Computer Networking

COMP 177 | Fall 2020 | University of the Pacific | Jeff Shafer

Scapy

Recap

Past Topics

- Overview of networking and layered architecture
- Wireshark packet sniffer
- Ethernet and WiFi
- IPv4

Today's Topics

- **Scapy**
  - Packet crafting for Python2 and Python3

What is Scapy?

- Scapy is a Python-based packet manipulation framework
- Using Scapy you can
  - Sniff packets passively
  - Dissect the sniffed packets into different headers and fields
  - Forge packets by editing different fields
  - Send packets to the destination
- Scapy can be installed on all major operating systems

Working with Scapy (Linux)

- Scapy can be used in two different ways:
  - As an interactive programming tool
  - As a library in a larger program
- After installation, you can simply run Scapy in interactive mode as follows: `$ scapy`
- In order to send/receive packets however, you should run Scapy with root privileges: `$ sudo scapy`
- In the interactive mode, you can
  - Define new packets
  - Read/write fields within packets
  - Send and receive packets
  - Read pcap files
  - Visualize the packets

Working with Scapy

- In order to make a packet of certain protocol, it suffices to call the function that corresponds to that protocol
  - `Ether()`, `IP()`, `ICMP()`, `TCP()`, etc.
- Calling these functions without any parameters populates the fields with default values
- The IP packet returned by invoking `IP()` has
  - `127.0.0.1` as both the source and destination addresses
  - `1` as the identifier
  - `0` as the fragmentation offset
- The Ethernet frame returned by invoking `Ether()` has
  - `00:00:00:00:00:00` as the source MAC address
  - `ff:ff:ff:ff:ff:ff` as the destination MAC address

Setting Protocol Fields

- You can set protocol fields in two different ways:
  - As parameters upon invoking the protocol function
    - `IP(src='192.168.1.1')`
    - `Ether(dst='43:63:a4:7f:18:01')`
    - `IP(src='192.168.1.1', ihl=5)`
  - Naming the packet, and then updating the fields under that name
    - ```
      a = Ether()
      a.dst = '43:63:a4:7f:18:01'
      ```
    - ```
      b=IP()
      b.src = '192.168.1.1'
      b.ihl = 5
      ```
  - You can also update an already set field to Scapy's default with the `del()` function
    - ```
      a = IP(src='192.168.1.1')
      del(a.src)
      ```
      A's `src` field will be restored to the default address `127.0.0.1`
      ```

Packet Encapsulation

- In order to encapsulate a packet within another packet the **/** operator is used – Scapy calls it the "layer" operator
  - `Ether()/IP()` returns an Ethernet frame that encapsulates an IP datagram. The fields in Ethernet and IP headers would be Scapy's default
  - `Ether(dst='43:63:a4:7f:18:01')/IP(ihl=5)` returns
    - Ethernet frame with destination MAC address `43:63:a4:7f:18:01`
    - IP payload where the IP header length is `5`
- Encapsulation also changes the fields values appropriately that reflect on encapsulation
  - In both `Ether()/IP()` and `Ether(dst='43:63:a4:7f:18:01')/IP(ihl=5)`
  - Ethernet `type` would be updated to `0x0800` which shows that the payload is an IP datagram
  - IP datagram `proto` fields still has the default value, since IP datagram does not have any payload.

Packet Encapsulation

- Upon encapsulating a packet within another, you can still refer to fields with `.<field-name>`
  - `(Ether()/IP()).ttl` will return the TTL value from IP header
  - `(Ether()/IP()).type` will return the type value from Ethernet header

Packet Encapsulation

- Question: What if two fields have the same name in two different protocol? For example:
  - Both source MAC address and source IP address are named `src`
  - Both destination MAC address and destination IP address are named `dst`
- Answer: Referring by field name returns the value from the *outer* header
  - `(Ether()/IP()).src` returns the source MAC address `00:00:00:00:00:00`

Packet Encapsulation

- Question: Then how can we refer inner header fields?
- Answer: By explicitly mentioning the protocol within brackets
  - `(Ether()/IP())[IP].src` returns IP address `127.0.0.1`

Useful Functions

- **`raw(pkt)`** returns the raw byte string of `pkt`. This raw byte string can be fed to each protocol function to populate fields
- **`hexdump(pkt)`** returns a hexadecimal dump of the packet. The output is similar to Wireshark's *packet bytes* section
- **`ls(pkt)`** lists all fields and their values within all headers of `pkt`. It also shows the data type that Scapy has defined for each field
- **`pkt.summary()`** shows a summary of the packet in one line
- **`pkt.show()`** lists all fields and their values within all headers of `pkt`
- **`pkt.show2()`** is similar to `pkt.show()` but displays the final packet (e.g. checksum fields are calculated)
- **`pkt.command()`** returns Scapy command as a string by which `pkt` can be generated

https://scapy.readthedocs.io/en/latest/usage.html

Generating List of Packets

- Giving multiple values to a field generates a list of packet, each with one value from that field
- Assigning list(s) of values
  - `IP(id=[3,10])` generates a list consisting of two IP packets, one with `id=3` and the other with `id=10`
  - `IP(id=[3,10], ttl=[78,45])` generates a list of four IP packets with all options for `id` and `ttl` fields
- Assigning a range of values
  - `IP(id=(3,8))` generates six IP datagrams with different ids ranging from `3` to `8`
- Assigning IP addresses with prefix lengths
  - `IP(dst = '192.168.12.0/30')` generates four IP packets with the destination IP addresses: 192.168.12.0, 192.168.12.1, 192.168.12.2, 192.168.12.3

Sending Packets

- `send(pkt)` sends *network layer* packet(s)
  - `send(IP(dst='192.168.1.1/30'))`
  - Scapy choses an appropriate interface and link layer protocol
- `sendp(pkt)` sends *data link layer* packet(s)
  - `sendp(Ether(dst='11:22:33:44:55:66')/IP(dst="4.2.2.3"))`
  - You can be explicit about the interface using `iface`, e.g., `sendp(Ether(dst='11:22:33:44:55:66')/IP(dst="4.2.2.3"), iface='eth0')`
- You can use `loop=1` to send packets indefinitely
  - `send(IP(dst='192.168.1.1'), loop=1)`
- You can use `inter` to set an interval in seconds between sending each packet
  - `send(IP(dst='192.168.1.1'), loop=1, inter=0.5)`

Sending and Receiving Packets

- `sr1(pkt)` sends network layer packet(s) and returns the *first* received packet in response
  - `sr1(IP(dst='192.168.1.1'))`
- `srp(pkt)` sends data link layer packet(s) and returns the *first* received packet in response
  - `srp(Ether()/IP(dst='192.168.1.1/28'))`
- `sr(pkt)` sends network layer packet(s) and returns two lists
  - List 1: Pairs of sent and answered packets
  - List 2: Packets that are sent but are unanswered
  - `x,y=sr(IP(dst='192.168.1.1'))`, where x consists of pairs of sent and answered packets, and y is the list of unanswered packets

Sending and Receiving Packets

- Use `timeout` to set the seconds before timing out to receive responses
  - `sr1(IP(dst='192.168.1.1'), timeout=3)`
- Use `retry=n` to retry sending the unanswered packets *n* times
  - `sr1(IP(dst='192.168.1.1'), retry=5)`
- Use `retry=-n` to retry to send the unanswered packets for *n* times in row with no answer for *any* of those packets
  - `sr1(IP(dst='192.168.1.1'), retry=-5)`

Closing Thoughts

Recap

- Today we discussed scapy
  - How to build packets of different protocols
  - How to set values to fields
  - Some useful functions to apply to packets
  - How to generate lists of packets
  - How to send and receive packets

Next Class

- Address Resolution Protocol (ARP)

Class Activity

CA.6 – Scapy

Due tonight at 11:59pm

COMING NEXT WEEK!!

Project 1

Due Sept 30th at 11:59pm