

TCPEXposure – A handy TCP experiment tool

Michio Honda, Keio University
micchie@sfc.wide.ad.jp

Outline

- Background
- Our TCP experiment libraries
- Functionality abstraction
- Crafting a TCP segment
- Send and receive TCP segments
- Print packets
- Example to describe TCP experiments
- The responder tool – a stateless TCP responder

Background

- Originally a middlebox measurement tool consisting of the initiator and the responder
 - The initiator transmits test TCP traffic to the responder
 - We measured 142 paths with contributors
- Its bottom-half libraries are useful for a range of TCP experiments that require customized TCP traffic
 - e.g., TCP packet with the undefined option, and TCP packet with the inconsistent sequence
- Our responder, a stateless TCP responder is also useful to interact with the test traffic
 - e.g., respond to the undefined TCP option with the same option
 - We would use public web servers as responders depending on the test

Our TCP experiment libraries

- All-in-one tarball is available at
 - <http://www.micchie.net/software.html>
- Send and receive raw TCP packets
- Run in Mac OSX, Linux and FreeBSD
- Require only python 2.5 or higher and libpcap (tcpdump relies on this) – pre-installed in most of the end systems
- Why don't we use existing tools like scapy?
 - Buggy, hard to customize, and require installing itself and additional packages – hard to widely distribute
 - Some platform (e.g., Planetlab) doesn't allow sending packet via Pcap
 - Our tool can use both Pcap and Raw socket

Functional Abstraction

- *pcaplib.py*
 - libpcap wrapper of Python
- *tcpsrplib.py*
 - *Raw packet creation method*
 - Output/input methods of raw packets on a raw IP socket or a Bpf device
- *tcplib.py*
 - TCP header structures and constants

Crafting a TCP segment

```
pkt = tcpsrlib.make_segment(daddr, saddr, dport, sport, awnd, seqno, ackno,\  
    tcpflags, options=options, payload=payload, ipcksum=ipcksum)
```

- *daddr* and *saddr* are destination and source address
 - host byte order values
- *dport* and *sport* are a destination and a source port
- *awnd* is the advertising window value
- *seqno* and *ackno* are the sequence and the ACK numbers, respectively
- *tcpflags* is the TCP flags (defined in *tcplib.py*)
 - E.g., (for SYN-ACK, *tcplib.TH_SYN* | *tcplib.TH_ACK*)
- (optional) *options* is TCP options (See next page for formatting)
- (optional) *payload* is payload (must be packed by struct library)
- (optional) *ipcksum* is if the segment includes checksum (0 or 1)
 - Checksum is required to transmit the packet via a Bpf device



See Slides xxx for actual examples

TCP Options

- Each option is described by name and values
 - `compose_options()` and `decompose_options()` in `tcpsrplib.py` use options described in this format
 - 512 Byte MSS: ('MSS', 512)
 - SACK_OK: ('SACKOK')
 - TIMESTAMP (TS_val: 12345, TS_ecr: 0): ('TIMESTAMP', 12345, 0)
 - When we give above options to a composing packet,

```
pkt = tcpsrplib.make_segment(....., \  
    options=(('MSS', 512), ('SACKOK',), ('TIMESTAMP', 12345, 0)), ....)
```

- For new options, add codes in `compose_options()` and `decompose_options()` in `tcpsrplib.py`, and add codes in `tcplib.py` (see the other option implementations, it's easy)

Send and receive TCP segments

```
rcvpkts, err = tcpsrlib.sendrecv_segments(daddr, pkts, timeout=timeout,\  
                                           sflags=sflags, usepcap=usepcap, ifname=ifname, \  
                                           smacaddr=smacaddr, dmacaddr=dmacaddr)
```

- *daddr* is the destination address (host byte order value)
- *pkts* is a tuple consisting of composed packets
- Below arguments are optional
 - *timeout* specifies seconds (floating-point value) to wait for response packets
 - *sflags*=*tcplib.TH_SYN* allows us to return immediately after receiving SYN-ACK
 - *usepcap* is whether we send packets via a Bpf device (if 0, send packets with a raw IP socket)
- To use a Bpf device, following arguments must be given
 - *ifname* specifies the interface name (e.g., en0)
 - *smacaddr* and *dmacaddr* are the source and the destination MAC address

Send and receive TCP segments (cont.)

- TCPExposure recognizes packets with the opposite source-destination address/port pair as the responding packets
- A tuple of responding packets and an error value are returned from `sendrecv_segments()` described before
 - Error value is 0 in normal, -1 in fatal error
 - Each responding packet includes the IP and the TCP header
- Packets are extracted to strings by `summarize_ansl()`
 - See the next slide

Print packets

- `summarize_pkt(pkt)`
 - `pkt` is a composed packet by `make_segment()`
 - Return string formatted packet (similar to `tcpdump` output)
 - So, to print packet information in your script:
 - `print tcpsrplib.summarize_pkt(pkt)`
- `summarize_ansl(ansl)`
 - `ansl` is a single element (1 packet) in the tuple returned from `sendrecv_segments()`
 - Return string formatted packet (similar to `tcpdump` output)
 - So, to print packet information in your script (`ans` is a tuple returned from `sendrecv_segments()`):

```
for ansl in ans:
```

```
    print summarize_ansl(ansl)
```

Packet Abstraction

- Receiving packet is a tuple of an iphdr class instance, a tcphdr class instance, a tuple of TCP options, and payload
 - iphdr class and tcphdr class is defined in tcplib.py
 - When the packet is ansI (one of elements in the returned tuple from sendrecv_segments()), the sequence number field can be referred as ansI[I].seqno
 - The advertising window field can be referred as ansI[I].window

Traffic Craft Example

- See sample code (example.py) in our files (downloadable from <http://www.micchie.net/software.html>)
- This sample performs 3WHS and transmits one data segment (use a Bpf device) to `vinson2.sfc.wide.ad.jp`
 - Rewrite that in the code to test the other responder (e.g., a public web server)

The Responder Tool and The Others

- Our files include 2 files in addition to our libraries described
 - tcpexposure.py
 - Middlebox test implementation
 - Require tcplib.py, tcpsrplib.py, pcaplib.py
 - mptcp-server.py
 - Our stateless TCP responder implementation
 - e.g., send a SYN-ACK packet to the SYN packet, acknowledge the receiving sequence number
 - Require tcplib.py
 - They implement some commands (see our IMC paper)