User space code to control NFQUEUE

Linux kernel apply operations to a packet according to **iptables** rules. NFQUEUE is a type of iptables rule target.

Here is an setup.sh of adding iptables rules to put a matching packet into NFQUEUE with specific number.

```
#!/bin/bash
server = $1
client = $2
iptables -t filter -F
iptables -A INPUT -p udp -s client -d server -j NFQUEUE --queue-num 0
iptables -A OUTPUT -p udp -s server -d client -j NFQUEUE --queue-num 0
```

This script takes two arguments as input, a server and a client ip address. It first reset the iptable and then for all the udp packets comes from client with destination server, it is enqueued to queue 0. It is also the same for the packet which comes from the server and goes to client.

To run this script:

```
sudo ./setup.sh <server ip> <client ip>
```

I'm not an expert in linux kernel and I just know that the packet queue is a chained list. The element of the list is the packet and metadata. I learned from the internet that this queue is a fixed length queue. Each packet in this queue has a index. User space code can have a handle to control how to deal with this packet. Once user space verdict issued, the packet is released.

We can use **libnetfilter_queue** to enforce user space code to control the actions on packets in NFQueue.

Here is the preparation steps:

```
int main(int argc, char **argv) {
        struct nfq_handle *nfqHandle;
        struct nfq_q_handle *myQueue;
        struct nfnl_handle *netlinkHandle;

        int fd, res;
        char buf[PACKET_MAX_LEN];

        // create a handler
        if (!(nfqHandle = nfq_open())) {
                fprintf(stderr, "Error in create a handler\n");

                exit(-1);
```

```
        }

        // Unbind the handler
        if (nfq_unbind_pf(nfqHandle, AF_INET) < 0) {
                fprintf(stderr, "Error in unbind\n");
                exit(1);
        }

        // Bind this handler
        if (nfq_bind_pf(nfqHandle, AF_INET) < 0) {
                fprintf(stderr, "Error in nfq_bind_pf()\n");
                exit(1);
        }

        // Install a callback on queue 0, we should implement a Callback
        // function somewhere else to deal with the packet
        if (!(myQueue = nfq_create_queue(nfqHandle,  0, &Callback, NULL))) {
                fprintf(stderr, "Error in nfq_create_queue()\n");
                exit(1);
        }

        // Turn on packet copy mode
        if (nfq_set_mode(myQueue, NFQNL_COPY_PACKET, 0xffff) < 0) {
                fprintf(stderr, "Could not set packet copy mode\n");
                exit(1);
        }

        netlinkHandle = nfq_nfnlh(nfqHandle);
        fd = nfnl_fd(netlinkHandle);

        // End here, the preparation finished.

        /*
          Get packet from the queue
          deal with it
        */

        nfq_destroy_queue(myQueue);

        nfq_close(nfqHandle);

        return 0;
}
```

The focus is how can we deal with the packet?

- To drop a packet or accept a packet
- To disorder the sequence of a packet
- To modify the content of a packet

How to drop a packet or accept a packet? By implementing Callback function and issue the verdict.

```
int main(int argc, char** argv) {
  /*
   * Preparation steps
   */

  /* Get packet from the queue and deal with it*/
  while ((res = recv(fd, buf, sizeof(buf), 0)) && res >= 0) {
    nfq_handle_packet(nfqHandle, buf, res);
  }

  return 0;
}
```

Here we get a packet into **buf**. By calling **nfq_handle_packet** , we deal with the packet in our **Callback** function.

```
static int Callback(struct nfq_q_handle *myQueue, struct nfgenmsg *msg,
                struct nfq_data *pkt, void *cbData) {
  unsigned int id = 0;
  struct nfqnl_msg_packet_hdr *header;

  // Get the id of the packets in the queue
  if ((header = nfq_get_msg_packet_hdr(pkt)))
    id = ntohl(header->packet_id);

  // to drop the packet
  nfq_set_verdict(myQueue, id, NF_DROP, 0, NULL);
  // to accept the packet
  // nfq_set_verdict(myQueue, id, NF_ACCEPT, len, pktData);

}
```

How to disorder a packet? I try a simple method in the main function. I just hold the current packet in somewhere else and then delay issue the verdict for it.

```
int main(int argc, char** argv) {
  /*
   * Preparation steps
   */

  /* Get packet from the queue and deal with it */
  int hold = 0;
  int reshold = -1;
  char bufhold[PACKET_MAX_LEN];

  while ((res = recv(fd, buf, sizeof(buf), 0)) && res >= 0) {

    if (hold == 0) {
```

```
      double holdratio = ((double) rand()) / ((double) RAND_MAX);
      if (holdratio > 0.5) {
        printf("delay current packet\n");
        hold = 1;
        reshold = res;
        memcpy(bufhold, buf, sizeof(buf));
      } else {
        printf("process current packet\n");
        nfq_handle_packet(nfqHandle, buf, res);
      }
    } else {
      // first process current packet
      printf("process current packet\n");
      nfq_handle_packet(nfqHandle, buf, res);
      // deal with hold packet
      double releaseratio = ((double) rand()) / ((double) RAND_MAX);
      if (releaseratio > 0.5) {
        printf("process hold packet\n");
        nfq_handle_packet(nfqHandle, bufhold, reshold);
        hold = 0;
        reshold = -1;
        memset(bufhold, 0, PACKET_MAX_LEN);
      }
    }
  }
}
```

For each packet I get from linux kernel, I first check whether there is packet which is held in **bufhold**. If there is no packet held before, I will randomly hold the current packet and delay to deal with it. If there is a packet held before, I first deal with the current packet, then randomly deal with the held packet.

How to modify the content of a packet? As linux kernel pass a copy of the packet to user space. We can just modify the copy. However we can deliver the modified packet when we issue verdict to linux kernel. The notice is that we need to re-compute the packet checksum. In this example, if we modify the application data, we need to re-compute the udp checksum as well as ip checksum.

```
static int Callback(struct nfq_q_handle *myQueue, struct nfgenmsg *msg,
                    struct nfq_data *pkt, void *cbData) {
  unsigned int id = 0;
  struct nfqnl_msg_packet_hdr *header;

  // Get the id of the packets in the queue
  if ((header = nfq_get_msg_packet_hdr(pkt)))
    id = ntohl(header->packet_id);

  // Read the packet content
  unsigned char *pktData;
  int len = nfq_get_payload(pkt, (char**)&pktData); // ip datagram len

  // Create the pointer to ip header

  struct iphdr *ipHeader = (struct iphdr *) (pktData);
```

```
  unsigned short ipcheck = ipHeader->check;
  printf("original ip checksum = %04x\n", ipcheck);

  // Create pointer to udp header
  struct udphdr *udpHeader = (struct udphdr *)(((unsigned char *) ipHeader) + ipHeader-
>ihl * 4);
  unsigned short udpcheck = udpHeader->check;
  printf("original udp checksum = %04x\n", ipcheck);

  // Read the application data in udp packet
  unsigned char *appData;
  appData = pktData + ipHeader->ihl * 4 + 8;

  // try to modify application data
  // here I modify the first 4 Bytes in appData
  unsigned int idx = 100;
  idx = htonl(idx);
  memcpy(appData, (char*)&idx, 4);

  // recalculate the udp checksum and ipchecksum
  // udp header : source port(2Bytes) dst port (2 Bytes)
  //              len (2Bytes) check(2Bytes)
  // We need to implement compute_udp_checksum by ourself

  // recalculate the ip checksum

  nfq_set_verdict(myQueue, id, NF_ACCEPT, len, pktData);
}
```

I think this is more like a experiment records more than a technical blog. There is still much basic knowledge behind my experiment.

Resources:

- Basic Knowledge
- Checksum Computation