Networking - Assignment 3

Packet Filtering and Encryption

Johan IJsveld & Rick van der Zwet
<jijsveld@xs4all.nl> & <hvdzwet@liacs.nl>

LIACS
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

29 april 2009

1 Implementation

We have chosen for a single emphC program without any arguments to keep it simple, meaning for the encryption key *0x12345678* will be hard-coded into the program and queue number are hard-coded *0* for input and *1* for output.

2 Global working

At first a package arrives at *netfilter* which is a Linux packet filter implementation. This will push the packet using callback function implemented at in*libnetfilter_queue* to our implementation. let's call our application *xorfilter* for a moment. The xorfilter preforms it's 'magic' which will be in this case a mangle of the data payload using a XOR computation with the key above. Next it recalculates the TCP checksum to ensure the checksum matches the altered payload, leading to proper validation of the packet on the other end.

3 Usage xorfilter

At first make sure to enable the *xorfilter*, this needed to be running under the root user, due to the enhanced privileges it needs at start by 'hooking' into *iptables*. Next alter *iptables* using the CLI script, for a reference sample, check *setqueue.sh*.

4 Usage echod

Ones started the program *echod* will listen at port *12345*. It's only purpose it serves (hences the 'd' of daemon in echod) is to reply the data it receives. One could connect using the program telnet to the program using *telnet localhost 12345* and everything you type will be returned as well.

5 xorfilter encryption

The key consist of 4 bytes, so for every 4 bytes, the data is XORed with the key, which is first translated using *htonl* to make sure we are using big endian instead of small endian as all network traffic is big endian by definition. Next in case of a remainder -when the length of the data is not a power of 4 bytes- the remainder is XORed with the relevant byte of the key. Example:

```
   DATA: AB CD EF 34 23 21 34 (7 bytes)
Step 1 (key repeated each 4 bytes, done using for loop on words):
   KEY : 12 34 56 78
   IMM : B9 F9 B9 4C 23 21 34
Step 2 (remainder key, done using for loop on bytes):
   KEY :             12 34 56
   IMM : B9 F9 B9 4C 31 15 62
```

Appendix

echod (1 file), xorfilter (9 files), netfilter CLI example (2 files)

```
/*  Netlab 2009 - group 1 - assignment 3
 *  Johan IJsveld
 *  Rick van der Zwet
 */
#include <sys/types.h>
#include <netdb.h>
#include <sysexits.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>

#define LISTEN_BACKLOG 0
#define BUF_SIZE 1000


#include <signal.h>


int main() {
  int conn_fd;
  char buf[BUF_SIZE];
  ssize_t recv_size;
  int retno;

  /* Create a tcp/ip socket */
  int mysocket = socket(PF_INET, SOCK_STREAM, 0);

  struct sockaddr_in my_addr;
  struct sockaddr_in their_addr;
  int sin_size;


  // ip(7) Create listen port, NOTE addresses in byte-order BYTEORDER(3)
  my_addr.sin_family = AF_INET; // By default
  my_addr.sin_port  = htons(12345);
  my_addr.sin_addr.s_addr  = htonl(INADDR_ANY);
  memset(my_addr.sin_zero, '\0', sizeof my_addr.sin_zero);

  // get connected
  retno = bind(mysocket, (struct sockaddr *) &my_addr, sizeof(struct sockaddr_in));
  retno = listen(mysocket, LISTEN_BACKLOG);
  sin_size = sizeof(struct sockaddr_in);

  // Make it a server, e.g. never ending
  while(1) {
    conn_fd = accept(mysocket, (struct sockaddr *)&their_addr, &sin_size);
    if (fork() == 0) {
      // child will help to echo the stuff
      while(1) {
      retno = recv(conn_fd, &buf, BUF_SIZE, 0);
      printf("%s\n", buf);
      if (retno <= 0)
        break;
      retno = send(conn_fd, &buf, retno, 0);
      if (retno <= 0)
        break;
      }
      close(conn_fd);
    }
  }
  close(mysocket);


return EX_OK;
}
```

```
#include "checksum.h"

long checksum( unsigned short *addr, int count )
{
    /* Compute Internet Checksum for "count" bytes
     *         beginning at location "addr".
     */
    long checksum;
    register long sum = 0;
    while( count > 1 ) {
        /* This  is the inner loop */
        sum += * (unsigned short *) addr++;
        count -= 2;
    }

    /* Add left-over byte, if any */
    if( count > 0 )
        sum += * (unsigned char *) addr;

    /* Fold 32-bit sum to 16 bits */
    while (sum>>16)
        sum = (sum & 0xffff) + (sum >> 16);
    checksum = ~sum;
    return checksum;
}
```

```
#ifndef CHECKSUM_H
#define CHECKSUM_H

/* Compute Internet Checksum for "count" bytes
 *         beginning at location "addr".
 */
long checksum( unsigned short *addr, int count );

#endif
```

```
#include "filter.h"
#include "xorencrypt.h"
#include "checksum.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <netinet/in.h>
#include <linux/types.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/netfilter.h>        /* for NF_ACCEPT */
#include <arpa/inet.h>
#include <inttypes.h>

const unsigned int key = 0x12345678;    /* The key used for encrypting */

/* Returns the IP header from the datagram
 */
struct iphdr *get_ip_hdr(char *datagram)
{
    return (struct iphdr *)datagram;
}

/* Returns the TCP header from the datagram
 */
struct tcphdr *get_tcp_hdr(char *datagram)
{
    char *tcpdata = datagram + get_ip_len(datagram);
    return (struct tcphdr *)tcpdata;
}

/* Returns pointer to TCP payload data from the datagram
 */
char *get_tcp_payload(char *datagram)
{
    struct tcphdr *tcphdr = get_tcp_hdr(datagram);
    int iphdr_len = get_ip_len(datagram);
    int tcphdr_len = get_tcp_len(tcphdr);
    return datagram + iphdr_len + tcphdr_len;
}

/* Returns the total length of the datagram
 */
__u16 get_ip_tot_len(struct iphdr *ip)
{
    return ntohs(ip->tot_len);
}

/* Returns the total length of the IP header
```

```
 */
__u16 get_ip_len(char *datagram)
{
    struct iphdr *iphdr = get_ip_hdr(datagram);
    return iphdr->ihl * 4; /* ihl is in words */
}

/* Returns the length of the IP options field
 */
__u16 get_ip_opt_len(char *datagram)
{
    return get_ip_len(datagram) - sizeof(struct iphdr);
}

/* Returns the total length of the TCP header
 */
__u16 get_tcp_len(struct tcphdr *tcphdr)
{
    return tcphdr->doff * 4; /* data offset is in words */
}

/* Returns the length of the TCP options field
 */
__u16 get_tcp_opt_len(struct tcphdr *hdr)
{
    return get_tcp_len(hdr) - sizeof(struct tcphdr);
}

/* Returns the length in bytes of the tcp payload data
 */
__u16 get_tcp_data_len(char *datagram)
{
    struct iphdr *iphdr = get_ip_hdr(datagram);
    struct tcphdr *tcphdr = get_tcp_hdr(datagram);
    int total_len = get_ip_tot_len(iphdr);
    int iphdr_len = get_ip_len(datagram);
    int tcphdr_len = get_tcp_len(tcphdr);
    return total_len - iphdr_len - tcphdr_len;
}

/* Returns the source address */
__u16 get_tcp_source(struct tcphdr *hdr)
{
    return ntohs(hdr->source);
}

/* Returns the destination address */
__u16 get_tcp_dest(struct tcphdr *hdr)
{
    return ntohs(hdr->dest);
}

/* the pseudo header used in calculating checksum */
struct pseudo_hdr {
    __u32 source;
    __u32 dest;
    __u8 zero;
    __u8 ptcl;
    __u16 tcplen;
};

void print_datagram(char *datagram)
{
    struct iphdr *iphdr = get_ip_hdr(datagram);
    struct tcphdr *tcphdr = get_tcp_hdr(datagram);
    unsigned int iphdr_len = get_ip_len(datagram);
    unsigned int tcphdr_len = get_tcp_len(tcphdr);
    unsigned int total_len = get_ip_tot_len(iphdr);
    unsigned int data_len = get_tcp_data_len(datagram);
    char *data = get_tcp_payload(datagram);
   printf("datagram length: %d, iphdr length: %d, tcphdr length: %d, data length: %d, data
pointer: %p, sizeof(struct pseude_hdr): %d\n",
           total_len, iphdr_len, tcphdr_len, data_len, data, sizeof(struct pseudo_hdr));
}


/* Encrypts TCP payload data using XOR in the datagram pointed to by datagram
 */
int encrypt_payload(char *datagram)
{
    struct iphdr *iphdr = get_ip_hdr(datagram);
    struct tcphdr *tcphdr = get_tcp_hdr(datagram);
    unsigned int tcphdr_len = get_tcp_len(tcphdr);
    unsigned int total_len = get_ip_tot_len(iphdr);
    unsigned int data_len = get_tcp_data_len(datagram);
    char *data = get_tcp_payload(datagram);
    struct pseudo_hdr *phdr;
    long crc;

    print_datagram(datagram);
    if (iphdr->protocol != 6) {
        printf("not a tcp datagram\n");
        return total_len;
    }

    /*  memset(data, 0, data_len); */
    /* xor payload data */
    printf("Before crypt: '%s'\n", data);
    xorencrypt(data, data_len, htonl(key));
    printf("After crypt: '%s'\n", data);


    printf("crc before: %d\n", tcphdr->check);
    /* calculate new crc value by making new datagram with pseudo header */
    tcphdr->check = 0;
    char *copy = malloc(sizeof(struct pseudo_hdr) + tcphdr_len + data_len);

    /* set pseudo header */
    phdr = (struct pseudo_hdr *)copy;
    phdr->source = iphdr->saddr;
    phdr->dest = iphdr->daddr;
    phdr->zero = 0;
    phdr->ptcl = iphdr->protocol;
    phdr->tcplen = tcphdr_len + data_len;
    phdr->tcplen = htons(phdr->tcplen);

    /* copy tcp header minus options */
    memcpy(copy + sizeof(struct pseudo_hdr), (char *)tcphdr, tcphdr_len);
    /* copy payload data */
    memcpy(copy + sizeof(struct pseudo_hdr) + tcphdr_len, data, data_len);

    /* calculate and set new crc */
     crc = checksum((unsigned short *)copy, sizeof(struct pseudo_hdr) + tcphdr
data_len);
    free(copy);

    tcphdr->check = (__u16)crc;
    printf("crc after: %d\n", tcphdr->check);
    return total_len;
}
```

```
#ifndef FILTER_H
#define FILTER_H

#include <linux/types.h>

/* Returns the IP header from the datagram */
struct iphdr *get_ip_hdr(char *datagram);

/* Returns the TCP header from the datagram */
struct tcphdr *get_tcp_hdr(char *datagram);

/* Returns pointer to TCP payload data */
char *get_tcp_payload(char *datagram);

/* Returns the total length of the datagram */
__u16 get_ip_tot_len(struct iphdr *ip);

/* Returns the total length of the IP header */
__u16 get_ip_len(char *datagram);

/* Returns the length of the IP options field */
__u16 get_ip_opt_len(char *datagram);

/* Returns the total length of the TCP header */
__u16 get_tcp_len(struct tcphdr *tcphdr);

/* Returns the length of the TCP options field  */
__u16 get_tcp_opt_len(struct tcphdr *hdr);

/* Returns the length in bytes of the tcp payload data */
__u16 get_tcp_data_len(char *datagram);

/* Returns the source address */
__u16 get_tcp_source(struct tcphdr *hdr);

/* Returns the destination address */
__u16 get_tcp_dest(struct tcphdr *hdr);

/* Ecrypts TCP payload data using XOR in the datagram pointed to by data */
int encrypt_payload(char *datagram);

#endif
```

```
#include "queue.h"

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <linux/types.h>
#include <linux/netfilter.h>           /* for NF_ACCEPT */

#include <libnetfilter_queue/libnetfilter_queue.h>

#define MAX_QUEUES   3

/* data for queue's */
struct queue_t {
        struct nfq_q_handle *qh; /* handle to the queue */
    callback cbp; /* callback function to handle packets from this queue */
    int n;
};

/* the queue's that are setup */
static struct queue_t queues[MAX_QUEUES];

```

```
026: /* The main netlib handle and file descriptor */
027: static struct nfq_handle *h = NULL;
028: static int fd;
029:
030: /*
031:         struct nfnl_handle *nh;
032: */
033:
034: void init()
035: {
036:     printf("opening library handle\n");
037:         h = nfq_open();
038:     if (!h) {
039:         fprintf(stderr, "error during nfq_open()\n");
040:         exit(1);
041:         }
042:
043:         printf("unbinding existing nf_queue handler for AF_INET (if any)\n");
044:     if (nfq_unbind_pf(h, AF_INET) < 0) {
045:         fprintf(stderr, "error during nfq_unbind_pf()\n");
046:            exit(1);
047:     }
048:
049:         printf("binding nfnetlink_queue as nf_queue handler for AF_INET\n");
050:         if (nfq_bind_pf(h, AF_INET) < 0) {
051:                 fprintf(stderr, "error during nfq_bind_pf()\n");
052:         exit(1);
053:     }
054:
055:         fd = nfq_fd(h);
056: }
057:
058: void deinit()
059: {
060:         printf("closing library handle\n");
061:         nfq_close(h);
062: }
063:
064: /* callback called by netfilter_queue when packet is available */
065: static int cb(struct nfq_q_handle *qh, struct nfgenmsg *nfmsg, struct nfq_data *nfa,
void *data)
066: {
067:     char *datagram;
068:     int n = *(int *)data;
069:         u_int32_t id = print_pkt(nfa);
070:     u_int32_t verdict;
071:     int len;
072:
073:         printf("entering nfq callback\n");
074:
075:     if (nfq_get_payload(nfa, &datagram) < 0) {
076:         fprintf(stderr, "error could not get payload data\n");
077:         exit(1);
078:     }
079:     verdict = queues[n].cbp(datagram, &len);
080:
081:         return nfq_set_verdict(qh, id, verdict, len, (unsigned char *)datagram);
082: }
083:
084: void setup_queue(int n, callback cbp)
085: {
086:     if (n >= MAX_QUEUES) {
087:         fprintf(stderr, "error not enough queues\n");
088:         exit(1);
089:     }
090:
091:         printf("binding this socket to queue '%d'\n", n);
092:     queues[n].n = n;
093:         queues[n].qh = nfq_create_queue(h,  n, &cb, &(queues[n].n));
094:         if (!queues[n].qh) {
095:                 fprintf(stderr, "error during nfq_create_queue()\n");
096:                 exit(1);
097:         }
098:
099:         printf("setting copy_packet mode\n");
100:         if (nfq_set_mode(queues[n].qh, NFQNL_COPY_PACKET, 0xffff) < 0) {
101:                 fprintf(stderr, "can't set packet_copy mode\n");
102:                 exit(1);
103:         }
104:     queues[n].cbp = cbp;
105: }
106:
107: void close_queue(int n)
108: {
109:     if (n >= MAX_QUEUES) {
110:         fprintf(stderr, "error not enough queues\n");
111:         exit(1);
112:     }
113:
114:         printf("unbinding from queue %d\n", n);
115:         nfq_destroy_queue(queues[n].qh);
116:
117: #ifdef INSANE
118:         /* normally, applications SHOULD NOT issue this command, since
119:          * it detaches other programs/sockets from AF_INET, too ! */
120:         printf("unbinding from AF_INET\n");
121:         nfq_unbind_pf(h, AF_INET);
122: #endif
123: }
124:
```

```
125: void receive_data()
126: {
127:         int rv;
128:         char buf[66000] __attribute__ ((aligned));
129:
130:         while ((rv = recv(fd, buf, sizeof(buf), 0)) && rv >= 0) {
131:                 printf("\n--- pkt received ---\n");
132:                 nfq_handle_packet(h, buf, rv);
133:         }
134: }
135:
136: /* Print packet and return packet id */
137: u_int32_t print_pkt(struct nfq_data *tb)
138: {
139:         int id = 0;
140:         struct nfqnl_msg_packet_hdr *ph;
141:         struct nfqnl_msg_packet_hw *hwph;
142:         u_int32_t mark, ifi;
143:         int ret;
144:         char *data;
145:
146:         ph = nfq_get_msg_packet_hdr(tb);
147:         if (ph) {
148:                 id = ntohl(ph->packet_id);
149:                 printf("hw_protocol=0x%04x hook=%u id=%u ",
150:                         ntohs(ph->hw_protocol), ph->hook, id);
151:         }
152:
153:         hwph = nfq_get_packet_hw(tb);
154:         if (hwph) {
155:                 int i, hlen = ntohs(hwph->hw_addrlen);
156:
157:                 printf("hw_src_addr=");
158:                 for (i = 0; i < hlen-1; i++)
159:                         printf("%02x:", hwph->hw_addr[i]);
160:                 printf("%02x ", hwph->hw_addr[hlen-1]);
161:         }
162:
163:         mark = nfq_get_nfmark(tb);
164:         if (mark)
165:                 printf("mark=%u ", mark);
166:
167:         ifi = nfq_get_indev(tb);
168:         if (ifi)
169:                 printf("indev=%u ", ifi);
170:
171:         ifi = nfq_get_outdev(tb);
172:         if (ifi)
173:                 printf("outdev=%u ", ifi);
174:         ifi = nfq_get_physindev(tb);
175:         if (ifi)
176:                 printf("physindev=%u ", ifi);
177:
178:         ifi = nfq_get_physoutdev(tb);
179:         if (ifi)
180:                 printf("physoutdev=%u ", ifi);
181:
182:         ret = nfq_get_payload(tb, &data);
183:         if (ret >= 0)
184:                 printf("payload_len=%d ", ret);
185:
186:         fputc('\n', stdout);
187:
188:         return id;
189: }
190:
```

```
01: #ifndef QUEUE_H
02: #define QUEUE_h
03:
04: #include <netinet/in.h>
05: #include <linux/netfilter.h>            /* for NF_ACCEPT */
06:
07: #include <libnetfilter_queue/libnetfilter_queue.h>
08:
09: /* callback receives a datagram and returns a verdict (either NF_ACCEPT or NF_DRO
10: typedef u_int32_t (*callback)(char *datagram, int *len);
11:
12: /* initialize netfilter */
13: void init();
14:
15: /* deinitialize netfilter */
16: void deinit();
17:
18: /* bind to queue n */
19: void setup_queue(int n, callback cbp);
20:
21: /* unbind from queue n */
22: void close_queue(int n);
23:
24: /* start receiving packets */
25: void receive_data();
26:
27: /* Print packet and return packet id */
28: u_int32_t print_pkt (struct nfq_data *tb);
29:
30: #endif
31:
```

```
01: #include "xorencrypt.h"
```

```
02:
03: typedef unsigned int key_t;
04:
05: /* Encrypt data of len bytes with XOR and key */
06: void xorencrypt(void *data, int len, key_t key)
07: {
08:     int i;
09:     key_t *word_data = (key_t *)data;
10:     int word_len = len / sizeof(key_t);
11:     int remainder = len % sizeof(key_t);
12:
13:     for (i = 0; i < word_len; ++i) {
14:         printf("%i\n", i );
15:         word_data[i] ^= key;
16:     }
17:
18:     for (i = len - remainder; i < len; ++i) {
19:         ((unsigned char *)data)[i] ^= ((unsigned char*)&key)[i];
20:     }
21: }
22:
```

```
1: #ifndef XORENCRYPT_H
2: #define XORENCRYPT_H
3:
4: /* Encrypt data of len bytes with XOR and key */
5: void xorencrypt(void *data, int len, unsigned int key);
6:
7: #endif
8:
```

```
01: #include <stdio.h>
02: #include <string.h>
03: #include "queue.h"
04: #include "filter.h"
05:
06: #include "xorencrypt.h"
07:
08: static u_int32_t handle_input(char *datagram, int *len)
09: {
10:     printf("encrypting input payload\n");
11:     *len = encrypt_payload(datagram);
12:     return NF_ACCEPT;
13: }
14:
15: static u_int32_t handle_output(char *datagram, int *len)
16: {
17:     printf("encrypting output payload\n");
18:     *len = encrypt_payload(datagram);
19:     return NF_ACCEPT;
20: }
21:
22: int main(void)
23: {
24:     init();
25:
26:     setup_queue(1, handle_input);
27:     setup_queue(2, handle_output);
28:
29:     receive_data();
30:
31:     close_queue(1);
32:     close_queue(2);
33:
34:     deinit();
35:
36:     return 0;
37: }
38:
```

```
1: #!/bin/sh
2: /sbin/iptables -t mangle -A INPUT -i lo -j NFQUEUE --queue-num 0
3: /sbin/iptables -t mangle -A OUTPUT -o lo -j NFQUEUE --queue-num 1
4:
5: /sbin/iptables -t mangle -A INPUT -i eth0 -j NFQUEUE --queue-num 0
6: /sbin/iptables -t mangle -A OUTPUT -o eth0 -j NFQUEUE --queue-num 1
7:
8: #/sbin/iptables -t mangle -A INPUT -i eth0 -d 10.211.55.4 -j NFQUEUE --queue-num 1
9: #/sbin/iptables -t mangle -A OUTPUT -o eth0 -s 10.211.55.4 -j NFQUEUE --queue-num 2
```

```
1: #!/bin/sh
2: /sbin/iptables -F
3: /sbin/iptables -t mangle -F
4:
```