り除いて、それ以外のデータをつなげる
 - そのデータからJPEGデータを抽出する
 - pythonなので、これをすべて一緒にできる

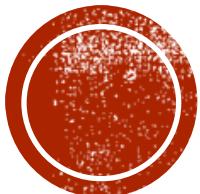
パケットを解析する

```
1  #! /usr/bin/env python
2  #! -*- coding:utf-8 -*-
3
4  from scapy.all import *
5
6  #pcapファイルの読み込み
7  p = rdpcap('ptunnel.pcap')
8
9  data = ""
10 for i in range(len(p)):
11     if p[i]['IP'].src == "192.168.75.136" and p[i]['ICMP'].type == 0:
12         load = p[i]['Raw'].load
13         if len(load) > 28:
14             # ptunnelのヘッダ(28byte)を除いたデータをくっつける
15             data += load[28:]
16
17 #jpgのデータを抜く
18 index = data.find('\xff\xd8')
19 jpg = data[index:]
20
21 #jpgデータをファイルへ書き込み
22 f = open('flag.jpg', 'wb')
23 f.write(jpg)
24 f.close()
```

パケットを解析する

- ちなみに...
- `packets[n].time`でタイムスタンプにアクセスできる

```
>>> packets[1].time  
1438225436.564559  
>>> packets[0].time  
1438225436.563944  
>>> packets[1].time = packets[0].time  
>>> packets[1].time  
1438225436.563944  
>>> |
```



まとめ

Scapyのメリット

- 対話型であること
 - 手軽に使える、試せる
- Pythonであること
 - 他の便利なライブラリと一緒に使える
 - Python最高
- 他のソフトウェアとの連携
 - Wireshark
 - nmap
 - p0f
- 他のツールよりも柔軟に処理が出来る

Scapyのデメリット

- インストールがやや面倒くさい
- Python使えないとつらい
- 開発が止まってる？
- L5~L7レイヤに対応してくれたらなあ...
- Pythonなのでやや遅い
 - その辺りはCで実装されたツールが良いかも
- PcapNgに対応してない
 - editcapで変換すれば良いが...

参考資料

- Scapyの公式ドキュメント
 - <http://www.secdev.org/projects/scapy/doc/index.html>
- Scapy Cheat Sheet
 - <http://packetlife.net/media/library/36/scapy.pdf>
- scapy でソケット通信
 - <http://nigaky.hatenablog.com/entry/20110716/1310813250>
- Scapyでのパケットの扱い方
 - http://mrt-k.github.io/scapy_nw/2015/02/16/Scapy%E3%81%A7%E3%81%AE%E3%83%91%E3%82%B1%E3%83%83%E3%83%88%E3%81%AE%E6%89%B1%E3%81%84%E6%96%B9/
- Scapy presentation
 - <http://www.slideshare.net/reonnishimura5/scapy-presentation>